Comparing of SGML documents

MohammadTaghi Hajiaghayi
mhajiaghayi@uwaterloo.ca

Department of computer science
University of Waterloo

Abstract

Documents can be represented as structures with a hierarchial arrangement of text and non-text nodes, where nodes are labeled by category names such as *paragraph* and *section*. Representing documents this way is a natural consequence of using the Standard Generalized Markup Language(SGML) to encode text documents which has many applications in different areas.

There are many circumstances in which one structured document is a variant of another and we need to compare them to find the relationships (editing changes) between them. One simple way to do that is to model SGML documents by ordered labeled trees and then find a mapping between them. In this proposal we consider this problem deeply, and mention some solutions from the literature. Also, we introduce a research area which consists of a better approach for modeling the SGML documents and related problems. Finally, we mention some methods for attacking to these problems.

keywords: SGML, Comparing of documents, Ordered Labeled Trees, Tree-to-tree correction, Directed Labeled Graphs.

1 Motivation and Significance of the problem

Documents can be represented as structures with a hierarchial arrangement of text and non-text nodes, where nodes are labeled by category names such as *paragraph* and *section*. Representing documents this way is a natural consequence of using the Standard Generalized Markup Language(SGML) to encode text documents. SGML or other variants such as HTML are widely used and even documents that are not simple hierarchies can be represented using SGML. One simple way to model the documents represented in SGML or HTML are trees with labeled nodes where the left to right ordering of the offspring of a node is significant. We call this tree an *ordered labeled tree*. However, there are other sophisticated ways to model SGML documents which we consider later.

There are many circumstances in which one structured document is a variant of another. For example, there may be several manuscript or printed versions of an existing text; there may be several translations of a text; there may be several machine-readable forms of a text produced for different audiences(e.g. a text with minimal apparatus for students and a more extensive apparatus for researchers) or finally there may be interest in maintaining several versions of a document that is being produced in a cooperative manner by several authors. In all of these circumstances, we need to compare different variants of a document and find the relationships (editing changes) among them and in most of the cases these variants have a hypertext structure such as SGML. Finding the editing changes also can be used is other applications like:

Editing or Co-authoring: When an author presents a modified version of a document to an editor or co-author, the two versions could be compared to isolate only the changed components. A sophisticated display mechanism and a powerful editing tool must be capable of computing and highlighting the differences.

Storing: Documents are often published in several versions. A *document control system* might be modeled on a source code control system to provide help in managing versions. One of the characteristics of such a system should be that it minimize storage requirements by retaining only the computed differences between versions of a document without having to store more that one complete document. This is another natural application of comparing of document families.

Querying: If documents are stored in an archive using a structured representation, a query against the archive could also have a tree structure. The document(or document fragments) that satisfy the query would be those that most closely match the query tree, given an appropriate metric for distance. This can also be done by comparing of the query against the documents.

In addition, comparing of documents with tree structure can be used in other fields such as bioinformatics, e.g. comparing of two DNA's, etc.

2 Objectives

In this paper, we will study the problem of managing document families where documents are given and relationships or editing changes among them are to be derived (or computed).

The general approach to *edit distance* problems for string (or trees) in the literature have been to define a sequence of primitive operations that can be applied to one object to produce another, and to define the distance between two objects as a function computed on a sequence of such operations. In simple cases it can be sufficient to determine the length of the sequence . More realistically and more generally, each operation is assigned a *cost* that represents the difficulty of making the indicated change to the object. The cost could be thought of as the perceived unlikelihood of the change having arisen at random in whatever process produced the changed object. The problem is attacked by deriving an algorithm to search for a sequence of operations with minimal cost.

For string to string editing, the operations most frequently considered are: insert a character, delete a character, and change one character into another. But, for trees and other types of graphs there are other kinds of operations which we talk about them later.

3 Preliminaries

3.1 Edit Operations and Editing Distance

Let us consider three kinds of operations. Changing node n means changing the label on n. Deleting a node n means making the children of n become the children of the parent of n and then removing n. Inserting is the complement of delete. This means that inserting n as the child of n' will make n the parent of a consecutive subsequence of the current children of n'. Figure 1 illustrate these editing operations.

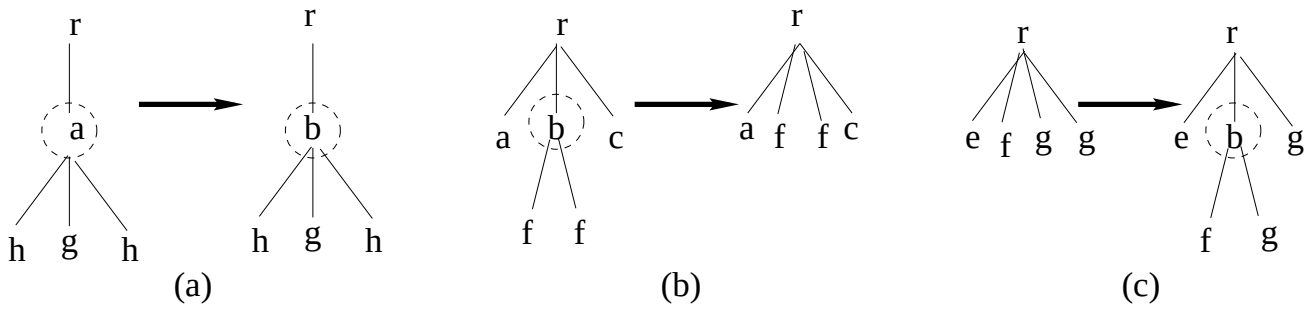

Figure 1: (a) Change (b) Delete (c) Insert operations.

We represent an edit operation as a pair $(a, b) \neq (\Lambda, \Lambda)$, sometimes written as $a \rightarrow b$, where a is either Λ or a label of a node in tree T_1 and b is either Λ or a label of a node in tree T_2. We call $a \rightarrow b$ a change operation if $a \neq \Lambda$ and $b \neq \Lambda$; a delete operation if $b = \Lambda$; and an insert operation if $a = \Lambda$. Since many nodes may have the same label, this notation is potentially ambiguous. It could be made precise by identifying the nodes as well as their labels. However, in this paper, which node is meant will always be clear from the context.

Let S be a sequence $s_1, \cdots, s_k$ of edit operations. An S-derivation from A to B is a sequence of trees $A_0, \cdots, A_k$ such that $A = A_0$, $B = A_k$, and $A_{i-1} \rightarrow A_i$ via s_i for $1 \leq i \leq k$. let γ be a cost function that assigns to each edit operation $a \rightarrow b$ a nonnegative real number $\gamma(a \rightarrow b)$. This cost can be different for different nodes, so it can be used to give greater weights to, for example, the higher nodes in a tree than to lower nodes. We constrain γ be a distance metric. That is,

1. $\gamma(a \rightarrow b) \geq 0$; $\gamma(a \rightarrow a) = 0$;
2. $\gamma(a \rightarrow b) = \gamma(b \rightarrow a)$; and
3. $\gamma(a \rightarrow c) = \gamma(a \rightarrow b) + \gamma(b \rightarrow c)$.

We extend γ to the sequence S by letting $\gamma(S) = \sum_{i=1}^{i=|S|} \gamma(s_i)$. Formally the distance between T_1 and T_2 is defined as follows:

$\delta(T_1, T_2) = min\{\gamma(S) | S$ is an edit operation sequence taking T_1 to $T_2\}$.

The definition of γ makes δ a distance metric also.

3.2 Mapping

The edit operation give rise to a mapping which is a graphical specification of what edit operation apply to each node in the two ordered labeled trees. Suppose that we have a numbering for each tree. Let $t[i]$ be the ith node of tree T and $T[i]$ be the subtree rooted at $t[i]$ in the given numbering. Formally we define a triple (M, T_1, T_2) to be a mapping from T_1 to T_2 where M is any set of pairs of integers (i, j) satisfying:

1. $1 \leq i \leq |T_1|$, $1 \leq j \leq |T_2|$;
2. For any pair of (i_1, j_1) and (i_2, j_2) in M,
   (a) $i_1 = i_2$ if and only if $j_1 = j_2$(one-to-one),

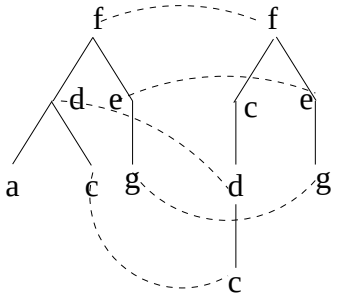

Figure 2: Mapping

(b) $t_1[i_1]$ is to the left of $t_1[i_2]$ if and only if $t_2[j_1]$ is to the left of $t_2[j_2]$(sibling order preserved),

(c) $t_1[i_1]$ is an ancestor of $t_1[i_2]$ if and only if $t_2[j_1]$ is an ancestor of $t_2[j_2]$(ancestor order preserved),

You can see one example of mapping in figure 2. The cost of M is the cost of deleting nodes of T_1 not touched by a mapping lines plus the cost of inserting nodes of T_2 not touched by a mapping line plus the cost of relabeling nodes in those pairs related by mapping lines with different labels. In [10], it is proved that $\delta(T_1, T_2)$ equals the cost of a minimum cost mapping from tree T_1 to tree T_2.

It is worthy to mention that above consitions are only one kind of the definition of the mapping and we can obtain other kinds by slightly changing some of them.

4 History of tree-to-tree correction

There are many approaches for solving the problem of tree-to-tree correction which consist of comparing of two ordered labeled trees and finding the minimum cost of primitive operations to reduce one tree to another. Most of these approaches use dynamic programming which is, in fact, the generalized approach for solving the string-to-sting correction (or sequence editing) problem introduced in [11], however, there are other methods; for example, in [14] this problem is considered as an optimization problem and a simple recursive (back-tracking) algorithm is suggested for this problem, but the running time of the algorithm is exponential and so it is not a fast algorithm comparing to dynamic approaches. In next subsections we discuss more about the dynamic solutions for this problem.

4.1 Tai's classical

Tai gave the definition of the edit distance between ordered labeled trees and the first non-exponential algorithm to compute it [10]. Tai used the preorder number to number the trees. The convenient aspect of this notation is that for any i, $1 \leq i \leq |T|$, nodes from $t[1]$ to $t[i]$ is a tree rooted at $t[1]$. Given two trees T_1 and T_2, let $D_t(T_1[1..i], T_2[1..j])$ be the edit distance between $t_1[1]$ to $t_1[i]$ and $t_2[1]$ to $t_2[j]$.

We can use the same approach as in sequence editing dynamic algorithm[11] and try to construct D_t of pair (i, j) from pairs $(i, j-1)$, $(i-1, j)$ and $(i-1, j-1)$. If either $t_1[i]$ or $t_2[j]$ is not involved in the mapping, then it is exactly the same as in sequence editing, otherwise we have some difficulties. In order to deal with this difficulty, Tai introduces another two measures between trees and the resulting algorithm is quite complex with a time and space complexity of $O(|T_1| \times |T_2| \times depth(T_1)^2 \times depth(T_2)^2)$.

4.2 Lu's algorithm

Another edit distance algorithm between ordered trees is reported by Lu[9]. Lu's definition of the edit operations are the same as Tai's. However the algorithm given by Lu does not compute the edit distance as it defined. Nevertheless, it does provide another edit based distance.

Let $t_1[i_1], \cdots, t_1[i_{k_1}]$ be the children of $t_1[i]$ and $t_2[j_1], \cdots, t_2[j_{k_2}]$ be the children of $t_2[j]$. Lu's algorithm considers the following three case:

1. $t_1[i]$ **is deleted:** In this case the distance would be to match $T_2[j]$ to one of the subtrees rooted at children of $t_1[i]$ and then to delete all the rest of the subtrees.
2. $t_2[j]$ **is inserted:** In this case the distance would be to match $T_1[i]$ to one of the subtrees rooted at children of $t_2[j]$ and then to delete all the rest of the subtrees.
3. $t_1[i]$ **matches** $t_2[j]$**:** In this case, consider $t_1[i_1], \cdots, t_1[i_{k_1}]$ and $t_2[j_1], \cdots, t_2[j_{k_2}]$, as two sequences and each individual subtree as a whole entity. Use the sequence edit distance to determine the distance between $t_1[i_1], \cdots, t_1[i_{k_1}]$ and $t_2[j_1], \cdots, t_2[j_{k_2}]$.

From the above description it is easy to see the difference between this distance and the edit distance. This algorithm considers each subtree as a whole entity. It does not allow one subtree of T_1 to map to more than one subtrees of T_2. But, using the definition of edit distance, we can delete the root of one subtree and then map the remaining subtrees of this subtree to more than one subtrees.

4.3 Selkow Distance

Selkow gave another tree edit algorithm in which the insertion and deletions are restricted to the leaves of trees. Only leaves may be deleted and a node may be inserted only as a leaf.

In this case, it is easy to see that if $t_1[i]$ maps to $t_2[j]$ then the parent of $t_1[i]$ must map to the parent of $t_2[j]$. The reason is that if $t_1[i]$ is not deleted (or inserted) , its parent can not be deleted (or inserted). Using this idea, it is easy to design an algorithm very similar to sequence editing dynamic algorithm with time complexity of $O(|T_1| \times |T_2|)$.

4.4 Zhang and Shasha algorithm

Let $t[i]$ be the ith node in the tree according to the left-to-right postorder numbering and $l(i)$ be the number of the leftmost leaf descendant of the subtree rooted at $t[i]$. So, when $t[i]$ is a leaf, $l(i) = i$. $T[i..j]$ is the ordered subforest of T induced by the nodes numbered i to j inclusive. $T[1..i]$ will be referred to as $forest(i)$, when the tree T referred to is clear. $T[l(i)..i]$ will be referred to as $tree(i)$ or simply $T[i]$.

The distance between $T_1[i'..i]$ and $T_2[j'..j]$ is denoted by $forestdist(T_1[i'..i], T_2[j'..j])$ or simply $forestdist(i'..i, j'..j)$ if the context is clear. We use more abbreviated notation for certain special cases. The distance between $T_1[i]$ and $T_2[j]$ is sometimes denoted by $treedist(i, j)$. We first present three lemmas and then give the idea of the algorithm.

Lemma 1 *We have*

- $forestdist(\emptyset, \emptyset) = 0$
- $forestdist(l(i_1)..i), \emptyset) = forestdist(l(i_1)..i-1, \emptyset) + \gamma(t_1[i] \rightarrow \Lambda)$
- $forestdist(\emptyset, l(j_1)..j) = forestdist(\emptyset, l(j_1)..j-1) + \gamma(\lambda \rightarrow t_2[j])$

where $i \in desc(i_1)$ *and* $j \in desc(j_1)$.

Proof: Trivial. □

Lemma 2 *Let* $i \in desc(i_1)$ *and* $j \in desc(j_1)$. *Then*

$$forestdist(l(i_1)..i, l(j_1)..j) = min \begin{cases} forestdist(l(i_1)..i-1, l(j_1)..j) + \gamma(t_1[i] \rightarrow \Lambda), \\ forestdist(l(i_1)..i, l(j_1)..j-1) + \gamma(\Lambda \rightarrow t_2[j]), \\ forestdist(l(i_1)..l(i)-1, l(j_1)..l(j)-1) \\ +forestdist(l(i)..i-1, l(j)..j-1) + \gamma(t_1[i] \rightarrow t_2[j]) \end{cases}$$

Proof: It is only suffices to consider these four cases arise when $t_1[i]$ is (not) touched by a line in M and $t_2[j]$ is (not) touched by a line in M. □

Lemma 3 *Let* $i \in desc(i_1)$ *and* $j \in desc(j_1)$. *Then*

1. *if* $l(i) = l(i_1)$ *and* $l(j) = l(j_1)$

$$forestdist(l(i_1)..i, l(j_1)..j) = min \begin{cases} forestdist(l(i_1)..i-1, l(j_1)..j) + \gamma(t_1[i] \rightarrow \Lambda), \\ forestdist(l(i_1)..i, l(j_1)..j-1) + \gamma(\Lambda \rightarrow t_2[j]), \\ forestdist(l(i_1)..i-1, l(j_1)..j-1) + \gamma(t_1[i] \rightarrow t_2[j]) \end{cases}$$

2. *if* $l(i) \neq l(i_1)$ *or* $l(j) \neq l(j_1)$ *(i.e. otherwise)*

$$forestdist(l(i_1)..i, l(j_1)..j) = min \begin{cases} forestdist(l(i_1)..i-1, l(j_1)..j) + \gamma(t_1[i] \rightarrow \Lambda), \\ forestdist(l(i_1)..i, l(j_1)..j-1) + \gamma(\Lambda \rightarrow t_2[j]), \\ forestdist(l(i_1)..i-1, l(j_1)..j-1) + treedist(i, j) \end{cases}$$

Proof: Immediately from lemma 2. □

Lemma 3, has three important implications: First the formulas it yields suggest that we can use a dynamic programming to solve the tree distance problem. Second, from part 2 of this lemma, we observe that to compute $treedist(i_1, j_1)$ we need have in advance almost all values of $treedist(i, j)$ where i_1 is the root of a subtree containing i and j_1 is the root of a subtree containing j. This suggests a bottom-up procedure computing all subtree pairs. Third, from part 1 of this lemma, we can observe that when i is in the path from $l(i_1)$ to i_1 and j is in the path from $l(j_1)$ to j_1, we do not need to compute $treedist(i, j)$ separately. This subtree distances can be obtained as a byproduct of computing $treedist(i_1, j_1)$.

Using the above ideas we can design a dynamic program that $forestdist$ and $treedist$ mutually call each other, and finally compute $treedist(|T_1|, |T_2|)$. You may see the whole program in [18]. An improved version

of this algorithm for the case that each of edit operation has unit cost is presented in [19]. Also, the problem of finding a parallel algorithm for this problem is considered in [17]. The idea of this algorithm again is lemmas 3. Author in this paper has shown that this problem has a polylogrithmic time algorithm and so can be parallelized efficiently.

5 Other Related Problems

In this section we introduce some other related problems which can be solved with the idea of Zhang and Shasha algorithm. These problems, also, can be used for comparing and querying the documents like SGML which has a tree structure.

We define a substructure U of tree T to be a connected subgraph of T. That is, U is rooted at a node n in T and is generated by cutting off some subtrees in the subtree rooted at n. Formally, let $T[i]$ represent the subtree rooted at $t[i]$. The operation of cutting a node $t[i]$ means removing $T[i]$. A set S of nodes of $T[k]$ is said to be a set of consistent subtree cuts in $T[k]$ if

1. $t[i] \in S$ implies that $t[i]$ is a node in $T[k]$, and
2. $t[i], t[j] \in S$ implies that neither is an ancestor of the other in $T[k]$.

Intuitively, S is the set of all roots of the removed subtrees in $T[k]$.

We use *Cut(T, S)* to represent the tree T with subtree removals at all nodes in S. Let $|Cut(T, S)|$ be the number of the nodes of this tree. Also, let *Subtree(T)* be the set of all possible sets of consistent subtree cuts in T. Given two trees T_1 and T_2 and an integer d, the size of the largest approximately common root containing substructures within distance k, $0 \leq k \leq d$, of $T_1[i]$ and $T_2[j]$, denoted $\beta(T_1[i], T_2[j], k)$(or simply $\beta(i, j, k)$ when the context is clear), is max $\{|Cut(T_1[i], S_1)| + |Cut(T_2[j], S_2)|\}$ subject to $\delta(Cut(T_1[i], S_1), Cut(T_2[j], S_2)) \leq k$, $S_1 \in Subtree(T_1[i])$, $S_2 \in Subtree(T_2[j])$. Finding the largest approximately common substructure(LACS), within distance d, of $T_1[i]$ and $T_2[j]$ amounts to calculate $max_{1 \leq u \leq i, 1 \leq v \leq j}\{\beta(T_1[u], T_2[v], d)\}$ and locating the $Cut(T_1[u], S_u)$ and $Cut(T_2[v], S_v)$, $S_u \in Subtrees(T_1[u])$, $S_v \in Subtrees(T_2[v])$ that achieve the maximum size. So, the size of LACS, within distance d, of T_1 and T_2 is the $max_{1 \leq i \leq |T_1|, 1 \leq j \leq |T_2|}\{\beta(T_1[i], T_2[j], d)\}$. Intuitively, LACS of two trees is very similar to largest common subsequence of two strings and using similar dynamic programming which we had in Zhang and Shasha algorithm, we can compute LACS of two trees. You can find the details in [13].

Pattern matching is another related problem to tree-to-tree correction problem and also can be solved with the same dynamic approach. Preliminary version of this pattern matching for trees is introduced in [12]. *Approximate tree matching* is defined as computing

$tree - cut(P, T) = min_{C \in Subtree(T)}\{\delta(P, cut(T, C))\}$.

Where both P and T are trees. Intuitively, this is the distance between the pattern tree and the cut data tree, where the cut yields the smallest possible distance.

The more general concept which has many applications in querying the SGML documents is introduced in [20]. Suppose some of nodes have one of the two labels: | and $\wedge$. A node with label | in the pattern tree can substitute part of a path from the root to a leaf of the data tree. A node with $\wedge$ in the pattern tree can substitute part of such path and all the subtrees emanating from the nodes of that path, except possibly at the lowest node of that path. At the lowest node, $\wedge$ symbol can substitute for a set of leftmost subtrees and

a set of rightmost subtrees. We call a | a *path* and $\wedge$ an *umbrella* labels, because of the shape they impose on the tree.

Let P be a pattern tree that contains both umbrella and path labels and let T be a data tree. A substitution s on P replaces each path in P by a path of nodes in T and each umbrella in P by an umbrella pattern of nodes in T. After this substitution we get another tree $\overline{P}$ in T, however still $\overline{P}$ is different from T. The approximate matching between P and T w.r.t. s, is defined as $tree - vldc(P, T, s) = tree - cut(\overline{P}, T)$. Then,

$tree - vldc(P, T) = min_{s \in S}\{tree - vldc(P, T, s)\}$

Where S is the set of all possible substitutions. In [20] again by deriving the similar formulae to lemmas 1-3, the author could design a dynamic algorithm. Also, in this paper, you can find other generalization of this problem.

6 Generalization

In this section, we discuss comparing of other structures than *ordered labeled tree* introduced in Introduction.

In the first step, we may consider layout [1] of *ordered labeled tree* as general unlabeled graphs. [15] has defined one kind of distances between genreal undirected graphs as follows: Let G be a finite undirected graph without loops and multiple edges. Let u, v, and w be three pairwise distinct vertices of G such that u is adjacent to v and is not adjacent to w. To perform the *rotation* of the edge uv into the position uw means to delete the edge uv from G and to add the edge uw to G. A special kind of the edge rotation is the edge *shift*. The shift of the edge uv into the position uw is the rotation of uv into the position of uw in the case when the vertices v, w are adjacent in G. In [8], it was proved that each two graphs from the class of all connected undirected graphs with n vertices and m edges can be transformed into each other by a finite number of edge shifts. [15] considers the problem specially when the two graphs are trees. There are so many other distance measures between graphs in the literature, however most of them are not so appropriate for comparing the graphs of SGML documents, e.g. there is not a good corresponding operations like edge shift in the SGML documents. In the rest of this section we consider other generalization which are more related to comparing SGML documents.

Unordered rooted labeled trees are trees whose nodes are labeled and in which only ancestor relationships are significant (the left-to-right order among sibling is not significant). This structure is so appropriate for the case that the order of children is not important in the hierarchial structure, e.g. in documents that each node is an item and has only unordered sub-items.

The edit distance for these graphs is defined similarly to our definition for *ordered labeled trees*. In [21] has been shown that computing the edit distance for unordered labeled trees in NP-complete, but still we would like to find the distances that can be efficiently computed. [16] considers a *constrained edit distance* metric between unordered labeled trees. The constraint is a natural one, namely, disjoint subtrees must be mapped to disjoint subtrees. Again using our general idea of dynamic programming this problem can be solved in $O(N_1 N_2 (d_1 + d_2)(\log_2 d_1 + \log_2 d_2)$ where $N_1(N_2)$ is the number of vertices in $T_1(T_2)$ and $d_1(d_2)$ is the maximum degree in $T_1(T_2)$.

Also, it is possible that we only want to compare the structures of the layouts of two graphs with the edit operations defined before and find out how much this two structures are similar. The problem of comparing CUAL graphs (Connected, Undirected , Acyclic graphs with nodes being Labeled) considered in [22]. For

[1] The *layout* G' of directed graph G can be obtained from G by removing the directions of its edges.

these graphs, we define the *distance* to be the weighted number (the user chooses the weighting) of edit operations (insert node, delete node and relabel node) to transform one graph to the other (Similar to definition of the original problem). By reduction from exact cover by 3-sets (see [6] for exact definition of this problem), one can show that finding the distance between two graphs is NP-hard. In view of the hardness of the problem, the authors propose a constrained distance metric, called the *degree-2 distance*, for graphs by requiring that any node to be inserted(deleted) have no more than 2 neighbors. In [22], it has been shown that this constraint is sensible in defining the edit operations on graphs and further, the measure is a natural extension of the edit distance for strings and Selkow's distance for trees. In this paper, again using the same ideas of dynamic programming with slight differences in the formulae, authors could compare the two graphs G_1 and G_2 in time $O(N_1N_2(min\{d_1, d_2\})^2)$ where $N_1(N_2)$ is the number of vertices in $G_1(G_2)$ and $d_1(d_2)$ is the maximum degree in $G_1(G_2)$.

Other generalization of this problem which has been introduced in [1] considers adding of other primitive operations. When editing documents most people perform operations that are more complex than our operations defined in section §3. Examples of operations which are commonly performed when editing documents are:

- merge two sections,
- split one section into two,
- permute sections of text under an unchanging encompassing structure, and
- move one section of text to another position.

However, many of above operations are, in fact, combinations of other primitive operations. [1] has claimed that by adding the below operations to our old primitive set of operations we have a better primitive set of operations:

- *deleteTree* which deletes a whole subtree.
- *insertTree* which inserts a whole subtree.
- *swap* which allows us to swap the adjacent sibling of a node and after that finds the edit distance.

Again the algorithm presented in this paper for comparing of two documents using these generalized set of primitive operations can be easily obtained by slight changes of the general Zhang and Shasha dynamic algorithm.

7 Proposed Research Program

In this section we propose our new research problems. As we have mentioned before, the current model for SGML documents is ordered labeled tree struture or other its generalization which most of all have tree structures. But, this model is not enough for representing and comparing of SGML documents, because most of the SGML documents like HTML documents have the reference link, and so their structures is not necessarily a tree. The best structure for modeling of these documents are Directed Labeled Graphs(DLG). We can also generalize this model futher in which some of its vertices have an ordering among outgoing edges. We call this generalized model an Approximately Ordered Direceted Labeled Graphs(AODLG). This

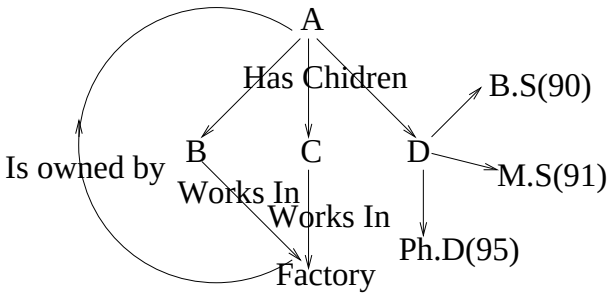

Figure 3: An example of AOLDG model for an SGML document

general concept can model every SGML documents. You can see one simple example in figure 3. This directed graph is an model of a SGML document. It's worthy to mention that, in fact, this graph can be considered as AODLG, because the order of outgoing edges for vertex A is not important, but the order of outgoing edges for vertex D is important.

Now our goal is to compare two AODLG, G_1 and G_2 and find a mapping M between them. Our primitive operations again are change, delete and insert operations. Also, we assume that each operation has a cost. The cost of M is the cost of deleting nodes of G_1 not touched by a mapping lines plus the cost of inserting nodes of G_2 not touched by a mapping line plus the cost of relabeling nodes in those pairs related by mapping lines with different labels.

There are several reasons for investigating a body of DLG comparing problems. First, a DLG is a natural choice for expressing SGML documents and so many sophisticated data in different applications can be compared using them. Second, DLG-to-DLG correction problems are natural extensions of string-to-string and tree-to-tree correction problems and are interesting to investigate their own right. Third, DLG-to-DLG correction probelm has not received much deserved attention mainely because of the high time complexity of performing editing operation on digraphs. However, in recent years, more and more researchers have adopted digraphs to implemetnt their data models. A thorogh investigation of this problem will provide efficient algorithms for some of the problems which are valuable for both current and potential appllications.

Another important thing is that DLG's are the generaliziation of Directed Unlabeled Graphs(DUG) because an DUG can be viewed as a DLG in which all nodes have the same label. Determining whether two DUG are isomorphic to each other, the simplest among all mapping DLG problems, is NP-complete[6]. So it emphasises that this problem is a hard problem. However, we still would like to find efficient solutions to these problems. Here we discuss about some suggestions which help us to attack these problems.

7.1 Consider the Approximately Ordered Labeled Trees(AOLT)

In section before, we have mentioned that comparing of labeled trees without ordering which is a special case of AOLT, is NP-complete. However, ALOT structures are often used in modeling of SGML documents in which some of the vertices have ordering, and some doesn't have. *Constrained edit distance* was restriction that enabled the authors in [21] to solve this problem. Another restriction which seems logical is that the maximum degree in SGML documents are not so large (in must of the case we can assume that it is less than 7). It seems that if we consider the case in which degree of each vertex is at most a constant C, then we would be able to solve this problem using the dynamic approach similar to Zhang and Shasha algorithm. I expect the research in this subject takes time about six months.

7.2 Consider the Directed Acyclic Graphs

In our generalized model of SGML documents, we consider the directed graphs without any constraints. However, in most of the cases the model graph of SGML documents (not HTML documents) does not have cycles. This is very important thing that we can use it to solve the generalized problem without forcing so much restrictions. However, still, the problems for AODLG seems to be NP-complete, because trees without ordering of children are special cases of this graphs and as we have mentioned before this problem is NP-complete. However, for the case that each vertex has an ordering on the outgoing edges, seems to be solvable using a dynamic approach. Another witness for possibility of finding an algorithm for this problem is that the related problem of pattern matching for Directed Acyclic Graphs considered by Fu in his Doctoral thesis[5] and he could solve this problem in polynomial time. It seems that his idea also can be used for our case. I expect the research in this subject takes time about one year.

7.3 Using Tree Decompositions

Let G be an undirected graph. G is called a *k-tree* if either G is a k-clique [2] or G has a vertex u of degree k such that u is adjacent to a k-clique, and the graph obtained by deleting u and all its incident edges is again a k-tree. A *partial k-tree* is subgraph of a k-tree.

Treewidth of a graph G is a minimum k such that graph G is a partial k-tree. In fact, treewidth is a measure that shows how much a graph is similar to a tree. The treewidth of a tree is one, because each tree can be obtained by iteration of adding one vertices to only one adjacent vertex(a 1-clique). Each graph G with n vertices has treewidth of at most n, since it is a subgraph of a n-clique.

A *tree decomposition* of a graph G is a pair $T(G) = (T, \chi)$ where T is a tree and χ is a mapping from $V(T)$ into the set of subsets of $V(G)$ satisfying:

1. for every edge $(u, v) \in E(G)$ there is a $b \in V(T)$ such that $\{u, v\} \subset b$.
2. for all $v \in V(G)$, the set of nodes $\{b \in V(T) | v \in b\}$ forms a connected part (i.e., a subtree) of T.

For $b \in V(T)$, the subset $\chi(b)$ is often identified with b. Then $V(b) = \chi(b)$ is the set of vertices of b. To distinguish between the graph G and a tree decomposition $T(G)$ the elements $v \in V(G)$ are called vertices and the elements $b \in V(T(G))$ are called *nodes*. Accordingly we distinguish between *edges* of G and *arcs* of T(G). An arc of $T(G)$ are directed from a parent to a child.

The *width* of a tree-decomposition $T(G)$ is max $\{|\chi(b)| - 1 | b \in V(T)\}$. It can be proved (see [4]) that the *treewidth* of a graph G is the minimum value k such that G has a tree decomposition of width k. A tree decomposition $T(G)$ is minimal, if its width is equal to $treewidth(G)$ and there are no(unnecessary) repetitions of nodes b and b' with $\chi(b) = \chi(b')$.

Now let G be a directed graph corresponds to a SGML document. The *layout* G' of graph G can be obtained from G by removing the directions of its edges. In fact, most of the layouts of SGML model graphs, are very much similar to trees, if they are not exactly tree (As we have mentioned before most of authors assume that it is a tree). This similarity causes that this graph has a small treewidth(for example less than 4 or 5). The important matter is that if a graph has small treewidth then we can efficiently find its tree decomposition [4] and using its tree decomposition we can solve most of the NP-complete problems very efficiently(even in

[2] A k-clique is a graph with k vertices such that all vertices are connected to each other

linear time)[3]. The problem of testing isomorphism has been solved using this ideas in [7]. However, in our case we want to compare two graphs and this is equivalent to find the largest common subgraph of these two graphs, because after that we can find out what vertices must be deleted, what vertices must be inserted and what vertices must be relabeled. We have studied this problem and we think that using the similar method used in [7], we can solve this problem and so we will have an efficient algorithm for general case of directed graphs which are corresponded to SGML documents. I expect the research in this subject takes time about one year.

7.4 Approximate and Randomized algorithm

This approach is a general approach for attacking most of the NP-complete problems. As we have mentioned in section 3, before finding the dynamic solution for tree-to-tree correction problem, someone considered this problem as an optimization problem and tried to solve it by finding approximate solutions for optimization problem. We can also use this approach for comparing of two DLG's or AODLG's problem. Also in most of the cases knowing a good approximation for the cost of a mapping is enough for us, and we need not necessarily the exact solution. The only important matter is that, our algorithm must be fast, because most of documents are too large and also we need to compare too many documents in a short period of time. Randomized algorithm is also another method to attack this problem because, in most of the cases randomized algorithm is fast and finds an approximately good solutions. I expect the research in this subject takes time about six months.

8 Scope of effort

I expect the research described in this proposal to be addressed over a three to four year period and one doctoral students works for this project during this period. Also only using tree-decomposition for solving this problem itself could be a project of a Master Student. It is so probable that I work on this project with my supervisor Professor Nishimura for my Masters Degree.

9 Conclusion

In this proposal we introduced the problem of comparing two SGML documents which has many applications in different areas. First, we have modeled SGML documents by ordered labeled trees, and review the literature about this problem, then we talked about other related problems and the generalization of these problems. Finally we proposed a new research area for better comparing of two SGML documents which can be a doctoral thesis and in some abbreviation form be a Master thesis.

Acknowledgments. I thank my supervisor, professor Nishimura for her thoughtful comments.

References

[1] D. Barnard, G. Clarke, N. Duncan, "Tree-to-tree correction for document trees", *Technical Report 95-375, Queen's University*, January 1995.

[2] D. T. Barnard, G. M. Logan, "Complementary approaches to representing differences between structured documents", *Computing and Information Sciences, Queen's University, Ontario, Canaca.*

[3] H. Bodlaender, "Dynamic programming on graphs with bounded tree-width", *In Proc. 15th Colloquium on Automata, Languages and Programming, LNCS 317, Tampere(1988)*, 105-118.

[4] H. Bodlaender, "A linear-time algorithm for finding tree-decompositions of small treewidth", *SIAM J. Comput., 25(6), 1996*, 1305-1317.

[5] J. J. Fu, "Pattern mathcing in directed graphs", *Doctoral Thesis, Dept. of Computer Science, University of Waterloo*, 1996.

[6] M. R. Garey, D. S. Johnson, *Computers and Intractability, A guide to the theory of NP-completeness*, W.H. Freeman and Company, 1979.

[7] A. Gupta, N. Nishimura, "Sequence and parallel algorithms for embedding problems on classes of partial k-trees", *Proc. SWAT'94, LNCS 824, 1994*, 172-182.

[8] M. Johnson, "An ordering of some metrics defined on the space of graphs", *Czech Math. J. 37, 1987*, 75-85.

[9] S. Y. Lu, "A tree-to-tree distance and its application to cluster analysis", *IEEE Trans. PAMI 1, 1979*, 219-224.

[10] K. C. Tai, "The tree-to-tree correction problem", *J. ACM 26(3), 1979*, 422-433.

[11] R. A. Wagner, M. J.Fischer, "The string-to-string correction problem", *Journal of ACM 21(1), 1974*, 168-173.

[12] J. Wang, K. Zhang, K. Jeong, D. Shasha, "A System for Approximate Tree Matching", *Proc. of the TKDE, IEEE Computer Society Press, 6(4), 1994*, 559-571.

[13] J. T. Wang, B. A. Shapiro, D. Shasha, K. Zhang, and K. M. Currey, "An algorithm for finding approximately common substructures of two trees.", *IEEE Trans. PAMI 20(8), 1998*, 889-895.

[14] R. Wilhelm, "A modified tree-to-tree correction problem ", *Information processing letters 12(2), 1981*, 127-132.

[15] B. Zelinka, "Edge shift distance between trees", *Archivum Mathematicum 28(1992)*, 5-9.

[16] K. Zhang, "A constrained edit distance between unordered labeled trees", *Algorithmica 15, 1996*, 205-222.

[17] K. Zhang, "Efficient parallel algorithms for tree editing problems", *Department of Computer Science, University of western ontario, Canada* .

[18] K. Zhang, D. Shasha, "Simple fast algorithm for the editing distance between trees and related problems", *SIAM J. Comput. 18(6), 1989*, 1245-1262.

[19] K. Zhang, D. Shasha, "Fast algorithms for the unit cost editing distance between trees", *Journal of algorithms 11, 1990*, 581-621.

[20] K. Zhang, D. Shasha, J. T. Wang, "Approximate tree matching in the presence of variable length don't cares", *Journal of algorithms 16, 1994*, 33-66.

[21] K. Zhang, R. Statman, D. Shasha, "On the editing distance between unordered labeled trees", *Information Processing letters 42, 1992*, 133-139.

[22] K. Zhang, J. T. Wang, D. Shasha, "On the editing distance between undirected acyclic graphs and related problems", *December 28, 1994.*