An Expressive Model for Comparing Tree-Structured Data*

Sudarshan S. Chawathe Hector Garcia-Molina

Computer Science Department, Stanford University, Stanford, California 94305

{chaw,hector}@cs.stanford.edu

1 Introduction and Overview

We study the problem of comparing tree-structured data. This problem has applications in several diverse fields, including molecular biology, natural language processing, image and pattern matching, program analysis, document management, and change management in databases [SZ90, NBR88, WZJS94, WCM+94, Yan91, WSC+97, C3]. Our interest in this problem stems from implementing a subscription service that detects database changes by periodically querying an external database and checking the results for new changes [CGL+97]. For example, consider a Web site listing local entertainment events [EG]. Our prototype allows a user to subscribe to certain changes e.g., the cancellation or rescheduling of an event. The prototype periodically queries the event listings, and uses a tree-differencing algorithm to detect changes in the results.

In this work we focus on comparing *rooted, unordered, labeled trees*, such as those depicted in Figure 1. Tree nodes are represented by circles; each node has a label, indicated next to it, and an identifier, indicated inside the circle. In our Web example, the label of node 3 may represent a section heading, and its child nodes the paragraphs in the section. Note that we use node identifiers for notational convenience only; we do not assume that these identifiers are object-identifiers or keys that can be used to match nodes in one tree with those in the other. (However, in cases where such object identifiers or keys exist, we can take advantage of them.)

Figure 1(a) also illustrates how changes to a tree T_1 can be represented by a *linear edit script* $\mathcal{E}_1 = (cpy(5,2),\ cpy(4,6),\ mov(4,5))$. The script is a sequence of *edit operations* that transforms T_1 into T_1'. For example, the first operation in our script, $cpy(5,2)$, makes a copy of the T_1 node 5, and places it under node 2. The new node has a new identifier, in this case 6. The edit operations commonly used in the literature are node *insertion*, node *deletion*, and node *relabeling*. Here we extend this set to include more expressive operations, such as subtree copy and subtree move, that capture changes more succinctly. For example, when comparing structured documents, saying that a paragraph was moved is more helpful than saying that the sentences in the paragraph were deleted and then inserted somewhere else.

Our goal is to find a compact representation of the changes between two trees. If we use linear edit scripts, our goal is to find a "minimum-cost" script that transforms the first tree into one that is isomorphic to the second. We assign costs to operations and look for a minimum-cost script to ensure that the script does not do more work than needed. Unfortunately, this traditional method of describing changes using linear edit scripts has several problems when used with subtree operations such as moves and copies.

The first problem is that it is difficult to understand an edit operation on its own, because its effect depends on the operations preceding it in the edit script. For example, consider the following edit script applied to T_1 in Figure 1(a): $\mathcal{E}_2 = (mov(4,5), cpy(5,2))$. If we focus on the copy operation, we would intuitively expect it to produce a copy of node 5, with node 2 as the parent. However, because of the preceding move, the copy actually produces a copy of both nodes 4 and 5. In fact, $\mathcal{E}_2$ has the same effect as the script $\mathcal{E}_1$ discussed earlier. This equivalence is not clear from the edit scripts themselves; we need to actually apply the edit scripts to discover it.

Another problem with the linear edit script model is that it may result in very unintuitive edit scripts. For example, consider structured documents, and suppose the cost of a copy operation is 5 units, while the

*Extended abstract. Contact information: phone: 650-723-6805; fax: 650-725-2588. This work was supported by by the Air Force Rome Laboratories under DARPA contract F30602-95-C-0119.

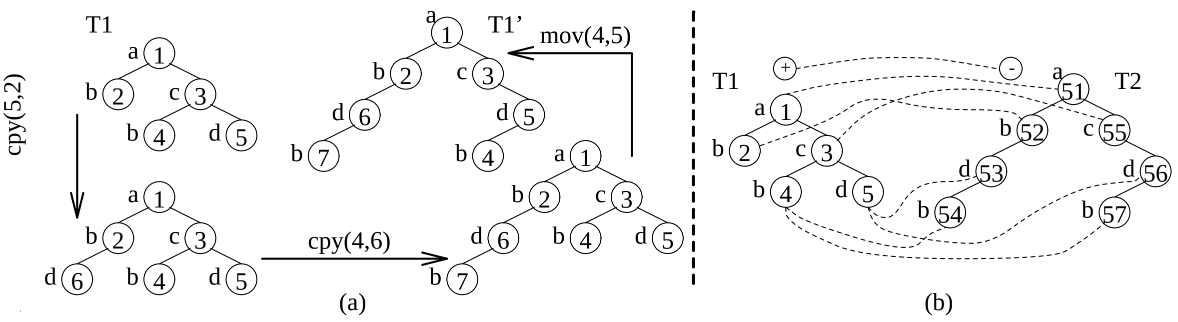

Figure 1: Applying a linear edit script.

cost of a move operation is 1 unit. Consider now a script that moves the subtree rooted at a node n_1 to below another node n_2, copies the subtree rooted at n_2 (thus also making a copy of the n_1 subtree), and then moves both the original and the copy of the subtree rooted at n_1 to other locations. We observe that the sole purpose of the initial move operation is to get a "free" copy of the subtree at n_1, thus reducing the overall cost of the edit script. However, this "trick" is not very intuitive in the application context: If n_1 and n_2 represent paragraphs, the above script says that paragraph n_1 is temporarily moved under n_2, not because n_1 is at all related to n_2, but simply to make it cheaper to make the copies we eventually need of n_1 that will go elsewhere.

In this paper we present a novel method to represent changes and to compare trees that avoids these and other problems. The intuitive idea is to apply edit operations "in parallel" as opposed to in sequence. That is, we apply a *set* of edit operations, called a *transformation*, to a tree by first disassembling the given tree into "*chunks*," then operating on each chunk independently, and finally reassembling the resulting chunks to get the final tree. (In Section 2 we describe our model in detail.) Our model is free from the the unintuitive artifacts resulting from the interdependencies between edit operations in the linear edit script model. Even more importantly, searching for a minimum-cost parallel transformation is simpler that searching for a minimum-cost edit script (when moves and copies are allowed). This simplicity is because the essential information in a transformation, including its cost, can be compactly represented in a *signature*. Thus, we can search for a minimum-cost signature and then map it back to the corresponding transformation. In this paper we show how signatures are constructed, and how they map to transformations. The mapping between signatures and transformation is independent of the cost model used, making our methods for detecting changes useful in diverse application domains.

The idea of working with signatures is widely used in the literature of differencing algorithms, in various forms (such as "traces" or matchings) [WF74, Mye86, ZS89, Yan91, CGM97]. However, the introduction of move and copy operations makes it hard to recover a script from a signature, and this makes it difficult to detect changes using signatures. To illustrate some of these difficulties, Figure 1(b) shows the "traditional" signature of the edit script in Figure 1(a). The trees T_1 and T_2 are represent the initial and final trees (respectively) from Figure 1(a). However, note that node identifiers in T_2 are different from those in T_1' (and T_1) because we do not know yet how T_2 was obtained from T_1. Intuitively, the signature is a relation (dashed lines) that maps each node in T_2 to the node or nodes in T_1 from which it is "derived." For example, if nodes $n_1, n_2 \in T_2$ are copies of a common node $m \in T_1$, m the signature maps m to both n_1 and n_2. Deleted T_1 nodes are mapped to a special node $\ominus$, while inserted T_2 nodes are mapped to a special node $\oplus$. In our sample script, there were no inserts or deletes, so the signature just links $\ominus$ to $\oplus$. (More formally, the signature is a minimal edge cover of the complete T_1, T_2 bipartite graph; see Section 3.)

If our search for a minimum-cost signature yields the signature of Figure 1(b), it is hard to recover the corresponding minimum-cost edit script. In particular, we note that since there are two dashed edges incident on nodes 4 and 5, we may conclude that these nodes were copied by the edit script. Similarly, since node 4 does not have a "partner" (by dashed edges) whose parent matches the parent of node 4, we may conclude that node 4 is moved. With some bookkeeping, this reasoning recovers the original edit script $\mathcal{E}_1 = (cpy(5,2), cpy(4,6), mov(4,5))$. Unfortunately, this edit script is not a minimum-cost edit script (assuming, say, unit costs for edit operations); the edit script $\mathcal{E}_2 = (mov(4,5), cpy(5,2))$ achieves the same result with one fewer operation. By moving node 4 to under node 5 before node 5 is copied, we get a "free" copy of node 4. Thus, to recover the minimum-cost edit script from the signature we would need to consider all such possibilities of saving operations by "piggy-backing" them on others. As we will see, our parallel transformation model does not have these problems: it is easy to recover a minimum-cost transformation from a signature, making the search for a minimum-cost transformation efficient and simple.

To summarize, our main contributions in this work are as follows: (1) We present a novel, "parallel" tree transformation model that permits expressive operations such as subtree move and copy. (2) We define representative signatures in this model, and describe how they are used by algorithms for finding a minimum-cost transformation between two trees. (3) We present some algorithms (which use our representative signatures) to compute a minimum-cost transformation between two trees. (4) We present empirical performance results for search-based tree-comparison algorithms and heuristics. In this extended abstract, we describe (1) and (2) in detail (relegating some definitions and proofs to the appendix), and sketch (3) briefly. In the full paper, we plan to cover all topics in detail.

2 Transformation Model

Let $\mathcal{N}$ be a domain of node identifiers, and let $\mathcal{L}$ be a domain of labels. A **rooted, unordered, labeled tree** T is a 4-tuple (N, r, p, l), where $N \subset \mathcal{N}$ is called the set of nodes in T, $r \in N$ is a distinguished node, called the **root** of T, $p : N \to N$ is a cycle-free[1] function called the **parent** function of T, and $l : N \to \mathcal{L}$ is called the **label** function of T. Henceforth, by trees we mean rooted, unordered, labeled trees.

In our model, a **transformation** is a set of edit operations (defined below). Each edit operation in a transformation has a unique identifier. In what follows, we often need a way to refer to nodes produced by insertion or copy operations. (For example, we may wish to update a node produced by a copy operation.) We use **node handles** for this purpose. In particular, we use the following notation for node handles: $\texttt{f(}n\texttt{,}i\texttt{)}$, where $n \in \mathcal{N}$ and $i \in Z^+$, refers to the copy of node n produced by the copy operation (with identifier) i; $\texttt{f(0,}i\texttt{)}$ refers to the node produced by insertion operation i; $\texttt{f(}n\texttt{,0)}$ refers to the node n. We denote the set of all node handles by $\mathcal{H}$. (Here and in the sequel, we use `type font` to represent literal strings, and *italics* to represent non-literals.) The **edit operations** on trees are introduced below, along with ana informal description of their effect; the formal definition follows as Definition 3.

Delete: $\texttt{del(}h\texttt{, }j\texttt{)}$, where $h = \texttt{f(}n\texttt{,0)}$ for $n \in \mathcal{N}$. Intuitively, this edit operation deletes the node n. (As we shall see later in Definition 2, the children of n are either deleted, moved, or glued.) In this and the following edit operations, the last argument $j \in Z^+$ is a unique identifier of the edit operation; for brevity, j is often omitted when not needed.

Insert: $\texttt{ins(}h\texttt{, }l\texttt{, }j\texttt{)}$, where $h \in \mathcal{H}$, $l \in \mathcal{L}$, and $j \in Z^+$. Intuitively, this edit operation inserts a node with parent (the node corresponding to) h and label l. (The newly created node has handle $\texttt{f(0,}j\texttt{)}$.)

Update: $\texttt{upd(}h\texttt{, }l\texttt{, }j\texttt{)}$, where $h \in \mathcal{H}$, $l \in \mathcal{L}$, and $j \in Z^+$. Intuitively, this edit operation changes the label of node h to l.

Move: $\texttt{mov(}h_1\texttt{, }h_2\texttt{, }j\texttt{)}$, where $h_1 = \texttt{f(}n\texttt{,0)}$ for $n \in \mathcal{N}$, $h_2 \in \mathcal{H}$, and $j \in Z^+$. Intuitively, this edit operation moves the chunk (defined below) rooted at n, making h_2 its new parent.

Copy: $\texttt{cpy(}h_1\texttt{, }h_2\texttt{, }j\texttt{)}$, where $h_1 = \texttt{f(}n\texttt{,0)}$ for $n \in \mathcal{N}$, $h_2 \in \mathcal{H}$, and $j \in Z^+$. Intuitively, this edit operation copies the chunk rooted at n, making h_2 the parent of the copy.

Glue: $\texttt{glu(}h_1\texttt{, }h_2\texttt{, }j\texttt{)}$, where $h_1 = \texttt{f(}n_1\texttt{,0)}$ for $n_1 \in \mathcal{N}$, $h = \texttt{f(}n_2\texttt{,0)}$ for $n_2 \in \mathcal{N}$, and $j \in Z^+$. Intuitively, glue is the inverse of a copy operation; it causes the chunk rooted at n_1 to disappear by "gluing" it over the chunk rooted at n_2.

[1]By cycle-free, we mean $p^k(n) \neq n$ for any $n \in N$ and $k > 0$.

As noted in Section 1, the first step to applying a transformation is the disassembly of the given tree into "chunks." Chunks, or *disassembly components* (see below), are produced by breaking up the tree at every node that is "operated on" by an edit operation. The break up points are called *disassembly points*.

Definition 1 Given a tree $T = (N, r, p, l)$ and a transformation F, we define the set of **disassembly points**, $dp(T, F)$, as follows:

$$\begin{aligned} dp(T,F) &= \{n \in N \mid n = r \vee \exists\, del(f(n,0)) \in F \vee \\ &\quad \exists\, mov(f(n,0),h) \in F \vee \exists\, cpy(f(n,0),h) \in F \vee \\ &\quad \exists\, glu(f(n,0),h) \in F \vee \exists\, glu(h, f(n,0)) \in F\} \end{aligned}$$

With reference to a transformation F applied to a tree T, we define the **nearest disassembly ancestor** $nda(n, T, F)$ of a node $n \in N$ to be the nearest (not necessarily proper) ancestor of n that belongs to $dp(T, F)$. Further, the **disassembly component** ("chunk") of n is defined as $dc(n, T, F) = \{n' \in N \mid nda(n') = nda(n)\}$. □

When discussing a given tree and transformation, we abbreviate $nda(n, T, F)$ by $nda(n)$, and $dc(n, T, F)$ by $dc(n)$. Not every transformation as defined above can be applied to a given tree. Given a tree T and a transformation F, we define the notion of *validity* of F over T as follows.

Definition 2 A transformation F is said to be **valid** for a tree $T = (N, r, p, l)$ if the following conditions hold.

1. The transformation F is *well-formed*; in particular, the following hold:
   (a) Identifiers of edit operations in F are unique.
   (b) For each node handle $f(n, 0)$ appearing in F: $n \in N$.
   (c) For each $f(0, i)$ in F: $ins(h, l, i) \in F$ for some $h \in \mathcal{H}$ and $l \in \mathcal{L}$.
   (d) For each $f(n, i)$ in F with $i > 0$: $n \in N$, and $cpy(f(nda(n), 0), h, i) \in F$ for some $h \in \mathcal{H}$.
   (e) If $mov(f(n_1, 0), f(n_2, 0)) \in F$, then n_1 is not an ancestor of n_2 in T.
   (f) For each $glu(f(n_1, 0), f(n_2, 0), i) \in F$, there exists an isomorphism g_i between $dc(n_1)$ and $dc(n_2)$. More precisely, there exists a function $g_i : dc(n_1) \to dc(n_2)$ that is one-to-one, onto, preserves the parent function p, and "preserves labels" in the sense that $g_i(x) = y$ implies the following: If $upd(x, l) \in F$ then either $l(y) = l$ or $upd(y, l) \in F$; else either $l(y) = l(x)$ or $upd(y, l(x)) \in F$.
2. For each node n in T, at most one of the following types of edit operations is in F: $del(f(n, 0))$, $cpy(f(n, 0), n')$, $glu(f(n, 0), h)$, and $glu(h, f(n, 0))$. Further, for each $n \in T$, there is at most one operation of the form $del(f(n, 0))$, at most one operation of the form $mov(f(n, 0), h)$, and at most one operation of the form $glu(f(n, 0), h)$ in F. Finally, no node is updated more than once.
3. If F contains $del(f(nda(p(n)), 0))$ or $glu(f(nda(p(n)), 0), h)$ (for some n), then one of $del(f(n, 0))$, $mov(f(n, 0), h)$, and $glu(f(n, 0), h)$ is in F. [2]

□

For ease of explanation, we henceforth assume, without loss of generality, that no edit operation acts on the root of a tree. (We can always add an artificial root to any tree to ensure this property holds.) We are now ready to define the tree $F(T)$ obtained by applying a transformation F to a tree T. Intuitively, we start with a working copy T' of T, and break T' into chunks (i.e., the disassembly components defined above). Nodes deleted by F are removed. Next, copy, update, glue, and move operations in F are applied to the chunks of T'. Nodes corresponding to insertion operations in F are created. Finally, the chunks are reassembled to yield the tree $F(T)$. Formally, we define $F(T)$ as follows:

[2] This condition is not strictly needed, but is used to make deletes (and glues) more symmetrical to inserts (resp., copies). Since any children of inserted (copied) nodes need to be inserted, moved, or copied to that location, we require that any children of a deleted (resp., glued) node be deleted, moved, or glued.

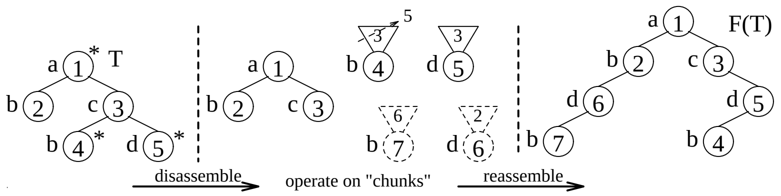

Figure 2: Applying the transformation in Example 1

Definition 3 Given a tree $T = (N, r, p, l)$ and a transformation F valid for T, the result of **applying the transformation** F to T is a tree $F(T) = (N', r', p', l')$ where N', p', r', and l' are defined below. In the following, we use a skolem function $f' : \mathcal{N} \times Z \to \mathcal{N}$ such that $f'(n, i)$ intuitively represents the node in T' referenced by the node handle `f(n, i)` in F.

$$
\begin{aligned}
N' &= \{f'(n,0) \mid n \in N,\ del(f(n,0)) \notin F,\ glu(f(nda(n),0),h) \notin F\} \\
&\cup \{f'(0,i) \mid ins(h,l,i) \in F\} \\
&\cup \{f'(n,i) \mid cpy(f(nda(n),0),h,i) \in F\} \\
r' &= f'(r,0) \\
p'(f'(0,i)) &= f'(n_2,j), \text{ where } ins(f(n_2,j),l,i) \in F \\
p'(f'(n,0)) &= f'(n_2,i), \text{ if } mov(f(n,0),f(n_2,i)) \in F \\
& \quad\ f'(p(n),0), \text{otherwise} \\
p'(f'(n,i)) &= f'(n_2,j), \text{ if } cpy(f(n,0),f(n_2,j),i) \in F \\
& \quad\ f'(p(n),i), \text{ otherwise} \\
l'(f'(0,i)) &= l_1, \text{ where } ins(h,l_1,i) \in F \\
l'(f'(n,i)) &= l_1, \text{ if } upd(f(n,i),l_1) \in F \\
& \quad\ l(n), \text{ otherwise}
\end{aligned}
$$

□

Example 1 Consider the tree T depicted in Figure 2, and the following transformation F: $\{mov(f(4,0), f(5,0)),\ cpy(f(4,0), f(5,101)),\ cpy(f(5,0), f(2,0), 101)\}$. The disassembly points of T by F are marked by an asterisk; they are, intuitively, the nodes in T that are acted on by edit operations in F (in addition to the root). The tree T's disassembly components also shown in the figure; the stubs on the nodes indicate the parent of the chunk. The results of applying the operations in F to the chunks are indicated using dashed lines. In particular, the operation $mov(f(4,0), f(5,0))$ results in the parent of the chunk rooted at node 4 to change from node 3 to node 5. The operation $cpy(f(5,0), f(2,0), 101)$ results in the duplication of the chunk rooted at node 5, producing a new node with identifier 6. (Thus, by our node handle notation, $6 = f(5,101)$.) Similarly, operation $cpy(f(4,0), f(5,101))$ results in a copy of the chunk rooted at node 4. Note that the parent of the copy is node 6 because $f(5,101) = 6$. Finally, the tree $F(T)$ obtained by reassembling the chunks (using the stubs) is also shown.

Note that the result of applying transformation F to tree T is independent of the order in which the edit operations in F are applied. For example, if we had considered applying the move operation after both copy operations, the result would be the same as above. Thus we can intuitively understand the effect of each edit operation on the chunks without worrying about the actions of other edit operations in the transformation. For example, we do not have to worry about a copy operation acting on a chunk resulting in surreptitious copies of other chunks (due to those chunks first being moved to below the copied chunk), as is the case when using the linear edit scripts described in Section 1. □

Recall (from Section 1) that we are interested in finding a minimum-cost transformation between two given trees. We define the **cost of a transformation** to be the sum of the costs of its constituent edit operations. The cost of each edit operation is given by some application dependent function. For example, in an application comparing structured documents, the cost of updating (tree nodes representing) words would depend on how similar the old and new values are. Thus updating "cat" to "cats" may cost 0.1 unit, while updating "cat" to "dinosaur" may cost 2 units. We do not discuss details of the cost model in this work, since our main results do not depend on them.

3 Representative Signatures of Transformations

In Section 1 we introduced signatures as a concise representation of the essential information in a transformation. In particular, given a signature S of a transformation F, one can easily recover a transformation F' that is essentially identical to F. Signatures satisfying this property are called **representative signatures**, and they are useful tools for computing a minimum-cost transformation. (Searching in signature space is more convenient than searching in the space of all possible transformations.) Below, we first define the signature $S(F,T)$ of a transformation F applied to a tree T. We then describe how to recover from $S(F,T)$ a transformation F' that is essentially identical to F (as indicated by Lemmas 1 and 2), thus showing that our signatures are representative.

Intuitively, we may think of generating signatures by using the following procedure: We start with the given tree T and a tree T' that is isomorphic to T. We create signature edges (as distinguished from tree edges) connecting each node in T to its partner in T' (based on the isomorphism). We apply the transformation F to T', updating our set of signature edges in the process as follows: When a node is deleted, signature edges incident on it are redirected to $\ominus$; similarly, we introduce signature edges connecting inserted nodes to $\oplus$. When a subtree is copied, we connect the copy c of a node n to all the nodes to which n is connected; glues are handled analogously. Moves and updates do not affect the set of signature edges. We are then left with a set of signature edges connecting nodes in T to nodes in the transformed T'. Formally, we have the following definition for signatures:

Definition 4 Let F be a transformation that is valid for a tree $T = (N, r, p, l)$, and let $F(T) = (N', r', p', l')$.. We define the **signature** of F on T to be a relation $S(F,T) \subset (N \cup \{\oplus\}) \times (N' \cup \{\ominus\})$ as follows. The function f' is from Definition 3, the function g_j is from Definition 2, and $\oplus$ and $\ominus$ are distinguished reserved nodes in $\mathcal{N}$.

$$\begin{aligned}
S(F,T) \;=\; & \{(\oplus, f'(0,i)) \mid ins(h,l,i) \in F\} \\
\cup\; & \{(n, \ominus) \mid del(f(n,0)) \in F\} \\
\cup\; & \{(n, f'(n,i)) \mid n \in F,\ cpy(f(nda(n),0),h,i) \in F\} \\
\cup\; & \{(n, f'(n',0)) \mid n \in F,\ glu(f(nda(n),0),h,j) \in F,\ g_j(n,n')\} \\
\cup\; & \{(n, f'(n,0)) \mid n \in F,\ del(f(n,0)) \notin F,\ glu(f(nda(n),0),h) \notin F\} \\
\cup\; & \{(\oplus, \ominus) \mid \nexists ins(\ldots) \in F \ \vee\ \nexists del(\ldots) \in F\}
\end{aligned}$$

□

We define the **induced graph** of two trees $T_1 = (N_1, r_1, p_1, l_1)$ and $T_2 = (N_2, r_2, p_2, l_2)$ to be the complete bipartite graph $IG(T_1, T_2) = (U, V, U \times V)$, where $U = N_1 \cup \{\oplus\}$ and $V = N_2 \cup \{\ominus\}$. In the appendix, we show that for any tree T and valid transformation F, $S(F,T)$ is a minimal edge cover of $IG(T, F(T))$. The converse is also true; that is, for every minimal edge cover K of $IG(T_1,T_2)$ there exists some transformation F' such that $F'(T_1) = T_2$, and $S(F', T_1) = K$. Therefore, when searching for the signature of a minimum-cost transformation between two given trees, it suffices to search over the space of all minimal edge covers of their induced graph. Once we have found a minimal edge cover that is the signature of a minimum-cost transformation, we can easily recover the actual transformation from it, as described below.

To recover a transformation from a minimal edge cover K of the induced graph of two trees T_1 and T_2, we proceed in two steps. The first step consists of determining the disassembly points of the required transformation. In the second step, we use these disassembly points to generate the actual edit operations in the transformation. Intuitively, we determine the disassembly points of T_1 and T_2 using K as follows. First, the tree roots, and the special nodes $\oplus$ and $\ominus$ are deemed disassembly points. Next, any node whose partners (by K) are in any way "different" from the partners of its parent is a disassembly point. We say the partners of a node n are "different" from those of its parent $p(n)$ if there is some partner of n whose parent is not a partner of $p(n)$, or vice versa. Finally, any partner of a disassembly point is also a disassembly point. Definition 9 in the appendix presents the formal definition of the **cover disassembly points** of trees T_1 and T_2 by an edge cover K of their induced graph; we denote this set of points by $cdp(T_1, T_2, K)$. (Lemma 4 in the appendix shows that the set of points given by this definition is exactly the set of disassembly points of a minimum-cost transformation whose signature is the given edge cover.)

Once we have determined the disassembly points as described above, we recover the actual transformation as follows: Nodes matched to the special node $\ominus$ are deleted. Nodes matched to $\oplus$ indicate nodes to be inserted. A one-to-one (edge cover) edge incident on a disassembly point signifies a move operation. For a disassembly point on which $k > 1$ (edge cover) edges are incident, we generate $k-1$ copy operations and zero or one move operation. The edge for which a copy operation is not generated is called "distinguished." The choice of this distinguished edge is significant only when we can avoid a move operation by choosing as distinguished edge an edge that connects two nodes whose parents also "match." The precise definition of the **distinguished edge** $de(n)$ incident on a node $n \in T_1 \cup T_2$, based on the above intuition, is in the appendix (Definition 7).

Given two trees T_1 and T_2, and a minimal edge cover K of their induced graph, Definition 5 below presents the details of recovering a transformation F from K based on the intuition described above. Recall that we require that $T_1' = F(T_1)$ be isomorphic to T_2. When generating the edit operations in F, we often need to refer to the node in T_1' that corresponds (by the isomorphism) to a certain node in T_2. As described in Section 2, nodes are referenced in edit operations using node handles; i.e., expressions of the form `f(n,i)`. Thus, we need a way to map each node in $n \in T_2$ to a node handle that represents its partner in T_1'; we call such a node handle the **representative handle** of n, and denote it by $h(n)$. Using the definition of node handles in Section 2, it is easy to observe that the representative handle of a node n in T_2 that is matched to $\oplus$ is $\texttt{f(0, }i\texttt{)}$, where i is the identifier of the insertion operation that produces the node in T_1' that is isomorphic to n. Further, for a node $n \in T_2$ that is matched to exactly one node $m \in T_1$ by a one-to-one edge, we have $h(n) = \texttt{f(}m\texttt{,0)}$. The case in which n is matched to more than one node in T_1 is similar; we simply pick the node m such that $[m, n]$ is the distinguished edge incident on n. Finally, if a node $n \in T_2$ is matched to a node $m \in T_1$ that has more than one edge incident on it, we have two cases: If $[m, n]$ is the distinguished edge incident on m, it means that n is the node corresponding to m, and $h(n) = \texttt{f(}m\texttt{,0)}$; otherwise n represents a copy of m, and $h(n) = \texttt{f(}m\texttt{,}i\texttt{)}$, where i is the identifier of the copy operation that produces the node in T_1' that is isomorphic to n. Definition 8 in the appendix defines $h(n)$ formally based on this intuition.

Definition 5 Let $T = (N, r, p, l)$ and $T' = (N', r', p', l')$ be two trees, and let K be a minimal edge cover of their induced graph $IG(T, T')$. Without loss of generality, let $[r, r']$ be the only edge in K incident on either of r or r'. We define the **transformation induced by the cover** K, denoted by $F(K, T, T')$ as follows, where $m, m' \notin \{r, \oplus\}$, $n \notin \{r', \ominus\}$, $E(m) = \{[m, n] \in K\} \cup \{[n, m] \in K\}$, and $cdp(T, K)$ is a short-hand for $cdp(T, T'K) \cap T$. (We use `type font` to represent literal strings, and *italic font* to represent non-literals, and where σ is a function that maps edges to arbitrary, unique, positive integers.)

$$
\begin{aligned}
F(K,T,T') \;=\;& \{\texttt{del(f(}m\texttt{,0))} \mid [m,\ominus] \in K\} \\
\cup\;& \{\texttt{ins(}h(p(n)), l(n), i) \mid [\oplus, n] \in K\} \\
\cup\;& \{\texttt{mov(f(}m\texttt{,0)}, h(p(n))) \mid m \in cdp(T,K),\ [m,n] \in K, |E(m)| = |E(n)| = 1\} \\
\cup\;& \{\texttt{mov(f(}m\texttt{,0)}, h(p(n))) \mid m \in cdp(T,K),\ [m,n] \in K,\ |E(m)| > 1,\ de(m) = [m,n], \\
& \quad de(cnda(p(m))) = [cnda(p(m)), n'] \neq de(n')\} \\
\cup\;& \{\texttt{cpy(f(}m\texttt{,0)}, h(p(n)), \sigma(\texttt{[m,n]})) \mid m \in cdp(T,K),\ [m,n] \in K, |E(m)| > 1,\ [m,n] \neq de(m)\} \\
\cup\;& \{\texttt{mov(f(}m\texttt{,0)}, h(p(n))) \mid m \in cdp(T,K),\ [m,n] \in K,\ |E(n)| > 1,\ de(n) = [m,n], \\
& \quad de(p(n)) \neq [p(m'), p(n)]\} \\
\cup\;& \{\texttt{glu(f(}m\texttt{,0)}, \texttt{f(}m'\texttt{,0))} \mid m \in cdp(T,K),\ [m,n] \in K,\ |E(n)| > 1,\ [m',n] = de(n),\ m \neq m'\} \\
\cup\;& \{\texttt{upd(}h(n), l(n)) \mid [m,n] \in K,\ [m,n] \neq de(m),\ l(m) \neq l(n),\ |E(m)| > 1\}
\end{aligned}
$$

□

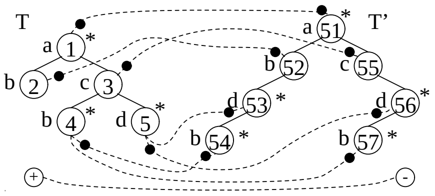

Figure 3: The trees in Example 2

Example 2 Let T be the initial tree from Example 1, and let T' be a tree isomorphic to the final tree $F(T)$ there, as shown in Figure 3. (Note that in Example 1 the final tree is obtained by modifying the initial tree. Here, the two trees are not related in this manner; hence the tree nodes do not share identifiers.) A minimal edge cover of their induced graph is indicated using dashed lines. (Note that this edge cover is in fact the signature of the transformation in Example 1.) The cover disassembly points, computed using Definition 9, are marked by asterisks. The distinguished edge incident on each node is marked by a small filled circle on that edge near the corresponding node. Since there are only two nodes with more than one edge incident on them, the choice of a distinguished edge is nontrivial in these two cases only. Intuitively, for node 5, we observe that the parent of node 56 is matched to the parent of node 5; therefore the edge $[5, 56]$ is chosen as distinguished. Node 4 is not matched to any node whose parent matches the parent of node 4; therefore, we select a distinguished edge arbitrarily from those incident on node 4, say $[4, 53]$. (Definition 7 in the Appendix describes the choice formally.)

Using this information about the cover disassembly points and distinguished edges, we now use Definition 5 to obtain a transformation. The edge $[4, 54]$ satisfies the conditions in line 4 of the equation in Definition 5, resulting in the operation `mov(f(4,0),` $h(p(54))$`)`. Now from Figure 3 we observe that $p(54) = 53$. Further, node 53 is matched to node 5, but $[5, 53]$ is not the distinguished edge incident on node 5 (i.e., $de(5) \neq [5, 53]$); therefore, the representative handle of node 53 is `f(5,`$\sigma([5, 53])$`)`, where σ is simply a function that generates a unique identifier for each edge. Say $\sigma([5, 53]) = 501$, so that $h(p(54)) = h(53) =$ `f(5,501)`, giving `mov(f(4,0), f(5,501))` as the edit operation generated corresponding to edge $[4, 54]$. Next, observe that edge $[5, 53]$ satisfies the conditions in line 6 of Definition 5, resulting in the operation `cpy(f(5,0), f(2,0), 501)`, since $h(p(53)) = h(52) =$ `f(2,0)`, and $\sigma([5, 53]) = 501$. A similar process for the edge $[4, 57]$ results in the operation `cpy(f(4,0), f(5,0))`. Definition 5 does not generate any more operations, giving {`mov(f(4,0), f(5,501)), cpy(f(5,0), f(2,0), 501), cpy(f(4,0), f(5,0))`} as the recovered transformation.

Observe that the transformation recovered above is essentially identical to that in Example 1. Apart from edit operation identifiers, the only difference is that instead of moving the node 4 to below the original instance of node 5 (and copying node 4 to below the copy of node 5) as done by that transformation, the above transformation moves node 4 to below the copy of node 5 (copying node 4 to below the original instance of node 5). This difference is a result of the freedom in the choice of a distinguished edge incident on node 4. □

We now state our main results, proved in the appendix, showing that the transformation F' recovered by Definition 5 from the signature $S(F, T)$ of a transformation F on tree T is essentially identical to F:

Lemma 1 *Let T and T' be two trees, and let K be a minimal edge cover of their induced graph $IG(T, T')$. Then (1) $F(K, T, T')$ is a valid transformation for T, (2) $F(K, T, T')(T)$ is isomorphic to T', and (3) $S(F(K, T, T'), T)$ is isomorphic to K.* □

Lemma 2 *Let T be a tree, let F be a transformation that is valid for T, and let $F' = F(S(F, T), T, F(T))$. Then F' has the same number of move, copy, and glue operations as F (respectively), and the insert, delete, and update operations in F' are identical to those in F, modulo edit operation identifiers.* □

4 Conclusion

When managing tree-structured data (e.g., structured query results, programs, documents, Web sites, circuit designs, and file systems), one often needs to find differences between related data (e.g., results of running a query at different times, two similar circuits, or different versions of a program or document). Such tree differences can be compactly and effectively captured by the novel transformation model we have presented here. Our model includes expressive subtree operations, such as move and copy, which make the detected differences more meaningful to a user. This model also admits representative signatures, which are compact representations of the essential information in transformations. These signatures make it possible to search for a minimum-cost transformation by searching instead for a minimum-cost signature, knowing that each signature can be mapped back to a transformation. As discussed in Section 1 and below, working with transformation signatures greatly simplifies algorithms for computing minimum-cost transformations. Our model, results, and strategy for computing a minimum-cost transformation are independent of the details of the cost model used. Furthermore, although in this paper we have focused on unordered trees, the results adapt easily to ordered trees, making our scheme widely applicable.

A general approach to computing a minimum-cost signature, without using application- or domain-specific features, is to use search-based techniques and heuristics. Recall from Section 3 that the signature of any transformation between the input trees T_1 and T_2 is a minimal edge cover of their induced graph (which is a bipartite graph that has an edge between every node in T_1 and every node in T_2). Thus, the search space that we need to explore is the space of all possible minimal edge covers of this bipartite graph. Note that this is a much simpler search space than the search space of all possible transformations between T_1 and T_2. This simplification is a result of the existence of representative signatures in our transformation model. Very often, many edges in the induced graph can be eliminated by using "pruning rules" that (conservatively) detect cases when two nodes can never be partners in any minimum-cost transformation (using upper and lower bounds on the contribution of an induced graph edge to the cost of a signature) [CGM97]. (In addition, we may optionally decide to use aggressive pruning rules that prune edges if it is "very unlikely" that the corresponding nodes could be partners.) Eliminating edges from the induced graph greatly reduces the size of the search space. Next, we use estimates of the cost contribution of induced graph edges to compute a minimum-(estimated)cost edge cover of the pruned induced graph. Finally, we search for a better signature in the neighborhood of this initial edge cover, using techniques similar to those in [WZC95, SWZS94]. In the full version of the paper we will present experimental performance results that compare various pruning heuristics, and that show how parameters such as tree size and depth, and level of homogeneity in the node labels and the shape of the trees, impact the performance and quality of the signature found.

Due to the simplicity of the relation between signatures and transformation in our model, we are able to derive tighter bounds on the edge costs described above than those possible in the linear edit script model. (With linear edit scripts, cost bounds need to take into account possible "piggy-backing" of edit operations.) These tighter bounds lead to more effective pruning of the induced graph, and thus give us better performance. This simplicity also allows us to easily derive better estimates for edge costs in the induced graph, thus improving the quality of the initial solution, and the effectiveness of the subsequent search process.

We can often further reduce the size of the search space of signatures by using features of the application domain. For example, consider an application comparing structured documents. Such documents are often represented using layered, ordered trees, with layers corresponding to structural elements (such as words, sentences, paragraphs, and sections). That is, each tree node has an immutable type, and the tree is layered by a partial order on these types. (For example, sentences are below paragraphs and sections.) Therefore we do not need to consider any signature that matches nodes of different types to each other. This fact leads to very effective pruning of the induced graph, and a corresponding reduction in the size of the search space of its minimal covers.

Finally, we can often use domain characteristics in conjunction with the properties of representative signatures to permit the exact computation of the contribution of an edge in the induced graph to the total cost of the signature (whereas in general we use estimates). Consequently, the initial solution that was earlier the estimated minimum-cost signature is now the actual optimal solution, so that the subsequent search phase is unnecessary. One such scheme (similar in spirit to the restrictions on matchings used in [Yan91, ZS89, ZWS95]) results in a simple bottom-up dynamic programming algorithm that produces optimal solutions if moves, copies, and glues are restricted to be "local." (A restriction of local copies, for instance, disallows a paragraph from being copied outside its section.) Even if these restrictions do not strictly hold in a given application domain, we may intuitively expect such algorithms produce solutions that are close to optimal. We plan to present the details of such algorithms in the full version.

References

[C3] Overview of the Stanford C^3 project. Available at `http://www-db.stanford.edu/c3/`.

[CGL+97] S. Chawathe, V. Gossain, X. Liu, J Widom, and S. Abiteboul. Representing and querying changes in heterogeneous semistructured databases (demonstration description). Available at `http://www-db.stanford.edu`, November 1997.

[CGM97] S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 26–37, Tuscon, Arizona, May 1997.

[EG] The Gate eGuide. Available at `http://www.sfgate.com/eguide/`.

[Mye86] E. Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.

[NBR88] M. Neff, R. Byrd, and O. Rizk. Creating and querying hierarchical lexical data bases. In *Proceedings of the 2nd Conference on Applied Natural Language Processing*, pages 84–93, 1988.

[SWZS94] D. Shasha, J. Wang, K. Zhang, and F. Shih. Exact and approximate algorithms for unordered tree matching. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4):668–678, April 1994.

[SZ90] B. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Comput. Appl. Biosci.*, 6:309–318, 1990.

[WCM+94] J. Wang, G. Chirn, T. Marr, B. Shapiro, D. Shasha, and K. Zhang. Combinatorial pattern discovery for scientific data: some preliminary results. In *Proceedings of the ACM SIGMOD Conference*, pages 115–125, May 1994.

[WF74] R. Wagner and M. Fischer. The string-to-string correction problem. *Journal of the Association of Computing Machinery*, 21(1):168–173, January 1974.

[WSC+97] J. Wang, D. Shasha, G. Chang, L. Relihan, K. Zhang, and G. Patel. Structural matching and discovery in document databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 560–563, 1997.

[WZC95] J. Wang, K. Zhang, and G. Chirn. Algorithms for approximate graph matching. *Information Sciences*, 82:45–74, 1995.

[WZJS94] J. Wang, K. Zhang, K. Jeong, and D. Shasha. A system for approximate tree matching. *IEEE Transactions on Knowledge and Data Engineering*, 6(4):559–571, August 1994.

[Yan91] W. Yang. Identifying syntactic differences between two programs. *Software—Practice and Experience*, 21(7):739–755, July 1991.

[ZS89] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18(6):1245–1262, 1989.

[ZWS95] K. Zhang, J. Wang, and D. Shasha. On the editing distance between undirected acyclic graphs. *International Journal of Foundations of Computer Science*, 1995.

A Supporting Definitions and Proofs

Definition 6 Given a bipartite graph $B = (U, V, E)$, with distinguished nodes $\oplus \in U$ and $\ominus \in V$, a set $K \subseteq E$ is called an **edge cover** of B if each node in $U \cup V$ is incident on at least one edge in K. The set K is said to be a **minimal edge cover** if it is an edge cover that (1) does not contain any paths of length three, and (2) does not contain any paths of length two ending at $\oplus$ or $\ominus$.[3] □

Lemma 3 *For any transformation F valid for a tree T, $S(F,T)$ is a minimal edge cover of $IG(T, F(T))$.* □

Proof: Let us first show that each node in $IG(T, F(T))$ is incident on at least one edge in $S(F,T)$. If there is an insert operation in F, then $[\oplus, f'(0,i)] \in S(F,T)$; if not, $[\oplus, \ominus] \in S(F,T)$. Thus, $\oplus$ is covered by $S(F,T)$. An analogous argument holds for $\ominus$. Now consider any node $n \in T$. If $del(f(n,0)) \in F$, then $[n, \ominus] \in S(F,T)$; else if $glu(f(n,0), f(n',0)) \in F$, then $[n, f'(n',0)] \in S(F,T)$; else $[n, f'(n,0)] \in S(F,T)$. Thus, in all cases n is covered by $S(F,T)$. Now consider a node $n' \in T'$. If $n' = f'(0,i)$, then $ins(h,l,i) \in F$, implying $[\oplus, n'] \in S(F,T)$; else if $n' = f'(n,0)$, then (from Definition 3) $del(f(n,0)) \notin F$ and $glu(f(n,0),h) \notin F$ implying $[n,n'] \in S(F,T)$; else $n' = f'(n,i)$ for $i > 0$, implying $cpy(f(n,0),h,i) \in F$ so that $[n,n'] \in S(F,T)$. Thus, in all cases n' is covered by $S(F,T)$. We have thus shown that each node in $IG(T, F(T))$ is covered by $S(F,T)$.

Let us now show that $S(F,T)$ is a *minimal* edge cover of $IG(T, F(T))$. We first show that there is no path of length two ending at $\oplus$ or $\ominus$. Consider an edge $[\oplus, n']$. From Definition 4, it follows that $n' = f'(0,i)$, for some edit operation identifier i. Using the uniqueness of edit operation identifiers, we see that there can be no other edge incident such a node n'. Thus there are no paths of length two terminating at $\oplus$. An analogous argument shows that there are no paths of length two terminating at $\ominus$.

We now show that there are no paths of length three in $S(F,T)$. Let, if possible, n_1, n_2, n_3, n_4 be a path of length three in $S(F,T)$ such that $n_1 \in T$, implying $n_2 \in F(T)$, $n_3 \in T$, and $n_4 \in F(T)$. Since we have shown that there are no paths of length two incident on $\oplus$ or $\ominus$, it follows that $n_i \neq \oplus, \ominus$, for $i = 1 \ldots 4$. From Definition 4, we see that if $n \in T$ is a node with multiple edges in $S(F,T)$ incident on it, then $nda(n)$ is acted on by a copy operation in F. Now n_3 is such a node, implying $cpy(nda(n_3), x) \in F$. We also observe that if $n' \in F(T)$ is a node with multiple edges $\{[m_i, n']\}$ $(i = 1 \ldots k,\ k > 1)$ in $S(F,T)$ incident on it, then $n' = f'(m_{i^*}, 0)$, where $i^* \in [1,k]$, and $glu(nda(m_i), nda(m_{i^*})) \in F$ for all $m_i \neq m_{i^*}$; thus there is a glue operation acting on each $nda(m_i)$, $i \in 1 \ldots k$. Now n_2 is such a node, with edges $[n_1, n_2]$ and $[n_3, n_2]$ incident on it. Therefore, there is a glue operation acting on $nda(n_3)$. Thus $nda(n_3)$ is acted on by both a copy and a glue operation, contradicting the validity of F (Definition 2). We therefore conclude that no such path exists in $S(F,T)$, proving minimality.

□

Lemma 4 *Let $T = (N, r, p, l)$ and $T' = (N', r', p', l')$ be two trees, and let K be a minimal edge cover of their induced graph $IG(T, T')$. Then $cdp(T,K) = dp(T, F(K,T,T'))$.* □

Proof: Let us first show that $cdp(T,K) \subseteq dp(T, F(K,T,T'))$. Let m be any node in $cdp(T,K)$. If $m = r$, then $m \in dp(T, F(K,T,T'))$ since the root is always included in $dp(T, F(K,T,T'))$ (Definition 1). If $[m, \ominus] \in K$, then $del(m) \in F(K,T,T')$ (by Definition 5), implying $m \in dp(T, F(K,T,T'))$. Now for all $m \in cdp(T,K)$ other than those considered above, Definition 5 generates either a *mov*, *cpy*, or *glu* operation, implying $m \in dp(T, F(K,T,T'))$. Thus $cdp(T,K) \subseteq dp(T, F(K,T,T'))$.

Let us now show that $dp(T, F(K,T,T')) \subseteq cdp(T,K)$. Let m be any node in $dp(T, F(K,T,T'))$. If $m = r$, $r \in cdp(T,K)$ as required. If $del(m) \in F(K,T,T')$, Definition 4 implies $[m, \ominus] \in K$, implying

[3] Our definition implies that no proper subset of a minimal edge cover is an edge cover.

$m \in cdp(T, K)$. Otherwise, either $mov(m, h)$, $cpy(m, h)$, $glu(h, m)$, or $glu(m, h)$ is in $F(K, T, T')$, implying $m \in cdp(T, K)$. Thus $dp(T, F(K, T, T')) \subseteq cdp(T, K)$ which, with our earlier result, completes the proof. □

Lemma 5 *Let $T = (N, r, p, l)$ and $T' = (N', r', p', l')$ be two trees, and let K be a minimal edge cover of their induced graph $IG(T, T')$. If $m, n \notin cdp(T, K)$, $m \neq \ominus$, $m \neq r$, $n \neq \oplus$, and $n \neq r'$, then $[p(m), p'(n)] \in K$. Consequently, we have $[cnda(m), cnda(n)] \in K$ for all nodes $m \in N$, $n \in N'$.* □

Proof: Follows from Definition 9. □

Definition 7 Let $T = (N, r, p, l)$ and $T' = (N', r', p', l')$ be two trees, and let K be a minimal edge cover of their induced graph $IG(T, T')$. Without loss of generality, let $[r, r']$ be the only edge in K incident on either of r or r'. Let $E(n)$ denote the edges in K that are incident on a node $n \in N \cup N'$. We define the **matched edge set** of a node $n \in N \cup N' - \{r, r'\}$ such that $[\oplus, n], [n, \ominus] \notin K$ as the set $E'(n)$ below:

$$\begin{aligned} E'(n) \quad &= \quad \{[n, n'] \in E(n) \ \mid \ [p(n), p(n')] \in K\}, \text{ if } n \in N \\ &\quad\quad \{[n', n] \in E(n) \ \mid \ [p(n'), p(n)] \in K\}, \text{ otherwise} \end{aligned}$$

Further, let us define the **distinguished edge** $de(n)$ incident on any $n \in cdp(T, K)$ as follows:

$$\begin{aligned} de(n) \quad = \quad & e, \text{ if } E(n) = \{e\} \\ & \text{an arbitrary edge in } E'(n), \text{ if } |E(n)| > 1, \ E'(n) \neq \emptyset \\ & \text{an arbitrary edge in } E(n), \text{ otherwise} \end{aligned}$$

Note that our definition of de implies $de(r) = de(r') = [r, r']$, $de(m) = [m, \ominus] \quad \forall [m, \ominus] \in K$, and $de(n) = [\oplus, n] \quad \forall [\oplus, n] \in K$. Furthermore, we extend the definition of de to all nodes in $N \cup N'$ by defining $de(n)$ for $n \in N \cup N' - cdp(T, K)$ as follows:[4]

$$de(n) \quad = \quad [m, n], \text{ where } de(cnda(n)) = de(cnda(m))$$

□

Definition 8 Let $T = (N, r, p, l)$ and $T' = (N', r', p', l')$ be two trees, and let K be a minimal edge cover of their induced graph $B = IG(T, T') = (U, V, E)$. Without loss of generality, let $[r, r']$ be the only edge in K incident on either of r or r'. We define the **representative handle** $h(n)$ of a node $n \in T'$ as follows, where we use `type font` to represent literal strings, and *italic font* to represent non-literals, and where σ is a function that maps edges to arbitrary, unique, positive integers:

$$\begin{aligned} h(n) \quad = \quad & \mathtt{f(0,}\sigma(de(n))\mathtt{)}, \text{ if } de(n) = [\oplus, n] \\ & \mathtt{f(}m\mathtt{,0)}, \text{ if } de(n) = [m, n] = de(m) \\ & \mathtt{f(}m\mathtt{,}\sigma(de(cnda(n)))\mathtt{)}, \text{ if } de(n) = [m, n] \neq de(m) \end{aligned}$$

□

Definition 9 Let $T_1 = (N_1, r_1, p_1, l_1)$ and $T_2 = (N_2, r_2, p_2, l_2)$ be two trees and let K be a minimal edge cover of their induced graph $B = IG(T_1, T_2)$. We define the **cover disassembly points** of T_1 and K as the following set: $cdp(T_1, K) \subset N$.

$$\begin{aligned} cdp(T_1, K) \quad = \quad & \{r\} \\ \cup \quad & \{m \in N \ \mid \ [m, \ominus] \in K\} \\ \cup \quad & \{m \in N \ \mid \ \exists [m, n] \in K : [p(m), p(n)] \notin K\} \\ \cup \quad & \{m \in N \ \mid \ \exists [m, n] \in K : (\exists [m', n] \in K : [p(m'), p(n)] \notin K)\} \\ \cup \quad & \{m \in N \ \mid \ \exists [m, n_1], [m, n_2] \in K : p(n_1) = p(n_2)\} \\ \cup \quad & \{m \in N \ \mid \ \exists [m, n] \in K : (\exists [m_1, n], [m_2, n] \in K : p(m_1) = p(m_2))\} \\ \cup \quad & \{m \in N \ \mid \ \exists [p(m), n] \in K : (\nexists [m, n'] \in K : p(n') = n)\} \\ \cup \quad & \{m \in N \ \mid \ \exists [m, n] \in K : ([m', n] \in K : (\nexists [m'', n] \in K : p(m'') = m))\} \end{aligned}$$

[4]From the definition of cdp and $cnda$, it is easy to observe that for any $n \in N \cup N' - cdp(T, K)$, there is exactly one node m such that $de(cnda(n)) = de(cnda(m))$.

We define the cover disassembly points of T_2 and K as $cdp(T_2, K) = \{n \in T_2 \mid [m,n] \in K,\ m \in cdp(T_1, K)\} \cup \{n \in N' \mid [\oplus, n] \in K\}$. We also define the **cover nearest disassembly ancestor**, $cnda(n)$, and the **cover disassembly component**, $cdc(n)$, of a node $n \in T_1 \cup T_2$ analogously to the corresponding definitions in Definition 1. □

Lemma 6 *Let T and T' be two trees, and let K be a minimal edge cover of their induced graph $IG(T, T')$. Then (1) $F(K, T, T')$ is a valid transformation for T, (2) $F(K, T, T')(T)$ is isomorphic to T', and (3) $S(F(K, T, T'), T)$ is isomorphic to K.* □

Proof: Part (1): Let us first show that $F = F(K, T, T')$ is a valid transformation for T. The conditions 1(a-e) and 2 in Definition 2 are easy to verify. Let us consider condition 3 for some node n such that $del(n) \in F$, and any child c of n. If $[c, \ominus] \in K$, $del(f(c, 0)) \in F$. Otherwise $[c, y] \in K$ for some $y \in T'$. Since the only edge in K incident on $n = p(c)$ is $[n, \ominus]$ (due to minimality of K and Definition 6), it follows from Definition 9 that $m \in cdp(T, K)$. Definition 5 shows that every node $m \in cdp(T, K)$ such that $[m, \ominus] \notin K$ is acted on by a mov or glu operation in F. Thus condition 3 is satisfied. Finally, let us verify condition 1(f) of Definition 2. If $glu(f(m_1, 0), f(m_2, 0), i) \in F$ then $[m_1, n], [m_2, n] \in K$ due to Definition 5. Let x be any node in $dc(m_1)$, implying that m_1 is an ancestor of x. Consider first the case when $p(x) = m_1$. Since $x \notin dp(T, F)$, it follows from Lemma 4 and Definition 9 that $\exists! [x, y] \in K$ such that $p(y) = n$.[5] Furthermore, $x \notin dp(T, K)$ implies $y \notin dp(T, F) = cdp(T, K)$, so that $\exists! [x', y] \in K$ such that $p(x') = m_2$. Thus for any child x of m_1, we determine uniquely a corresponding child x' of m_2, and we define $g_i(x) = x'$. By using induction on the depth of a node x in the subtree $dc(m_1)$, we extend the definition of g_i to all nodes in $dc(m_1)$. We thus have a one-to-one function $g_i : dc(m_1) \rightarrow dc(m_2)$ that preserves the parent function; by symmetry, it follows that g_i is also an onto function. Finally, it is easy to verify that g_i "preserves labels" in the sense of Definition 2. Therefore F is a valid transformation for T.

Part (2): Let us now show that $F(T) = (N'', r'', p'', l'')$ is isomorphic to $T' = (N', r', p', l')$. Define a function $h' : N' \rightarrow N''$ intuitively reflecting the representative handle function h in Definition 8 as follows:

$$\begin{aligned} h'(n) \quad = \quad & f'(0, i), \text{ if } h(n) = f(0, i) \\ & f'(m, 0), \text{ if } h(n) = f(m, 0) \\ & f'(m, i), \text{ if } h(n) = f(m, i) \end{aligned}$$

We claim that h' is an isomorphism from $F(T)$ to T'. Since there is exactly one distinguished edge $de(n)$ incident on any node $n \in F(T)$, it follows that h, and hence h', is a **one-to-one** function.

Now let us show that h' is an **onto** function. Consider any $m' \in F(T)$, and the possibilities according to Definition 3. If $m' = f'(0, i)$, we know that $ins(\ldots, i) \in F(T)$, implying $de(n) = [\oplus, n]$ and $\sigma(de(n)) = i$, so that $h(n) = f(0, i)$ and $h'(n) = m'$. If $m' = f'(m, 0)$, we know $m \in T$ and $del(f(m, 0)), glu(cnda(m), \ldots) \notin F$. Thus $[m, \ominus] \notin S(F, T)$ and $\exists n = cnda(n) \in T' : de(cnda(m)) = [cnda(m), n] = de(n)$ (since $de(cnda(m)) \neq de(cnda(n))$ implies $glu(f(cnda(m), 0), \ldots) \in F$), implying $h(n) = f(m, 0)$ and $h'(n) = f'(m, 0) = m'$. Finally, if $m' = f'(m, i)$, $i > 0$ then $cpy(f(cnda(m), 0), ...) \in F$, so that $\exists n' \in T' : de(n') = [cnda(m), n] \neq de(cnda(m))$. Now using the definition of $cdp(T, K)$ and the fact that $m \notin cdp(T, K)$, it is easy to observe that $\exists n \in cdc(n') : de(n) = [m, n] \neq de(m)$, implying $h(n) = f(m, i)$ and $h'(n) = f'(m, i)$.

We shall now show that h' **preserves the parent function**; that is, we shall show that $h'(p'(n)) = p''(h'(n))$ for all $n \in N'$, $n \neq r'$. If $h'(n) = f'(0, i)$ then $ins(h(p(n)), l(n), i) \in F$, implying $p''(f'(0, i)) = h'(p'(n))$ as needed. If n (and therefore $h(n)$) is not a disassembly point, clearly $p''(h'(n)) = h'(p'(n))$ by Definition 3. If n (and therefore $h'(n)$) is a disassembly point, we have $n = cnda(n)$, $m = cnda(m)$, and the following two cases:

Case 1: $h'(n) = f'(m, 0)$. In this case, $h(n) = f(m, 0)$, implying $de(n) = [m, n]$ and $de(m) = de(n)$ (since m and n are disassembly points) by the definition of the handle function h. Using the definition of $F(K, T, T')$, it is easy to observe that for all the three of the possibilities (1) $|E(n)| = |E(m)| = 1$, (2) $|E(n)| = 1$, $|E(m)| > 1$, and (3) $|E(n)| > 1$, $|E(m)| = 1$, $mov(f(m, 0), h(p(n))) \in F$, implying $p''(f'(m, 0)) = h'(p(n))$ as needed.

[5] We use $\exists!$ to denote "there exists a unique."

Case 2: $h'(n) = f'(m,i),\ i > 0$. In this case, $cpy(f(m,0), h(p(n))) \in F$, implying $p''(f'(m,i)) = h'(p(n))$ as needed.

From the definition of $h'(n)$, $h(n)$, and $F(K,T,T')$, it is easy to see that h' **preserves the label function** l. Thus, h' is the required isomorphism between T' and $F(T)$.

Part (3): Now let us show that $S = S(F,T)$ is isomorphic to K; more precisely, we show that $[m,n] \in K$ if and only if $[m, h'(n)] \in S(F,T)$.[6] It is easy to observe that $[\oplus, \ominus] \in K$ if and only if $[\oplus, \ominus]$ in S; therefore we will exclude this special edge from our discussion below.

Consider any $[m,n] \in K$. We will show that $[m, h'(n)] \in S$. If $m = \oplus$, $ins(\ldots, i) \in F$, so that $[\oplus, f'(0,i)] = [m, h'(n)] \in S$ as required. Otherwise, we have two cases:

Case 1: $h'(n) = f'(m,0)$. If $de(n) = [m', n],\ m' \neq m$ then $[cnda(m), cnda(n)]$, $[cnda(m'), cnda(n)] \in K$ (due to Lemma 5), implying $glu(f(cnda(m),0), f(cnda(m'),0))$ so that $[m, f'(m,0)] \in S$. Otherwise $de(n) = [m,n]$, implying $glu(f(m,0), \ldots) \notin F$. Now if $\exists m' \in N : glu(f(m',0), f(m,0)) \in F$, we can argue $[m, f'(m,0)]$ as before; else the absence of glu and del operating on m gives $[m, f'(m,0)]$ as required.

Case 2: $h'(n) = f'(n,i),\ i > 0$. From the definition of the representative handle function h and the copy operation-generating part of the definition of $F(K,T,T')$ (Definition 8), we see that $\exists cpy(f(cnda(m),0), m', i) \in F$. Therefore, $[m, f'(m,i)] \in S$ as required.

Thus $[m,n] \in K \Rightarrow [m, h'(n)] \in S$. Now since $h' : N' \to N''$ is an isomorphism, all edges in S are of the form $[m, h'(n)]$, where $m \in N \cup \{\oplus\}$ and $n \in N' \cup \{\ominus\}$. Consider any such edge. If $m = \oplus$ then $h'(n) = f'(0,i)$, implying $[\oplus, n] \in K$ by Definition 8. If $h'(n) = f'(m,0)$ then $h'(n) = m$ gives $[m,n] \in K$. Otherwise, $h'(n) = f'(m,i),\ i > 0$, implying $[m,n] \in K$ again. Thus $[m, h'(n)] \in S \Rightarrow [m,n] \in K$, which together with $[m,n] \in K \Rightarrow [m, h'(n)] \in S$, shows that S and K are isomorphic. □

Lemma 7 *Let T be a tree, let F be a transformation that is valid for T, and let $F' = F(S(F,T), T, F(T))$. Then F' has the same number of move, copy, and glue operations as F (respectively), and the insert, delete, and update operations in F' are identical to those in F, modulo edit operation identifiers.* □

Proof: Let $T = (N, r, p, l)$, $T' = F(T) = (N', r', p', l')$. Consider first any **insert** operation $ins(h(p'(n)), l(n), i)$ in F'. From the definition of $F(K,T,T')$, we know that $[\oplus, n] \in S(F,T)$ such that $\sigma([\oplus, n]) = i$, which in turn implies $ins(h,l,i) \in F$ and $n = f'(0,i)$ due to Definition 4. Since $l(f'(0,i)) = l$, it follows that $l = l(n)$. If $h = f(n_2, 0)$, $p'(n) = f'(n_2, 0)$; else $h = f(n_2, i)$ and $p'(n) = f'(n_2, i)$; in either case, $h = h(p'(n))$. Thus $ins(h(p'(n)), l(n), i) \in F$.

Consider any **delete** operation $del(f(m,0)) \in F'$. From the definition of $F(K,T,T')$, we obtain $[m, \ominus] \in S(F,T)$, which in turn implies $del(f(m,0)) \in F$ due to Definition 4. The above arguments for insert and delete operations can also be repeated in the reverse direction.

Consider a node $m \in T$ such that $|E(m)| = k > 1$. Let the set of edge-cover edges incident on m be $E(m) = \{[m, n_i]\}_{i=1}^{k}$. Now since the edge cover is actually $S(F,T)$, Definition 4 implies that there are exactly $k-1$ copy operations of the form $cpy(f(m,0), h)$ in F. It is easy to observe that the definition of $F(K,T,T')$ generates exactly $k-1$ copy operations for such a node m. Since the above argument can be repeated for each node $m \in T$, we conclude that the number of **copy** operations in F is equal to that number in F'.

The argument for **glue** operations is analogous to the above argument for copy operations: Consider a node $n \in T'$ such that $|E(n)| = k > 1$. Let the set of edge-cover edges incident on n be $E(n) = \{[m_i, n]\}_{i=1}^{k}$. Now since the edge cover is actually $S(F,T)$, Definition 4 implies that there are exactly $k-1$ glue operations of the form $glu(f(m_j, 0), f(m', 0))$ in F, where $m' \in \{m_i\}$ and $m_j \in \{m_i\}$ for $j = 1 \ldots k-1$. It is easy to observe that the definition of $F(K,T,T')$ generates exactly $k-1$ glue operations corresponding to such a node n. Since the above argument can be repeated for each node $n \in T'$, we conclude that the number of glue operations in F is equal to that number in F'.

Now consider **move** operations. Consider first any move operation in the third subset of the definition of $F(K,T,T')$. We know that $m \in cdp(T,K)$, so that $m \in dp(T,F)$ by Lemma 4. From Definition 1, we see that a non-root node is in $dp(T,F)$ only if it is acted on by some edit operations other than update. Now, m cannot

[6] We extend h' by defining $h'(\ominus) = \ominus$ for notational convenience.

be deleted, because that would imply $[m, \ominus] \in S(F,T)$. Furthermore, the fact that $|E(m)| = |E(n)| = 1$ indicates that m cannot be subject to a copy or a glue operation. Consequently, it must be the case that m is moved by F.

Now consider any move operation $mov(f(m,0), h(p(n)))$ in the fourth subset of the definition of $F(K,T,T')$. We know that $|E(m)| > 1$; let $E(m) = \{[m, n_i]\}_{i=1}^{k}$ where $k > 1$. As we have seen above, there are $k-1$ copy operations of the form $cpy(f(m,0), \eta)$ in F. Let $[m, n^*]$ be the unique edge in $E(m)$ that does not correspond to a copy operation. Suppose $mov(f(m,0), \eta) \notin F$. Then $p'(f'(m,0)) = f'(p(m),0)$ by Definition 3. Now if either $del(f(p(m),0)) \in F$ or $glu(f(p(m),0), h') \in F$, the validity of F implies $mov(f(m,0), h) \in F$ (due to Definition 2), contradicting our assumption. Therefore, it must be the case that $del(f(p(m),0)) \notin F$ and $glu(f(p(m),0), h') \notin F$, in turn implying $f'(p(m),0) \in N'$ due to Definition 3. Now, using Definition 4, the above facts yield $[p(m), f'(p(m),0)] \in S(F,T) = K$, in turn implying $E'(m) \neq \emptyset$ in Definition 5, which gives $de(m) \in E'(m)$, contradicting the condition in the fourth subset of Definition 5. Therefore, we conclude that F contains a move operation $mov(f(m,0), h)$. The argument for move operations in the sixth subset of the definition of $F(K,T,T')$ is analogous to the above.

Finally, let us consider **update** operations. Consider first an edge $[m,n] \in S(F,T)$ such that $|E(m)| = |E(n)| = 1$, implying $n = f'(m,0)$. Since $l(n) \neq l(m)$, clearly $upd(f(m,0), l(n)) \in F$ due to Definition 3. Now consider an edge $[m,n] \in S(F,T)$ such that $|E(m)| = 1$ and $|E(n)| = k > 1$, implying $k-1$ glue operations corresponding to $k-1$ of the k edges in $E(n)$. Now if $[m,n]$ is the edge not corresponding to a glue operation, we can argue as above that $upd(f(m,0), l(n)) \in F$. On the other hand, if $[m,n]$ is an edge corresponding to a glue operation $glu(f(m,0), f(m',0))$, the condition 1(f) in Definition 2 requires $upd(f(m,0), l(n)) \in F$. We have thus shown that all the update operations in $F'(K,T,T')$ for nodes m such that $|E(m)| = 1$ are also present in F. Now consider an edge $[m,n] \in S(F,T)$ such that $|E(m)| = k > 1$ and $|E(n)| = 1$, implying $k-1$ copy operations corresponding to $k-1$ of the k edges in $E(n)$. Now if $[m,n]$ is the edge not corresponding to a copy operation, we can argue as above that $upd(f(m,0), l(n)) \in F$. On the other hand, if $[m,n]$ is an edge corresponding to a copy operation $cpy(f(m,0), h, i)$, $n = f'(m,i)$, implying $upd(f(m,i), l(n)) \in F$ since $l(m) \neq l(n)$ (by Definition 3). It is easy to show that the argument for update operations also holds in the reverse direction; that is, an update operation in F implies a corresponding one in F'. $\square$