Ordered Tree Compression

Habibeche Salah Eddine
Evolutionary Engineering and
Distributed Information Systems Laboratory
Department of Computer Science
Djillali Liabes University of Sidi Bel Abbes
Algeria
Email: habibechesalaheddine1@gmail.com

Farah Ben-Naoum
Evolutionary Engineering and
Distributed Information Systems Laboratory
Department of Computer Science
Djillali Liabes University of Sidi Bel Abbes
Algeria
Email: farahbennaoum@yahoo.fr

***Abstract*—Ordered trees are trees in which the left to right order among siblings is important. Trees shows repetitions in their intern structure, using this redundancy, this paper seek to reduce the complexity of tree structures by merging its isomorphic subtrees producing a directed acyclic graph without losing data**

***Index Terms*—Ordered tree, Tree reduction, Tree compression, directed acyclic graph (DAG), Tree-isomorphism.**

I. Introduction

Trees are among the most important nonlinear structure arising in computer science. They appear in many fields to represent hierarchical data such as XML data formats, genealogical studies [1], computer vision [2], pattern recognition [3], programming compilation and natural language processing [4].

The main problem with tree structure is the number of vertices when it became more and more important this influences complexity of algorithms and makes them ineffective in face of massive data. However,tree structure is characterized by some internal repetitions of structures. This notion is closely related to the definition of self-similarity in natural trees. Hence, the problem of tree compression seek the elimination of the structural redundancy appearing in trees. For this, similar parts in a tree are condensed, resulting in a directed acyclic graph (DAG). This avoids unnecessary duplication of different instances of the same graphical object and allows to manipulate efficiently massive data.

In this paper we focused on the particular class of ordered labeled trees which are trees whose nodes are labeled and in which the order among siblings is significant. Thus we consider the problem of ordered tree compression with no loss of information we then introduce and study in next section some definitions and make some notational conventions. In Section 3, we will use tree isomorphism to identify identical tree patterns in an ordered tree. Based on this and on the previous definitions and their properties, we will present an algorithm able to compress an ordered tree into a DAG and its reverse version that allows the recovering of the initial tree.

II. Preliminaries and notations

A. Trees

A directed graph $G = (V, E)$ consists of a set V of vertices, a set of edges E, each edge is represented by an ordered pair of vertices (x, y) where x is called the parent of y and y the child of x. The set of children of x is represented by $child(x)$ and n_x denote the size of $child(x)$. For any $k \in \{1..n_x\}$ x_k represents a child of x. A vertex x is called an ancestor of an other vertex w and w is called a descendant of x if there exists a sequence of vertices $(x_1, x_2, ..., x_n)$, called a path, such that $x_1 = x$ and $x_n = w$, and for each consecutive pair of vertices $(x_i\ ,\ x_{i+1})$, x_i is the parent of x_{i+1}. The ancestor-descendant relation on the set of vertices of a directed graph is a partial order relation on the set of its vertices denoted by $\leq$.

The order (i.e., number of vertices) of a graph $G = (V, E)$ is denoted by $|G|$ and the number of vertices in a subset of vertices $X \subseteq V$ is denoted by $|X|$. A rooted tree is then an acyclic, connected, directed graph $T = (V, E)$ in which every vertex except one (the root r) has exactly one parent vertex: the root has no parent. A node with no children is a leaf and otherwise an internal node. $T[x]$ denotes the subtree of T rooted at x which contains all the descendants of x, and the empty tree and the empty graph are both denoted by ϕ = (ϕ, ϕ).

The depth of a node $v \in V$, $depth(v)$, is the number of edges on the path from v to $root(T)$. The in-degree of a node v, $deg(v)$ is the number of children of v. We extend these definitions such that $depth(T)$ and $deg(T)$ denotes the maximum depth and degree, respectively, of any node in T. The height of a node $v \in V$, $h(v)$, is the maximum number of the edges on the path from v to any leaf in T.

An ordered tree is a pair $(T, S_{\preceq})$ where T is a rooted tree and $S_{\preceq}$=$\{\preceq_x$,x$\in$T$\}$ is a set of total order relations such that for any vertex x of T $\preceq_x$ is a total order relation on $child(x)$ [5].

A directed acyclic graph (DAG) is a graph containing no directed cycle (but which may contain cycles). In a DAG, the ancestor relationship is a partial order relation [5].

B. Tree isomorphism

Let us consider two rooted ordered trees, $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$. A bijection ψ from V_1 to V_2 is a tree isomorphism if for each $(x, y) \in E_1$, $(\psi(x), \psi(y)) \in E_2$ and for each x_i, x_j from $child(x)$ with $x_i \preceq x_j$ we have $\psi(x_i) \preceq \psi(x_j)$ where $\preceq$ is a total order relation on $child(x)$. For each $x \in V$, let $c(x)$ denote the equivalence class of x.

If there exists an isomorphism between two structures T_1 and T_2, the two structures are identical up to a relabeling of their components. In this case, we write $T_1 \equiv T_2$. We show in Fig.1 an example on such trees relation.

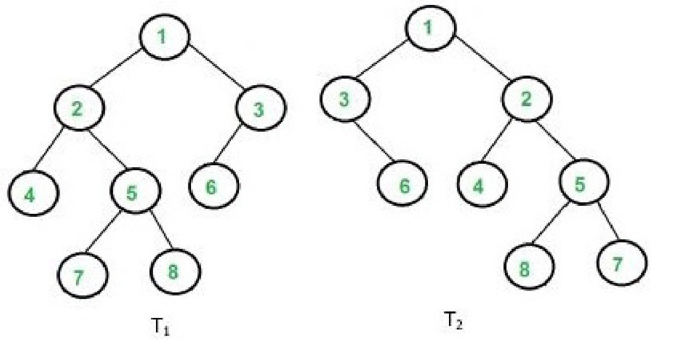

Fig. 1. $T_1[1]$ and $T_2[1]$ are not isomorphic, while $T_1[2]$ and $T_2[2]$ are isomorphic sub-trees.

C. Signature of a vertex

Godin and Ferraro defined in [6] the signature of a vertex $x \in T$ for unordered trees as a multiset $\sigma(x) = \{(J, n(x, J)), J \in Q(T)\}$ where $Q(T)$ is a quotient graph obtained from T using equivalence classes of the isomorphism relation on the set of T subtrees and $n(x, J)$ the number of children of x that have class J.

It's obvious that a multiset defines the vertex signature in an unordered tree since there is no order between sibling. However, ordered trees require some order between the children of a vertex thus we propose an adaptation of this definition for our approach applied to ordered trees.

Definition 1. *(Signature of a vertex) The signature of a vertex in an ordered tree is defined as an ordered set of his children's signatures for the root and internal vertices and $\{\theta\}$ for a leaf.*

Signatures recursively characterize vertices which have identical equivalence classes, based on the signatures of their children. We use $\sigma(x)$ to denote the signature of the vertex x. Fig.2 shows an example of an ordered tree with its set of vertices signatures, while Proposition.1 gives the fundamental characteristic of tree isomorphism on which rests the ordered tree reduction.

Proposition 1. $\forall x, y \in T, T[x] \equiv T[y] \iff \sigma(x) = \sigma(y)$.

Proof. a) Let us consider two isomorphic subtrees $T[x]$ and $T[y]$, in other terms $c(x) = c(y)$. Then by definition $\sigma(x) = \sigma(y)$.

b) Let us consider two vertices x and y in a tree T such that $\sigma(x) = \sigma(y)$. This implies that $|child(x)| = |child(y)|$, and $\forall x' \in child(x)$, $\forall y' \in child(y)$, where x', y' are of the same order in respectively $child(x)$ and $child(y)$, $T[x']$ and

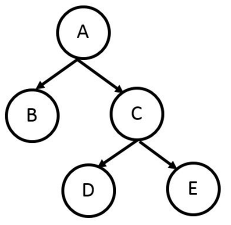

Fig. 2. An ordered tree. $\sigma(B) = \sigma(D) = \sigma(E) = \{\theta\}$ labeled 0, $\sigma(C) = \{0, 0\}$ labeled 1, $\sigma(A) = \{0, 1\}$ labeled 2.

$T[y']$ are isomorphic. Thus $T[x]$ and $T[y]$ are isomorphic as well. □

D. Overview on tree reduction

The aim of the tree reduction is to transform a rooted tree T into a directed acyclic graph denoted $D = (V, E)$. Each vertex of D corresponds to an equivalence class of T subtrees. All subtrees belonging to the same class are isomorphic. The reduction of a tree may contain the same information as the initial tree, and may also allow the recovering of the initial tree.

The problem of constructing a compression of a tree has been raised early. Godin and Ferraro [6] proposed two algorithms for rooted unordered trees, the first one based on vertex signature and computes the reduction of an unordered tree in time $O(|T|^2 deg(T) log(deg(T)))$. The second algorithm use edit distance to detect isomorphic subtrees in a bottom-up manner. The main idea is to compute the distance from a tree T to itself. It's natural that the resulting distance is null, but as a by-product, all the distances between any two subtrees of T are computed recursively in close to quadratic time. Since a null distance between two subtrees denotes the existence of an isomorphism between the two structures. Given the complexity of the edit distance computing algorithms, the complexity of this solution is also important and comes up to $O(|T|^2 deg(T) log(deg(T)))$ for the used Zhangs algorithm [7].

In an other proposition [8] authors went far from a compression of tree in width to a higher compression in both width and height by merging isomorphic structures in width as in the preview work then introducing new techniques for compressing substructure in the resulting DAG that are not exactly isomorphic as they are nested within each-other. Authors present a formal way to create DAGs with return edges that compress in hight the resulting DAG $G(V, E)$ in $O(|E|)$ time.

III. Computing Ordered tree reduction

A. Computing vertex signature algorithm

The signature of a vertex as it was defined as the succession of the children signatures is built recursively in a bottom-up manner. Function *add_signatures* in Alg.1 computes the list of signatures of all T vertices.

Property 1. *Ordered tree vertices signatures can be computed by function $add\_signatures$ in worst case time $O(n^2)$ where $n = |T|$.*

Algorithm 1: Vertices signature computation.

```
Data: Ordered Tree $T = (V, E)$
Result: List of signatures of $T$ vertices
Function $add\_signatures(T)$;
foreach leaf $l \in V$ do
    $l.signature = []$;
    $l.signaturelabel = 0$;
$list\_signature = [[]]$;
$n\_signature = 1$;
$N$:=the list of leafs from $V$;
while $N$ is not empty do
    foreach vertex $n \in N$ do
        foreach vertex $m \in child(n)$ do
            $n.signature+ = [m.signaturelabel]$;
        if $(list\_signature.contain(n.signature))$;
        then
            $n.signaturelabel = list\_signature[n.signature]$;
        else
            $list\_signature+ = [n.signature]$;
            $n.signaturelabel = n\_signature$;
            $n\_signature+ = 1$;
    empty $N$ ;
    add to $N$ all not signed vertices for which all children have a signature;
return $list\_signature$;
```

B. Ordered Tree reduction algorithm

Let us now consider the equivalence relation defined by the tree isomorphism on the set of (complete) sub-trees of a tree $T = (V, E)$ as defined in Property.1.

The essential step in the reduction is to build a quotient graph $G_Q = (V_Q, E_Q)$ from $T(V, E)$ using the equivalence relation on T vertices, V_Q is the set of equivalence classes I on V, and E_Q is a set of pairs of equivalence classes such that $(I, J) \in E_Q$ if and only if $\exists(x, y) \in E$ and $class(x) = I$, $class(y) = J$.

For the example in Fig.2, $V_Q = \{0, 1, 2\}$ and $E_Q = \{(class(A), class(B)), (class(A), class(C)), (class(C), class(D))\} = \{(2, 1), (2, 0), (1, 0)\}$. The the resulting quotient graph $G_Q = (V_Q, E_Q)$ is shown in Fig.3.

A quotient graph G_Q obviously condenses the structural contained in the tree T. However, for each vertex the children order is very important in ordered trees and G_Q does not contain this information. We thus consider how to augment the definition of G_Q so that the resulting condensed representation can be used to reconstruct the original tree. For this, we shall associate an ordered list to each edge of E_Q as seen in Def.2.

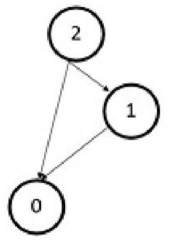

Fig. 3. The quotient graph corresponding to tree of Fig.2.

Definition 2. *In the ordered list $\{a_1, a_2..a_n\}$ associated to an edge (I, J) and denoted by $l_{(I,J)}$, each a_i represents the order of a child of class J among the other sibling in the initial tree T.*

To illustrate this definition, starting from the quotient graph of Fig.3, we present in Fig.4 the complete reduction of tree of Fig.2 including the ordered lists associated to each edge.

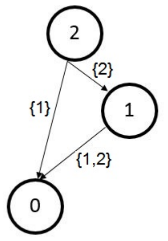

Fig. 4. The Tree reduction of tree of Fig.2.

Before presenting Alg.2, the method computing the reduction of an ordered tree based on the vertices signatures and on the quotient graph construction, let us formally define the resulting reduction graph.

Definition 3. *Given an ordered tree T and its quotient graph $G_Q = (V_Q, E_Q)$ regarding the isomorphism relation on T, we define its reduction graph $R(T)$ as a DAG $D = (node\_list, edge\_list)$ in which $node\_list = V_Q$ and $edge\_list = \{(e, l_e), \forall e \in E_Q\}$.*

Property 2. *The Tree reduction algorithm compress an ordered tree in the worst case time of $O(n^2)$ where $n = |T|$.*

Fig.5 summarizes steps for computing a tree reduction of an ordered tree through an example.

The DAG contains practically all informations contained in the initial tree T that allow its reconstitution from its compression $R(T)$. Alg.3 presents a recursive algorithm for the tree reconstitution.

Property 3. *The Ordered Tree reconstruction is performed in the worst case time of $O(|R(T)| \times deg(R(T))$.*

C. Properties of tree reduction

We notice that the tree and its reduction are equivalent and contain exactly the same data.

Algorithm 2: Tree reduction algorithm.

```
Data: Ordered Tree $T = (V, E)$
Result: DAG $D = (node\_list, edge\_list)$
$node\_list = []$;
$edge\_list = []$;
$list\_signatures = add\_signatures(T)$;
foreach $node\ x \in V$ do
    $n = new\,Node(x.signaturelabel)$ ;
    if $(!node\_list.contain(n))$;
    then
        $node\_list+ = [n]$;
    else
        _
foreach $node\ x \in V$ do
    for $i = 1;\ i \leq child(x);\ i = i + 1$ do
        $e = newedge$;
        $e.begin = x.signaturelabel$;
        $e.end = child(i).signaturelabel$;
        if $(edge\_list.contain(e))$;
        then
            $edge\_list[e].list+ = [i]$;
        else
            $e.list = [i]$;
            $edge\_list+ = e$;
return $node\_list, edge\_list$;
```

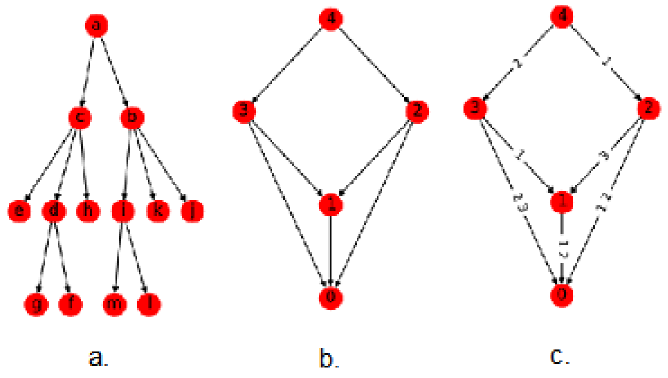

Fig. 5. a. An ordered rooted tree T of vertices signatures: $\sigma(e) = \sigma(f) = \sigma(g) = \sigma(h) = \sigma(k) = \sigma(j) = \{\theta\}$ labeled 0, $\sigma(d) = \sigma(i) = \{0, 0\}$ labeled 1, $\sigma(c) = \{0, 1, 0\}$ labeled 3, $\sigma(b) = \{1, 0, 0\}$ labeled 2 and $\sigma(a) = \{3, 2\}$ labeled 4, b. The quotient graph formed of equivalence classes where each class gather a set of vertices of a same signature: signature of label 0 for class $\{g, f, m, l\}$, 1 for $\{d, i\}$, 2 for $\{d\}$, 3 for $\{c\}$, and 4 for $\{a\}$, c. reduction graph $R(T)$

Property 4. *Any tree T can be recovered from its reduction $R(T)$.*

Proof. The proof is obvious from Alg.2 computing the reduction of a tree T and Alg.3 performing its reconstruction. □

Recall that, for any node I in $R(T)$, there exists a vertex x in T such that $I = c(x)$.

Algorithm 3: Tree Reconstitution.

```
Data: DAG $D = (V, E)$
Result: Tress $T = (node\_list, edge\_list)$
Function $Reconstitution(Dag\_node\ m, Tree\_node\ father)$:;
$node\_list = []$;
$edge\_list = []$;
if $(m\ is\ a\ leaf)$;
then
    create $Tree\_node\ n$;
    $node\_list.add(n)$;
else
    create $Tree\_node\ n$;
    $node\_list.add(n)$;
    foreach $M_i \in child(M)$ do
        foreach $i \in e(M, M_i).list()$ do
            create $Tree\_edge$
            $e_t$=$(n, Reconstitution(M_i, n, i))$;
            $edge\_list.add(e_t)$;
```

Property 5. *Any tree and its reduction have the same height.*

Proof. Let T be a tree and $R(T)$ its reduction. Since the proposed compression is performed in width, in such a way that for any vertex x in T their exists a unique node I in $R(T)$ with $I = c(x)$ and for which $h(I) = h(x)$. Then $h(T) = h(R(T))$. □

Many quantities that characterize a tree T can be directly computed on it's reduction $R(T)$, thus we define:

- $n(I) = |T[x]|$ the size of the tree rooted in x as

$$n(I) = 1 + \sum_{J \in child(I)} |l_{(I,J)}|.n(J) \tag{1}$$

Note that for a leaf of T, $child(I) = \phi$, and then $n(I) = 1$.

- The number of isomorphic subtrees in T rooted in x. for any I in $R(T)$:

$$m(I) = |\{x \in T | c(x) = I\}| \tag{2}$$

if $parent(I) = \phi$ then

$$m(I) = 1 \tag{3}$$

if $parent(I) \neq \phi$ then

$$m(I) = 1 + \sum_{k \in parent(I)} |l_{(K,I)}|.m(K) \tag{4}$$

- For a tree $T(V, E)$ and it's compression $R(T) = (V_r, E_r)$, the vertex compression factor is defined by

$$\rho_v = 1 - \frac{|V_r|}{|V|} \tag{5}$$

- Similarly, the edge compression factor is defined by:

$$\rho_e = 1 - \frac{|E|}{|E|-1} \quad (6)$$

IV. Conclusion

This paper has considered the problem of reduction of ordered trees where the trees are compressed in width using internal repetitions of their structures. Isomorphic subtrees are replaced by a unique class in the reduction. The efficiency of the compression algorithm is quantified by a compression factor reflecting the ratio between the size of the DAG and the size of the initial Tree. The new challenge is to adapt the solution for a new class of trees as semi-ordered trees and apply the solution for a real problem in particular the field of biology and plant biology.

References

[1] Zhang, Kaizhong and Shasha, Dennis, "Simple fast algorithms for the editing distance between trees and related problemss," SIAM journal on computing, vol. 18,number 6, pp. 1245–1262,SIAM, 1989.

[2] Frank Y Shih, "Object representation and recognition using mathematical morphology model," In: Journal of Systems Integration, 1.2, pp. 235-256, 1991.

[3] Frank Y Shih and Owen Robert Mitchell, "Threshold decomposition of gray-scale morphology into binary morphology," In: IEEE Transactions on Pattern Analysis and Machine Intelligence, 11.1, pp. 31-42, 1989.

[4] M Chodorow and JL Klavansl, "Locating syntactic patterns in text corpora,"In: Manuscript, Lexical systems, IBM Research, TJ Watson Research, Center, Yorktown Heights, New York,1990.

[5] F. Preparata and R. Yehl, "Introduction to discrete structures for computer science and engineering,"Reading Menlo Park London: Addison-Wesley,1973

[6] Godin, Christophe and Ferraro, Pascal, "Quantifying the degree of self-nestedness of trees: application to the structural analysis of plants,'IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB),vol 7,number 4,pp 688-703,IEEE Computer Society Press,2010

[7] Zhang Kaizhong, "Algorithms for the constrained editing distance between ordered labeled trees and related problems,'Pattern recognition,vol 28,numver 3,pp 463–474,Elsevier,1995.

[8] Ben-Naoum Farah and Godin Christophe, "Algorithmic height compression of unordered trees,'Journal of theoretical biology,vol 389,pp 237–252,Elsevier,2016.