On comparing bills of materials: A similarity/distance measure for unordered trees

Carol J. Romanowski and Rakesh Nagi[1]

Abstract— **Many enterprise areas such as marketing, variant design, group technology and cellular manufacturing require their wide variety of products to be organized into families, which are clusters of similar products. In this paper, we propose a similarity metric for finding the distance between existing products based on bills of materials (BOMs), a class of unordered trees. We show that existing editing operations for unordered trees are not consistent for BOMs, and present a similarity metric based on the symmetric difference. We also provide an polynomial time algorithm for finding the *minimum weighted symmetric difference* between a pair of unordered trees. The results of the pairwise comparisons are used as a distance metric for a clustering algorithm that groups the BOM trees into product families.**

Index Terms—**bills of material, similarity measure, symmetric difference, unordered trees.**

I. Introduction

Recent manufacturing paradigms like agile manufacturing and globalization have resulted in product proliferation, and mass customization is the order of the day. Consequently, the number of products and part numbers have increased exponentially. At the same time, product development lead times have to be reduced; therefore, companies are eagerly interested in exploiting similarities among the variants, and benefiting as much as possible from previously done work. The historical approach to classification (or grouping) of individual parts into families is the well-known concept of Group Technology (GT) ([1], [2], [3], [4], [5]). The practical acceptance of GT has remained limited due to the enormous effort involved in developing a "coding system" to summarize key design, manufacturing, and other attributes, and translating the legacy part database into this code. This classification and coding process has largely remained manual, although some efforts towards automation have also been made [6]. Today, Data Mining, a growing field, is providing a credible approach to sifting through terabytes of data records to identify meaningful, machine learning-based patterns and relationships between attributes.

This paper is focused in the area of new tree mining methods that are applicable to industrial product databases. While products from different domains such as mechanical, electrical, electronic, civil/infrastructure differ in their key design and manufacturing attributes, a common data type is the bill of materials (BOM). A BOM (also called a *recipe*, *formulation*, or *specification* in other engineering disciplines) is the hierarchical, structured representation of a product, containing critical information such as components, raw materials, quantities, instructions for manufacture, and consumable items [7]. BOMs capture the make-up, content, and structure of complex products from these engineering domains.

The major purpose for BOMs is to define the recursive parent-child relationships between the end item, its components or subassemblies, and the raw (or purchased) materials they contain. These relationships provide the data needed to efficiently schedule end items for manufacture and ensure sufficient inventory levels to support their production.

BOMs can be depicted as rooted, unordered trees. The end item, or finished product, is the root of the tree; manufactured or assembled components are the nodes; and purchased parts or raw materials are the leaves. Fig. 1 shows an office chair BOM structure as a tree.

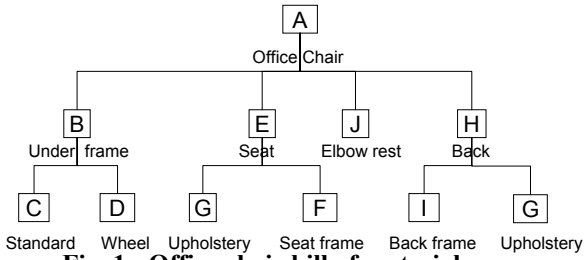


Fig. 1: Office chair bill of materials

A. Types of differences in BOMs

Different engineers may build completely identical end items with very different BOM structures; since there is no common rule or template to follow, the engineer develops the BOM based on her understanding of how the product is manufactured or assembled. Thus, trees representing otherwise identical end items can have very different topologies, from relatively flat trees (not much different than mere parts lists) to highly structured, multi-level trees.

BOM trees may differ in three ways:

1. Structural differences such as the number of intermediate parts, parts at different levels, and parts with different parents.
2. Differences in component labels.
3. Differences in both components and structure.

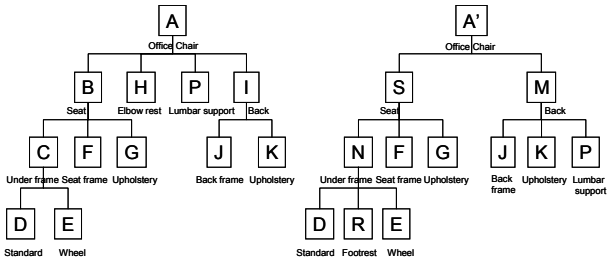
For example, Fig. 2 shows an office chair (A) and a variant

[1] Manuscript received May 30, 2003. This work was supported by the Engineering Research Program of the Office of Basic Energy Sciences at the Department of Energy and the National Science Foundation under career grant DMI-9624309.

Carol J. Romanowski is with the Department of Industrial Engineering, University at Buffalo, Buffalo, NY 14260 USA (e-mail: cfr@buffalo.edu).

Rakesh Nagi is with the Department of Industrial Engineering, University at Buffalo, Buffalo, NY 14260 USA (phone: 716-645-2357 ext. 2103; e-mail: nagi@buffalo.edu).

(A'). Note the lumbar support (P); in Tree 1, on the left, it appears as the child of the end item, Office Chair (A). In Tree 2, on the right, the lumbar support is included as part of the subtree rooted at M, which represents the chair back subassembly. The change in the subassembly label from *I* to *M* reflects the new part number generated by adding the lumbar support to the original subassembly.



Tree 1 **Tree 2**

Fig. 2: Variants of an office chair

Therefore, similar BOMs may have the same components or parts, but have different structure, with some parts appearing at one level in one tree and at another level – or with a different parent – in a second tree. Additionally, BOMs may have similar structure but different components. These situations are common in actual practice.

BOMs are also unordered - meaning that the order of nodes, or components, is not significant. For instance, it does not matter if we say a car has a body, wheels, and transmission or a car has a transmission, body, and wheels.

B. Motivation and outline of the paper

The notion of similarity in BOMs is rooted more in content than in topology; the commonality of content lies in similarities between the component parts in the two BOMs. However, we do want to capture the differences in structure. Quantifying the similarities between BOMs is important in identifying similar end items whose designs can be re-used in new products. Research has been done on similarity measures for ordered trees ([8], [9], [10], [11], [12]), but these methods are not consistent for unordered trees. Unit cost editing operations are typically used to determine distance between unordered trees, but we show in Section II.B that these operations do not give accurate distances for BOMs. In this paper, we propose a metric based on the symmetric difference that calculates accurately the similarity of BOM trees. The similarity measure results are then input to a *k-medoid* algorithm to cluster the similar BOMs.

This paper is organized as follows: in Section II we discuss existing approaches to matching both ordered and unordered trees, and show how those approaches give incorrect distance values for BOM trees. Section III introduces definitions, notations, and presents the minimum weighted symmetric difference metric. In Section IV, we propose an algorithm to find the difference between two trees. Finally, Section V oncludes the paper, discussing a pilot study and recommending further work.

II. APPROACHES TO MATCHING TREES

A. Matching ordered and unordered trees

Ordered tree matching is easier than unordered matching because the order of sibling nodes is fixed. In exact matching, each node maps precisely from one tree to the other, using the same labels. Approximate matching allows inexactness in labeling and topology comparisons between the pattern tree and the data, or target tree. For example, a node labeled "*ide" exactly matches "side" but only partially matches "kids." The distance between "kids" and "side" is the number of editing operations needed to transform "kids" into an exactly matching node.

References [10] and [11] give algorithms for exact matching of ordered trees; [12] and [13] discuss a PAC (probably approximately correct) machine learning approach to learning ordered and unordered tree patterns from queries. Using an approximate matching approach first proposed by [14], Reference [15] develops a system that allows users to build a pattern tree or modify an existing one on screen, then retrieve similar trees from the database. Reference [16] looks for similar consensus (largest approximately common substructures) between ordered trees by using the isolated-subtree distance metric first proposed by [17].

Most tree matching algorithms use a set of editing operations, derived from string comparison research, to transform the trees into isomorphisms. These editing operations, which are constrained to be metrics, include node insertion, node deletion, and node substitutions (essentially, label changes). Reference [18] calculates the distance between two unordered trees as the minimum sum combination of unit cost node deletions, node insertions, and node substitutions needed to transform two trees into isomorphisms. Their algorithm finds this minimum cost by forming a state space, partitioning each tree into two sets of strings ending at a leaf node: a set of unmarked strings to be deleted, and a set of marked strings that are condensed into single nodes. Each state represents a different combination and/or number of strings assigned to each set. The reduced trees resulting from deleting and condensing strings are compared using level-by-level bipartite matching, where nodes with a different number of children or on a different level are considered to be at infinite distance. The minimum cost marking on two trees is found by examining the cost of the bipartite matching plus the editing distance between the deleted strings of both trees.

Reference [8] restricts node operations to deletions only, removing nodes that are not common to both trees. Reference [19] aligns the two trees into a single tree, inserting null nodes where the trees were mismatched. Reference [20] represents the trees as a bipartite graph, and finds the edge cover that corresponds to a minimum cost edit sequence, or script. The authors include an edit operation that allows subtrees to be moved to a different parent with a unit cost, unlike the more expensive string-based edit requiring deletion of all children in the subtree and its root node. Additionally, [20] allows a label change operation to fail the triangle inequality test for a metric.

Computing the editing distance between unordered labeled trees is an NP-Complete problem ([8] provides a proof). Reference [20] shows that the problems of finding the largest common subtree and the minimum edit distance of two rooted, unordered trees are both MAX-SNP hard.

In all these approaches, the focus is on a topological matching based on strings. In BOMs, we are interested primarily in content; topology is a secondary concern, and strings do not adequately represent this domain. In fact, the string editing operations would give a false distance measure for many BOMs. We show two cases where unit cost editing operations do not give the correct cost, or distance.

B. *Incorrectness of string-based unit cost editing operations*

Case 1: Trees with same topology but different content (see Fig. 3: Case illustrating infeasible transformations). In this case, editing operations would replace the labels for nodes V, W, X, and Z even though the nodes are obviously not similar in content.

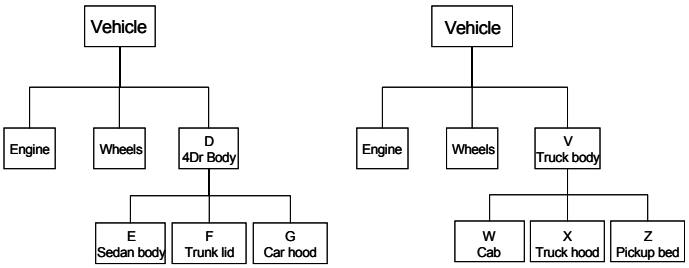


Fig. 3: Case illustrating infeasible transformations

Case 2: Trees with identical subtrees that have different parents (see Fig. 4: Incorrect cost).

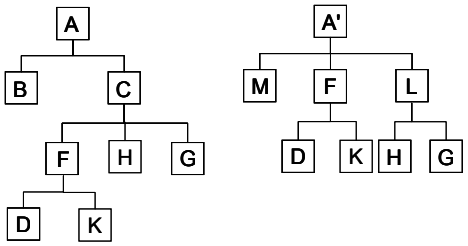


Fig. 4: Case illustrating incorrect cost

Editing operations would delete nodes D, K, and F and insert the same three nodes as children of the root node A. The cost for this set of operations would be 6. However, for BOM trees, a more intuitive way to look at the difference is to disconnect the entire subtree rooted at F and move it to become a child of A – at a cost of 2. This operation is analogous to the *move* operation of [19].

The addition of *move* implies the existence of rules for applying the different operators. For the purposes of determining a minimum cost edit script, or sequence of editing operations, such as in [19], the greater accuracy in distance measurement for certain domains is offset by the greater number of possible edit scripts that must be considered.

The *copy* operation used in [20] would actually give a lower cost for BOMs. Addition of another subtree requires adding the subtree root node and all its children, resulting in a cost of unit*(root+children), not a single unit cost.

C. *Summary of tree editing operations and costs vs BOM needs*

Editing operations and cost functions should, to be accurate, reflect the needs of the domain in which they are applied. Table 1 summarizes tree editing operations, their restrictions, costs, and their applicability to the BOM domain. Note: deletion of an internal node means removing a parent node *i* that is a child of *j*; *i*'s children become the children of *j*. Likewise, inserting an internal node means adding a node *i* as a child of *j*; a subset of *j*'s children become the children of *i*. In contrast, changing a subtree's parent is described in Case 2.

Table 1: Comparison of editing operations and costs

Op no.	Editing operations	Description/ Restrictions	Metric	Cost of operation*	BOM cost
	Deletion	Leaf nodes	Y	Unit [SWZ, CGM]	Unit
		Internal nodes	Y	Unit [SWZ, CGM]	Unit
1		Change subtree parent	Y	Unit*(root+children) [SWZ]	Unit
	Insert	Leaf nodes	Y	Unit [SWZ, JWZ, CGM]	Unit
		Internal nodes	Y	Unit [SWZ, JWZ, CGM]	Unit
2		Change subtree parent	Y	Unit*(root+children) [SWZ, JWZ]	Unit
	Substitute/ update (label change)	No partial matches	Y	Unit [SWZ]	Partial, metric
3		Partial matches allowed	N	Domain dependent [CGM]	NA
4	Move	Change subtree parent	?	Less than copy [CGM]	Unit
5	Copy	Insert a subtree	?	Unit; inverse of glue [CGM]	NA
6	Glue	Delete a subtree	?	Unit; inverse of copy [CGM]	NA

*SWZ = Shasha, Wang, Zhang, Shih 1994; JWZ = Jiang, Wang, Zhang, 1995; CGM = Chewathe, Garcia-Molina, 1997

From the table, we can see that the BOM cost/distance model and allowable operations are distinctly different from previous methods (shaded cells highlight the dissimilarities). Therefore, we need a different means of finding the distance between two BOM trees.

D. *Definitions and notation*

Bill of materials: A structural representation of subassemblies, components, parts, and their relationships that make up a particular end item. Level 0, the highest level of an indented BOM, is occupied by a single entity, the end item, which has no parent. Levels 1-*n* are considered to be below Level 0 and contain subassemblies and purchased items.

End items: Entities that are sold to customers, and therefore appear in Level 0 of the indented BOM. End items contain subassemblies and parts, and in practice may contain other end items. End items may also be purchased and sold to customers without any value-added manufacturing activity. These types of end items have no children (lower level components).

Subassembly: Entities that are generally not sold to customers, and therefore do not appear in Level 0 of the indented BOM. Subassemblies are manufactured items that may contain manufactured or purchased parts or other subassemblies, and therefore do not appear in the lowest level of the BOM.

Purchased parts: Entities that are either raw or purchased materials, and therefore only appear in the BOM tree as leaf nodes. Purchased parts by definition have no children.

Quantity representation: In bills of materials, repeated subassemblies or parts are represented by a *quantity per* value. This value is the number of the part required per *one unit* of the part's parent.

Consider the left BOM tree in Fig. 5. The node values correspond to the quantities of each part per parent. If we were to add another subassembly rooted at *B*, the result would be the right BOM tree. Note that the child quantities do not change, as they reflect only quantity per parent.

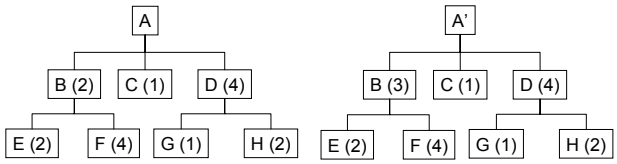


Fig. 5: Quantity representation in BOMs

Matrix representation of BOMs: BOMs can also be represented as square adjacency matrices, where a_{ij} = the quantity of *j* needed to make one unit of *i* if *j* is a child of *i*, and 0 otherwise. For example, Fig. 6 shows the adjacency matrix for the left tree in Fig. 5.

	A	B	C	D	E	F	G	H
A	0	2	1	4	0	0	0	0
B	0	0	0	0	2	4	0	0
C	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	1	2
E	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0

Fig. 6: Adjacency matrix for Tree A

Node: A node is a vertex in the graph that is connected to other nodes or to leaves by directed edges. The root node of a tree is a special node that has no parent, corresponding to an end item. Subassembly nodes have several children; purchased parts nodes have no children. The set of nodes, or vertices, in a tree is represented as *V*.

Node labels and part numbers: Part numbers are alphanumeric strings that uniquely identify the end items, subassemblies, or purchased parts. Each number corresponds to a specific item with specific characteristics. For instance, an oval button would have a different part number than a round button. Some companies use meaningful part numbers that provide information about the part, while others use arbitrary schemes.

If we compared two BOMs using part numbers as labels, the two BOMs would only match where the part numbers were exactly the same. For instance, suppose Part XYZ-10 is a washer with I.D. = 10 mm. Part XYZ-20 is a washer with I.D. = 20 mm. These two washers would not be matched because of the unique part numbers. However, we are interested in finding BOMs of similar – not just exact – content and topology. For this reason, we replace the part numbers with general node labels derived from the part characteristics and types. In the case of these two parts, we would replace the unique part labels with a single label XYZ for the class of washers.

As we noted previously, this part label generalization requires domain knowledge to accomplish effectively, and is currently a primarily manual operation. Additionally, the contribution of each child to its parent is also a matter of domain knowledge. Some parts are more or less critical than others (compare product literature with a circuit board, for example), and so contribute proportionally to the distance between two parent nodes. We assume in this paper that the criticality of parts is known, the label generalization is predetermined, and thus the distances between purchased items (leaf nodes) are available in an offline lookup table (see [6]; also, [23] and [24] present data mining approaches to part number generalization, involving the construction of an industry-specific thesaurus and index based on text mining of part descriptions).

Size: The size of a tree or graph is the total number of edges, and is denoted as $|E|$, where *E* represents the edge set. In a tree, $|E| = |V|-1$.

Degree: The degree of a node is the sum of edges coming into (in-degree) and going out of the node (out-degree). Since every node in a BOM has only one parent, we use degree to refer to out-degree – the number of children of a node.

Isomorphism: Two trees are considered to be isomorphic if they contain the same nodes and have the same structure.

V: the set of vertices (nodes) in a graph.

E: the set of edges (arcs) in a graph.

***G*(*V*,*E*)**: a graph made up of vertices *V* and edges *E*.

$\cup$: the union operator.

$\oplus$: the ring-sum operator (see also Section III).

{}: the empty set.

Parenthesized notation for a tree or directed graph: nested parentheses that denote parent and child nodes. For example: the tree in Fig. 7 would be *A*(*B*(*E*,*F*),*C*,*D*).

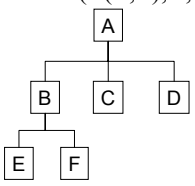


Fig. 7: Sample tree

Preorder traversal: Parent nodes are visited before the child nodes, left to right. In preorder traversal, the nodes of the tree shown in Fig. 7 would be visited in the order *A*-*B*-*E*-*F*-*C*-*D*. This method corresponds to the order shown in parenthesized notation, and is similar to the node order in a depth-first search.

III. General Overview Of Our Approach

Editing operation algorithms focus on topology and transformation of one tree into another to find isomorphisms.

These algorithms were initially an outgrowth of string comparisons, adapted to ordered and unordered tree structures. When comparing BOMs we must consider two important factors: first, BOMs are not a product of nature – they are man-made, and thus subject to inconsistencies in structure. Secondly, because of that man-made inconsistency, using existing editing algorithms may inflate the transformation cost between two otherwise similar BOMs, as we have shown in Section II.B.

To address these problems, we modified a graph difference operation (also called the symmetric difference, or ring-sum difference) to determine the similarity between two BOM trees. The symmetric difference is the set of edges that appear in one graph but not the other. Since edges in BOMs represent parent-child relationships, the symmetric difference between two BOM trees is the dissimilarity in both content and topology.

However, in some cases label differences (representing content dissimilarities) are not completely orthogonal, and partial matching of nodes is needed. Our modification to the "classical" symmetric difference allows these partial matches to be made between nodes in the two trees. In support of these matches, the offline lookup table defines the distances between purchased parts. Distances between intermediate, or parent, nodes are directly derived from these predefined values and "rolled up" through the trees; i.e., the distance between parent nodes i and j is the sum of the distances between the children of i and j, and so on.

Sections A through C discuss the classical definition and application of symmetric difference; Section D introduces the modification that includes partial label matching, and Section E gives an illustration of the approach.

A. Definition of symmetric difference

Consider two BOM graphs, $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$. The difference between the two BOMs is analogous to a symmetric difference ($G_1 \oplus G_2$) of two graphs G_1 and G_2, where

$$G_1 \oplus G_2 = \left(\left(V_1 \cup V_2\right), \left(E_1 \cup E_2\right) - \left(E_1 \cap E_2\right)\right) \quad (1)$$

In other words, the symmetric difference represents the edges in G_2 not found in the graph G_1 and the edges in G_1 not found in G_2.

B. Definition of sparsity value

Consider two rooted, directed trees G_a (V_a, E_a) and $G_b(V_b$, E_b). We construct a square adjacency matrix for each BOM, where a_{ij} = the quantity of j per unit of i if a_j is a child of a_i and 0 otherwise. In the column corresponding to the root node every entry is 0, since the root node is never a child of any other node.

If G_a and G_b are not of the same order, we augment the adjacency matrix of the smaller G with additional columns and rows (whose entries a_{ij} are 0) until the two matrices are the same size. The augmented adjacency matrix A represents the smaller of G_a and G_b. The adjacency matrix B represents the larger of G_a and G_b

We find the symmetric difference matrix, D_{ab}, by taking the absolute difference between the two adjacency matrices.

$$D_{ab} = |A\text{-}B| \quad (2)$$

D_{ab} is a sparse matrix; we then calculate a sparsity value S for this matrix by finding the ratio of the sum of non-zero entries to the total number of entries in the matrix. This sparsity value represents the similarity between the two BOM trees; the smaller the sparsity value, the more similar the tree structures.

$$S = S_{ab} = \frac{G_a \oplus G_b}{\left|V_a \cup V_b\right|^2}, \quad (3)$$

which is equivalent to the normalized sparsity value

$$S = \frac{D_j}{n^2}, \quad (4)$$

where D_j is the sum of non-zero entries in D_{ab} and n^2 is the number of entries in the matrix.

The symmetric difference and the normalized sparsity value are both metrics (see [25] for a proof of symmetric difference, and the Appendix for a proof of sparsity value).

C. Finding the symmetric difference between two trees

Lemma 1: $|A\text{-}B| = A \oplus B$; i.e., the absolute value difference between A and B equals the symmetric difference between A and B.

Proof: Consider two BOM tree adjacency matrices, A and B, each of order n. Then,

$$A \oplus B = (V_a \cup V_b, ((E_a \cup E_b) - (E_a \cap E_b))$$

We calculate the entries for the difference matrix D_{ab} as

$$d_{ij} = |\, a_{ij} - b_{ij}|$$

Four cases can arise from this operation. For the purpose of simplicity, and without loss of generality, we assume the maximum quantity of child node j to make parent node i is 1.

Case 1: An edge exists between vertices i and j in matrix A. An edge also exists between vertices i and j in matrix B. In other words, j is a child of i in both A and B. Therefore, the pair (a_{ij}, b_{ij}) belong to the sets ($E_a \cup E_b$) and ($E_a \cap E_b$), and $d_{ij} = |a_{ij} - b_{ij}| = |1\text{-}1| = 0$.

Case 2: No edge exists between vertices i and j in matrix A. No edge exists between vertices i and j in matrix B. In other words, j is a child of i in neither A nor B.

Therefore, $d_{ij} = |a_{ij} - b_{ij}| = |0\text{-}0| = 0$.

Case 3: An edge exists between vertices i and j in matrix A. No edge exists between vertices i and j in matrix B. In other words, j is a child of i in A but not in B.

Therefore, the pair (a_{ij}, b_{ij}) belong to the set ($E_a \cup E_b$), and $d_{ij} = |a_{ij} - b_{ij}| = |1\text{-}0| = 1$

Case 4: No edge exists between vertices i and j in matrix A. An edge exists between vertices i and j in matrix B. In other words, j is a child of i in B but not in A.

This case is similar to Case 3. Therefore, the pair (a_{ij}, b_{ij}) belong to the set ($E_a \cup E_b$), and $d_{ij} = |a_{ij} - b_{ij}| = |0\text{-}1| = 1$

Thus, the non-zero entries in D_{ab} represent only the edges in A that do not exist in B, and the edges in B that do not exist in A – the symmetric difference between the two graphs. □

Lemma 2: $|A_m\text{-}B_m| \neq |A_n\text{-}B_n|$, where A and B are adjacency matrices and $\{m, n\}$ are different orderings of the vertex sets V_a and V_b. That is, the symmetric difference between two adjacency matrices is dependent on the numbering sequence of the vertex sets V_a and V_b.

Proof: Consider two unordered trees, G_a and G_b, with

vertex sets V_a and V_b, respectively, consisting of the same nodes and internal structure, albeit in different order. Number the vertex sets by any consistent numbering sequence (preorder, post order, or level order). Let $V_a = \{a(b(c, d, e), f(g, h))\}$. Let $V_b = \{a(f(h, g), b(e, c, d))\}$. Suppose we use level order traversal to number the vertex sets (the method of numbering is not significant). Using this numbering sequence, we form adjacency matrices for V_a and V_b. Recall that $a_{ij} = 1$ if an edge is present between vertices i and j, and 0 otherwise. Clearly, $|A - B| = 0$ only if, $\forall\ a_{ij}$ and b_{ij}, $a_{ij} = b_{ij}$.

The vertex ordering for V_a is $\{a, b, f, c, d, e, g, h\}$. The vertex ordering for V_b is $\{a, f, b, h, g, e, c, d\}$. The two adjacency matrices formed by this numbering sequence are shown in Fig. 8.

Even though the two trees are isomorphic, by inspection we can see that the symmetric difference for these adjacency matrices is not 0. In fact, no matter what consistent traversal method we use, and no matter how small or large the isomorphic trees, the symmetric difference will not equal 0 unless the trees' vertices appear in exactly the same order. □

Tree A

	a	b	c	d	e	f	g	h
a	0	1	1	0	0	0	0	0
b	0	0	0	1	1	1	0	0
c	0	0	0	0	0	0	1	1
d	0	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	0

Tree B

	a	f	b	h	g	e	c	d
a	0	1	1	0	0	0	0	0
f	0	0	0	1	1	0	0	0
b	0	0	0	0	0	1	1	1
h	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0	0
c	0	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0	0

Fig. 8: Adjacency matrices for two trees

Corollary 3.1: An ordering of the vertex set V_a or V_b exists that finds the minimum symmetric difference between two graphs G_a and G_b.

Proof: Unlike ordered trees, BOM trees do not place restrictions on the left-to-right ordering of sibling vertices. This property makes finding the similarity between two unordered trees more difficult, since we cannot specify a consistent vertex traversal convention that will guarantee the trees are ordered in the same way.

However, by permuting the vertex set of one graph, swapping rows and columns of the adjacency matrix as in Lemma 1, we can find the ordering sequence that minimizes the symmetric difference – and thus, the sparsity value – between two trees.

Consider two BOM trees, G_a and G_b both of order n. Let A be the adjacency matrix constructed from G_a, using any vertex numbering scheme; likewise, let B be the adjacency matrix constructed from G_b.

We keep A fixed, and generate $j = 1,\ldots, n!$ permutations of the rows and columns of B. We then find the difference matrix $D_j = |A\text{-}B_j|$.

Recall that the sparsity value is $S_j = \frac{D_j}{n^2}$, where D_j is the sum of non-zero entries and n^2 is the number of entries in the matrix.

$$\min G_a \oplus G_b = \min_j S_j = \min_j \left(\frac{D_j}{n^2} \right)$$

It is clear that when D_j is minimum, the quantity S_j is also minimum. Therefore, the permutation vector j corresponding to the minimum D_j represents the minimum symmetric difference, and thus the minimum topological dissimilarity, between G_a and G_b. □

D. Modification of the symmetric difference measure

The nature of BOMs and their domain requires a modification of the symmetric difference measure. This change modifies the edge differences from Cases 3 and 4 in Lemma 1 by allowing partially matching nodes.

Partial matching is accomplished using weights w ($0 \le w \le 1$, where 0 is a perfect match and 1 is no match at all) that represent the design, functional, or manufacturing similarities between two parts or components. While it is certainly possible to generalize labels in the preprocessing phase so there is no similarity whatever between labels, it is not always useful to do so. For instance, we could refer to all engines as "engines"; however, we may want to distinguish between 4 cylinder and 8 cylinder engines. We assume these weights to be given for leaf nodes, which represent purchased items in a BOM. The distances between subtrees whose child nodes are also subtree roots are derived from the summing, or "rolling up" of child subtree weights.

This modification changes the calculation of D_j from the simple "count" of unmatched edges to a sum of weighted edges. Using a weighted bipartite matching algorithm that minimizes the sum of weights of a perfect match, we find a minimum cost solution that pairs single level subtrees i and j from the two BOM trees. D_j, then, is the sum of these paired subtree weights, or distances, w_{ij}.

We can define the minimum weighted symmetric difference between two trees A and B as

$$MWSD(A, B) = \min_j S_j = \min_j \left(\frac{D_j}{n^2} \right),$$

where $D_j = \sum_{i,j} w_{ij} x_{ij}$, which is the objective function of the weighted bipartite matching problem. Recall that the x_{ij} term is 0 if no match exists between subtree i and j, and 1 if the two subtrees are matched.

E. Illustration of the minimum weighted symmetric difference metric (MWSD)

In Table 1 we showed that existing edit operations did not accurately measure the difference between two BOMs. Table 2 lists the various ways BOM trees differ, their analogous edit operation number from Table 1, BOM cost, and a sample calculation of the symmetric difference. The last column refers to figures that accompany examples of the weighted symmetric difference metric application.

Table 2: Weighted symmetric difference calculations

Tree Differences	Analogous Edit Operation	BOM Distance	WSDM Calculation Example: Element in Tree A matrix	Element in Tree B matrix	WSDM cost
Subtree/node parent difference					
Tree A (h child of i)	1, 2, 4	Unit per tree	a_{ih} = 1	b_{ih} = 0	
Tree B (h child of g)			a_{gh} = 0	b_{gh} = 1	**Cost = 2**
Node difference		Unit			
Tree A (h child of i)	1, 2		a_{ih} = 1	b_{ih} = 0	
Tree B (no h)					**Cost = 1**
Repeated existing subtree (same parent)		|x-y|, where			
Tree A (h child of i), qty per = 2	2, 5	x=QtyPer(TreeA)	a_{ih} = 3	b_{ih} = 1	
Tree B (h child of i), qty per = 1		y=QtyPer(TreeB)			**Cost = 2**
Add subtree					
Tree A (h child of i, j and k children of h)	2, 5	Unit per added node	a_{ih} = 1	b_{ih} = 0	
Tree B (no h, j, k)			a_{hj} = 1	b_{hj} = 0	
			a_{hk} = 1	b_{hk} = 0	**Cost = 3**
Label match					
Tree A (h child of i)			a_{ih} = 1	b_{ih} = 0	
Tree B (k child of i)	3	0 <= Match <= 1	a_{ik} = 0	b_{ik} = 1	
			Match (h,k) = .8		
			Cost = Match		**Cost = 0.8**

Figs. 9-13 show the graphical and matrix calculation of these operations. In Fig. 9, the subtree rooted at *C* has a different parent in each tree. The adjacency matrices for these trees are shown in Fig. 10. The shaded entries in these matrices are exactly matching edges that result in zero entry values in the difference matrix. Clearly we can see that the difference matrix will consist of 2 non-zero entries: *A*(2,3) corresponding to the edge between *B* and *C*; and *A'*(4,3) corresponding to the edge between *D'*and *C*. The sparsity value for this difference matrix is (2/64) = 0.03125. In contrast, using unit cost edit operations would yield a sparsity value of (6/64) = .09375.

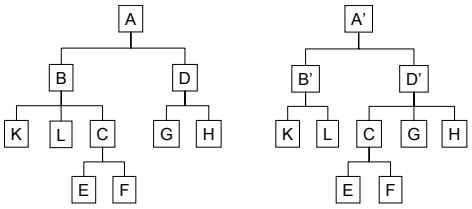


Fig. 9: Illustrating subtree move

	A	B	C	D	E	F	G	H	K	L
A	0	1	0	1	0	0	0	0	0	0
B	0	0	1	0	0	0	0	0	1	1
C	0	0	0	0	1	1	0	0	0	0
D	0	0	0	0	0	0	1	1	0	0
E	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0

	A'	B'	C	D'	E	F	G	H	K	L
A'	0	1	0	1	0	0	0	0	0	0
B'	0	0	0	0	0	0	0	0	1	1
C	0	0	0	0	1	1	0	0	0	0
D'	0	0	1	0	0	0	1	1	0	0
E	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0

Fig. 10: Adjacency matrices for Fig. 9 trees

Fig. 11 shows the operation of adding a subtree that exists in one tree and not the other. Tree *A* contains a subtree rooted at *E* that does not appear in Tree A'. In this case, we need to augment the matrix for Tree *A'* so the two matrices are the same size (see Fig. 12: *du*1 – *du*4 are the dummy entries for matrix *A*'). The shaded cells correspond to zero entries in the difference matrix; the sparsity value for these trees is (4/121) = 0.033.

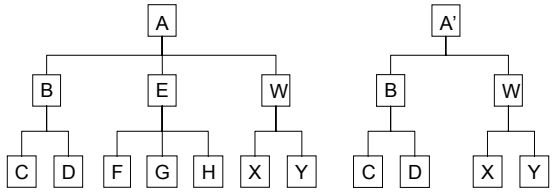


Figure 11: Illustrating added subtree

	A	B	C	D	E	F	G	H	W	X	Y
A	0	1	0	0	1	0	0	0	1	0	0
B	0	0	1	1	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	1	1	1	0	0	0
F	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0
W	0	0	0	0	0	0	0	0	0	1	1
X	0	0	0	0	0	0	0	0	0	0	0
Y	0	0	0	0	0	0	0	0	0	0	0

	A'	B	C	D	du1	du2	du3	du4	W	X	Y
A'	0	1	0	0	0	0	0	0	1	0	0
B	0	0	1	1	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0
du1	0	0	0	0	0	0	0	0	0	0	0
du2	0	0	0	0	0	0	0	0	0	0	0
du3	0	0	0	0	0	0	0	0	0	0	0
du4	0	0	0	0	0	0	0	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0
X	0	0	0	0	0	0	0	0	0	1	1
Y	0	0	0	0	0	0	0	0	0	0	0

)ifference

	Root	B	C	D	E	F	G	H	W	X	Y
Root	0	0	0	0	1	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	1	1	1	0	0	0
F	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0
X	0	0	0	0	0	0	0	0	0	0	0
Y	0	0	0	0	0	0	0	0	0	0	0

Fig. 12: Adjacency and different matrices for Fig. 11 trees

The next case shows the result of partially matching two nodes. In Fig. 13, subtree *B* is made up of items *K* and *P*; subtree *B'* is made up of items *K* and *L*. Topologically, these trees are identical, and a difference matrix will have no non-zero entries. However, in comparing the set of nodes with a lookup table, we can find the partial match for *P* and *L*; suppose these two items match with a value of 0.8; then, the sparsity value for these two trees is (0.8/49) = 0.016.

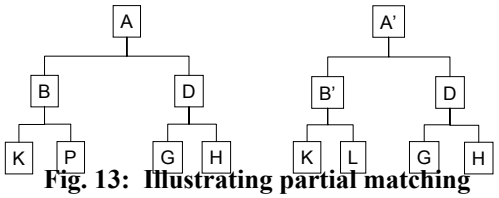


Fig. 13: Illustrating partial matching

IV. AN ALGORITHM TO FIND THE MINIMUM WEIGHTED SYMMETRIC DIFFERENCE

We could find the minimum topological symmetric difference between the adjacency matrices A and B by permuting the matrices and evaluating their absolute value difference, but there are $n!$ possible permutations to compare. Clearly, as n increases, the number of comparisons soon becomes too large for brute force enumeration.

This problem, when disregarding weighted label matches, can be characterized as finding the maximum common *subforest* between two trees. The nature of BOMs, where structural differences mean less than content dissimilarity, allows us to solve the problem in polynomial time. We present a bottom-up approach similar to the maximum common subtree approach by [26] that compares the topology and content of the trees at the subtree level and uses bipartite matching to find the minimum difference between individual subtrees.

Before we describe this method, we must establish some definitions.

Single-level subtree: a parent node and its children; the children of these subtrees may or may not be leaf nodes.

Terminal subtree: a parent node and its children who are *all* leaf nodes.

Consider two trees, T_a and T_b.

We choose a single-level subtree t_a, with root node n and children $C(a_i, \ldots, a_n)$ from T_a and a single-level subtree t_b, with root node m and children $D(b_j, \ldots, b_m)$ from T_b.

The subtrees t_a and t_b are *exactly matching* subtrees if $n = m$ and $C = D$.

In this approach, we form the adjacency matrices using preorder traversal. The matrices built using this method have the root node at the left side of the matrix, and group subtree root nodes with their child nodes, making it simple to find the rows and columns corresponding to terminal subtrees. We first calculate the minimum possible sparsity value, S_l, which is $\frac{|B|-|A|}{n^2}$; this value is 0 if both trees are the same size. We decompose the trees into single-level subtrees, starting with terminal subtrees and moving up the levels of the trees. We then find the distances between subtrees, comparing every node in each subtree in Tree A with every node in each subtree in Tree B. Child nodes that match exactly (same label, same parent) are removed from the subtrees. Then, we perform a weighted matching (function LabelMatch) to determine the minimum distance between labels of the remaining nodes. Label distances between nodes are found in the lookup table, *SubTreeDist*. Previously calculated subtree distances are also stored in *SubTreeDist*.

We next find the minimum distance matching over all subtrees using function TreeMatch. The result of this function is a set of paired subtrees with a distance, or weight, w_{ij}. The distance between the entire trees A and B is the sum of these weights divided by n^2, where n is the number of nodes in the larger tree.

We first present the algorithm and then prove that it finds the minimum weighted symmetric difference between two unordered trees.

A. The decomposition-reduction (DeRe) algorithm:

Input: Unordered BOM trees T_a and T_b, $|T_a| \le |T_b|$
Output: Sparsity metric denoting the minimum normalized weighted symmetric difference between T_a and T_b.

1. Form adjacency matrices A and B using preorder traversal.
2. Calculate the lower bound $S_l = \frac{|B|-|A|}{n^2}$ where n is the number of columns in B
3. If $|B| > |A|$, then augment A with ($|B| - |A|$) rows and columns of zeros at the right end and bottom of the matrix.
4. Calculate $S_{ab} = \frac{D}{n^2}$, the initial sparsity value
   If $S_{ab} == S_l$ then set $S_{min} = S_{ab}$
5. **Find exactly matching subtrees**
   For i = 1 to m, where m is the number of subtrees in B
   Beginning at the right side of matrix B and moving left toward the root,
   Find the first terminal subtree t_{bi} rooted at n
   Search A for an exactly matching terminal subtree t_a with root n
   If t_a and t_{bi} are exactly matching subtrees, then reduce both A and B by removing the rows and columns corresponding to the child nodes of t_a and t_{bi}.
   Next i
   Repeat for single-level subtrees in A and B. If no more exact matches remain, continue.
6. **Find paired subtrees**
   For j=1 to r, where r is the number of remaining subtrees in B,
   For k=1 to s, where s is the number of remaining subtrees in A,
   For each subtree t_{bj} find the distance to t_{ak} using table *SubTreeDist* and function LabelMatch(t_{bj}, t_{ak})
   Find the paired subtrees using function TreeMatch (T_a, T_b) .
7. Calculate $S_{ab} = \frac{D}{n^2}$
8. Return S_{ab}

Function LabelMatch (t_{bj}, t_{ak})
For nodes i to m in subtree t_{ak},
For nodes j to n in subtree t_{bj}
Perform weighted bipartite match to find best matching of subtree nodes
$D_{sub(i,j)} = \Sigma w_{ij}$ from weighted bipartite matching
Store D_{sub} for all subtree pairs in *SubTreeDist*

Function TreeMatch (T_a, T_b)
For trees T_a and T_b with subtree sets {t_a} and {t_b}
Using $D_{sub(a,b)}$ from *SubTreeDist,* perform weighted bipartite match to find best matchings of subtree pairs
$D = \Sigma$ min D_{sub} (t_a, t_b) for all subtree pairs from {t_a}, {t_b}
Return D

B. An illustration of the DeRe algorithm

Consider the trees from Fig. 9, repeated in Fig. 14:

The first step in the algorithm is to form the matrices A and A' using preorder traversal. Because these two trees are of identical size, we do not need to augment either matrix. The lower bound $S_l = 0$.

Next, we calculate the sparsity value S_{ab}. Fig. 15 shows the

matrices for *A*, *A'*, and the difference matrix. S_{ab} for these two trees is 8/100 =.08.

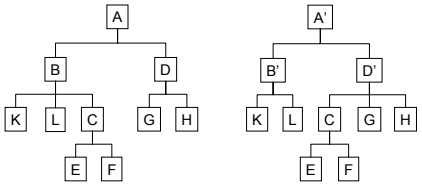


Fig. 14: Illustrating DeRe algorithm

	A	B	K	L	C	E	F	D	G	H
A	0	1	0	0	0	0	0	1	0	0
B	0	0	1	1	1	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	1	1	0	0	0
E	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	1	1
G	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0

	A'	B	K	L	D'	C	E	F	G	H
A'	0	1	0	0	1	0	0	1	0	0
B	0	0	1	1	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0
D'	0	0	0	0	0	1	0	0	0	0
C	0	0	0	0	0	0	1	1	0	0
E	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0

Difference	A	B	K	L	C	E	F	D	G	H
A	0	0	0	0	1	0	0	0	0	0
B	0	0	0	0	1	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	1	0	0	0
E	0	0	0	0	0	0	1	1	0	0
F	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	1	1
G	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0

Fig. 15: Matrices for DeRe example

There are two terminal subtrees in Tree *A*: one rooted at *C* and one rooted at *D*. In Tree *A'* there are also two terminal subtrees: one rooted at *C* and one rooted at *B'*. The two subtrees rooted at *C* are exact matches. Therefore, we can remove their children *E* and *F*. Fig. 16 shows the reduced tree.

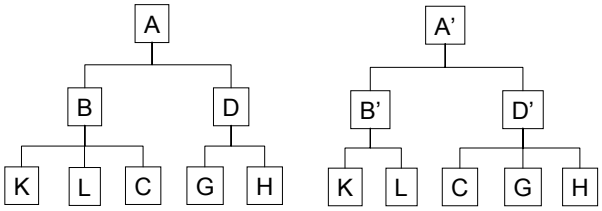


Figure 16: Trees after removing exact matches

We cannot evaluate the distance between the single-level subtrees rooted at A and A' because their children are also subtrees, and have not yet been evaluated.

Next, we look at the four subtrees rooted at *B*, *D*, *B'*, and *D'*. We assume for this example that the distance between differently labeled nodes is 1. Consider the subtree rooted at B. The distance between *B* and *B'* is 1, since *K* and *L* match exactly and *C* has no match in *B'*. The distance between *B* and *D'* is 2, since the *C* nodes match exactly but the remaining nodes do not. Consider the subtree rooted at *D*; the minimum distance between *D* and *B'* is 2. The distance between *D* and *D'* is 1, which represents the parent difference of *C*. We store these subtree pair distances in *SubTreeDist.*

The last remaining subtrees are those rooted at *A* and *A'*. We have already calculated the distances between their children and stored them in *SubTreeDist*, so we use the values to find the best match for the child nodes of *A* and *A'*. The best match for subtree *B* is subtree *B'*, at a distance of 1; the best match for *D* is *D'*, at a distance of 1. so S_{ab} for *A* and *A*' $= D_{sub}(B,B') + D_{sub}(D,D') = 1 + 1 = 2/100 = 0.02$. This result is the same as the result in Case 2 (Section II.B.).

C. Proof of the DeRe algorithm

Theorem 1: The DeRe algorithm finds the minimum weighted symmetric difference between two unordered trees T_a and T_b.

Proof:

Consider two unordered trees T_a and T_b and their respective adjacency matrices *A* and *B*. We decompose the trees into two sets of single-level subtrees $(t_{a1},\ldots,t_{an})$ and $(t_{b1},\ldots,t_{bm})$. Let *C* be the set of child nodes of single-level subtree t_{ai}, found in T_a, and rooted at *p*. Let *D* be the set of child nodes of single-level subtree t_j, found in T_b, and rooted at *q*. We define $\Gamma_{(a,b)}$ as the set of children in all exactly matching single-level subtrees in T_a and T_b. Then, $\{C(t_i, p), D(t_j,q)\} \supset \Gamma_{(a,b)}$ if $p=q$ **and** $(C \cup D) - (C \cap D)) = \{\}$.

Therefore, the elements of $\Gamma_{(a,b)}$ correspond to Case 1 in Lemma 1, and represent zero-valued elements in $|A - B|$.

We remove the elements in $\Gamma_{(a,b)}$ from T_1 and T_2 to form the reduced trees T_{1r} and T_{2r}. Again, we decompose the trees into two sets of single-level subtrees $T_a(t_i,\ldots,t_n)$ and $T_b(t_j,\ldots,t_m)$. The function LabelMatch, as a weighted bipartite matching of individual subtree nodes, provides the minimum distance values between subtrees. These distances (w_{ij}) are used in another weighted bipartite matching (the function TreeMatch) of the single level subtrees $T_a(t_i,\ldots,t_n)$ and $T_b(t_j,\ldots,t_m)$.

By definition, weighted bipartite matching gives a minimum cost matching of the reduced subtrees.

Clearly,

$$MWSD(T_1,T_2) = \frac{\sum_{i,j} w_{ij}x_{ij}}{\left(\max(|T_1|,|T_2|)\right)^2},$$

where the numerator is the resulting minimum cost objective function from the weighted bipartite matching problem.

Therefore, the algorithm finds the minimum weighted symmetric difference between two unordered trees.

V. CONCLUSION

In this paper, we have discussed the motivation for finding similar BOM trees, and shown that calculating the distance between these unordered trees requires a different approach than those prevalent in recent literature. We consider these properties in proposing a minimum weighted symmetric difference metric that accurately computes the distance between two BOM trees, as the domain requires.

We employed the minimum weighted symmetric difference metric between two trees as the distance measure in a *k-medoid* clustering algorithm [22] to group similar bills of materials into product families. 75 bills of materials with known product family classifications were collected from an electronics manufacturer and the DeRe algorithm used to calculate the pairwise distances between them. The resulting distance matrix was input to the clustering algorithm, which correctly placed the BOMs in the appropriate families. This preliminary result shows the metric accurately represents the difference between BOM trees. We intend to expand the study to over 5000 BOMs and a finer cluster granularity, searching for subgroups within the different product families.

The DeRe algorithm as presented could "double count" parts with different parents. For instance, a tree with few intermediate subassemblies may have exactly the same raw material or purchased part content but a flatter structure than another tree (see Figure 17, where Tree 1 is a less structured version of Tree 2). Using editing operations, the distance between Tree 1 and Tree 2 (or Tree 3) is 6 (removing two nodes from subtree *B*, inserting the subtree parent node and the two leaf nodes as siblings of *B*, and changing *B*'s label). Using the weighted symmetric difference measure, and supposing we match subtree *B* with subtree *C*, the distance is 5 (two unmatched nodes from subtree *B*, the subtree rooted at *D* and its two children). In that case, the DeRe algorithm gives a higher weighted symmetric difference than some designers might prefer.

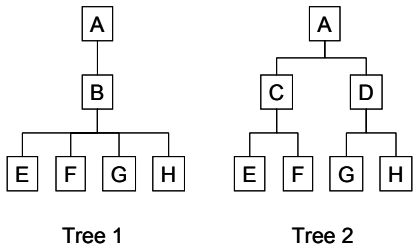


Figure 17: Two similar trees

We are investing future work on refining the DeRe algorithm further by allowing nodes with different parents to be counted only once in the difference calculation.

In manufacturing applications, being able to quantify the differences between BOMs allows engineers to find existing products and reuse proven designs. This research provides an accurate, fast alternative to GT coding and other efforts to find similar parts and subassemblies.

APPENDIX

Lemma A.1: The normalized weighted symmetric difference *S* is a metric if the cost matrix for the weighted bipartite problem is constrained to be a metric.

Proof: A metric must satisfy three properties:

x-x = 0 (positivity)

x-y = y-x (symmetry)

x-$z \le (x$-$y) + (y$-$z)$ (triangle inequality)

The first two properties have trivial proofs. We prove the triangle inequality property as follows:

Consider three single level trees *L*, *M*, and *N*, each with *n* leaves. The given leaf node distances are constrained to be metrics. We assume, without loss of generality, that the weighted bipartite matching has paired leaf nodes with the same index number; i.e., L_1 is paired with M_1, L_2 is paired with M_2, etc. Likewise, L_1 is paired with N_1, M_1 is paired with N_1, and so on.

Since the distance between subtrees is the summed distances between the pairs of leaf nodes, we can say that

$$D_{Tree1,Tree2} = \sum_{i=1}^{n} D\left(Tree1_{Node\ i}, Tree2_{Node\ i}\right).$$

Therefore, we can write

$$D(L,M) \le D(L,N) + D(M,N)$$

as

$$\sum_{i=1}^{n} D(L_i, M_i) \le \sum_{i=1}^{n} D(L_i, N_i) + \sum_{i=1}^{n} D(M_i, N_i).$$

By the triangle inequality, the distance $D(Tree1_{Node\ i}, Tree2_{Node\ i})$ between paired nodes is

$$D(L_1,M_1) \le D(L_1,N_1) + D(M_1,N_1)$$
$$D(L_2,M_2) \le D(L_2,N_2) + D(M_2,N_2)$$
$$D(L_3,M_3) \le D(L_3,N_3) + D(M_3,N_3)$$
$$\dots$$
$$D(L_n,M_n) \le D(L_n,N_n) + D(M_n,N_n).$$

Clearly, the sum of the terms on the left side of these equations is less than or equal to the sum of the terms on the right side of these equations. Therefore, the triangle inequality holds.

However, if the underlying cost matrix is *not* metric, then the triangle inequality does *not* hold. For instance, suppose we have three single level subtrees as in Fig. 18. The distance between these subtrees relies on the relationships between leaf nodes *G*, *H*, and *K*. If the distances between these nodes are not a metric, then the distances *D* between the subtrees will not be a metric either. We see in our distance table that the distance between *G* and *H* is 0.5; between *G* and *K* is 0.1, and between *H* and *K* is 0.3.

By the triangle inequality,

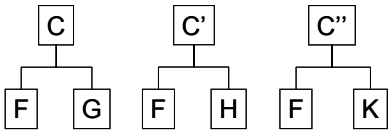


Figure 18: Three single level subtrees

$$D_{CC'} \le D_{CC''} + D_{C'C''}.$$

Since the distances between these subtrees are the distances between nodes G, H, and K, we can rewrite the equation as:

$$D_{GH} \le D_{GK} + D_{HK}.$$

Substituting the values from the lookup table, we have:

$$0.5 \le 0.1 + 0.3$$

which is obviously not true.

Complexity analysis of the DeRe algorithm

There are three basic steps to the DeRe algorithm:

1. Find exactly matching subtrees and reduce the matrices accordingly
2. Find paired subtrees and reduce the matrices accordingly
3. Calculate the weighted symmetric difference of the remaining root node subtree.

To find the overall complexity of the DeRe algorithm, we find first the complexity of the three steps.

Step 1: In this step, we scan the A matrix m times for exactly matching subtrees. Only the first p ($p \geq 1$) scans involve all n nodes; once the children of exactly matching subtree t are removed, the number of nodes in both matrices are reduced to $(n - c)$ nodes, where c ($c \geq 1$) is the number of children in t. We can say, then, that this step is $O(n^2)$ at worst.

Step 2: This step begins with $(n\text{-}C)$ nodes, where C is the total number of children removed during Step 1. We find the minimum distances between nodes, working from known label distances found in the lookup table and derived distances found in the *SubTreeDist* table. The next major procedure is two sequential bipartite matchings, one at the node level and one at the subtree level. These matchings are each $O(n^2)$ operations and dominate the running time for this step.

Step 3: As the tree root node subtrees are the only subtrees remaining, this step is simply the sum of the distances from matched subtrees determined in Step 2.

Therefore, the algorithm takes $O(n^2)$ time.

REFERENCES

[1] M. Henderson and S. Musti, "Automated Group Technology Part Coding from a Three-Dimensional CAD Database," *Journal of Engineering for Industry*, vol. 110, no. 3, pp. 278-287, 1988.

[2] G. Harhalakis, A. Kinsey, and I. Minis, "Automated Group Technology Code Generation Using PDES," in *Proc. 3rd Int. Conf. Computer Integrated Manufacturing*, Rensselaer Polytechnic Institute, Troy NY, 1992.

[3] H. Opitz, *A Classification System to Describe Workpieces* (translated by A. Taylor). New York: Pergamon Press, 1970.

[4] I. Ham, D. Marion, and J. Rubinovich, "Developing a Group Technology Coding and Classification Scheme," *Industrial Engineering*, vol. 18, no. 7, pp. 90-97, 1986.

[5] J. Shah and A. Bhatnagar, "Group Technology Classification from Feature-Based Geometric Models," *Manufacturing Review*, vol. 2, no. 3, pp. 204-213, 1989.

[6] S. Iyer and R. Nagi, "Automated Retrieval and Ranking of Similar Parts in Agile Manufacturing," *IIE Trans. Design and Manufacturing*, special issue on Agile Manufacturing, vol. 29, no. 10, pp. 859-876, 1997.

[7] J. Clement, A. Coldrick, and J. Sari, *Manufacturing Data Structures*. Essex Junction, VT: Oliver Wight Ltd. Publications, Inc., 1992.

[8] P. Kilpelainen and H. Mannila, "Ordered and Unordered Tree Inclusion," *SIAM Journal on Computing*, vol. 24, pp. 340-356, Apr. 1995.

[9] D. Shasha and K. Zhang, "Tree Pattern Matching," in *Pattern Matching Algorithms*, A. Apostolico, Ed. London: Oxford University Press, 1998, pp. 341-369.

[10] K. C. Tai, "The tree-to-tree correction problem." *Journal of the ACM*, vol. 26, no. 3, pp. 422-433, July 1979.

[11] C. M. Hoffmann and M. J. O'Donnell, "Pattern Matching in Trees," *Journal of the ACM* vol. 29, no. 1, pp. 68-95, 1982.

[12] T. R. Amoth, P. Cull, and P. Tadepalli, "Exact Learning of Tree Patterns from Queries and Counterexamples," in *Proc. 12th Annual ACM Conference on Computational Learning Theory (COLT)*, Santa Cruz CA, July 1999, pp. 323-332.

[13] T. R. Amoth, P. Cull, and P. Tadepalli, "Exact Learning of Unordered Tree Patterns from Queries," *Proc. of the 11th Annual ACM Conference on Computational Learning Theory (COLT)*, Madison WI, July 1998, pp. 175-186.

[14] K. Zhang, D. Shasha and J. T. -L. Wang, "Approximate Tree Matching in the Presence of Variable Length Don't Cares," *Journal of Algorithms*, vol. 16, no.1, pp. 33-66, January 1994.

[15] J. T. -L. Wang, K. Zhang, K. Jeong, and D. Shasha, "A System for Approximate Tree Matching." *IEEE Transactions on Knowledge & Data Engineering*, vol. 6, no. 4, pp. 559-571, August 1994.

[16] J. T. -L.Wang and K. Zhang, "Finding similar consensus between trees: an algorithm and a distance hierarchy," *Pattern Recognition*, vol. 34, no. 1, pp. 127-137, Jan. 2001.

[17] E. Tanaka and K. Tanaka, "The tree-to-tree editing problem," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 2, no. 2, pp. 221-240, June 1988.

[18] D. Shasha, J. T.-L. Wang, K. Zhang, and F. Shih, "Exact and Approximate Algorithms for Unordered Tree Matching," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, pp. 668-678, Apr. 1994.

[19] T. Jiang, L. Wang, and K. Zhang, "Alignment of trees - an alternative to tree edit," *Theoretical Computer Science* vol. 143, pp. 137-148, May 1995.

[20] S. Chawathe and H. Garcia-Molina, "Meaningful Change Detection in Structured Data," in *Proc. ACM SIGMOD Int. Conf. Management of Data*, Tucson AZ, May 1997, pp. 26-37.

[21] K. Zhang and T. Jiang, "Some MAX SNP-hard results concerning unordered labeled trees," *Information Processing Letters*, vol 49, pp. 249-254, March 1994.

[22] R. Ng and J. Han, "CLARANS: A method for clustering objects for spatial data mining," *IEEE Trans. Knowledge and Data Engineering*, vol. 14, pp. 1003-1016, Sept. 2002.

[23] C. J. Romanowski and R. Nagi, "A Data Mining And Graph Theoretic Approach To Building Generic Bills Of Materials," in *Proc. 11th Industrial Engineering Research Conference*, Orlando FL, May 2002.

[24] C. J. Romanowski and R. Nagi, "A Data Mining-Based Engineering Design Support System: A Research Agenda," in *Data Mining for Design and Manufacturing: Methods and Applications*, D. Braha, Ed. Dordrecht: Kluwer Academic Publishers, pp. 235-254, 2001.

[25] W. D. Wallis, P. Shoubridge, M. Kraetz, and D. Ray, "Graph distances using graph union," *Pattern Recognition Letters*, vol. 22, pp. 701-704, 2001.
[26] G. Valiente, *Algorithms on Trees and Graphs*. Berlin: Springer-Verlag, 2002.

Carol Romanowski is a doctoral candidate in the Department of Industrial Engineering at the University at Buffalo (SUNY). She also received her B.S. (1996) and M.S. (1999) degrees at the University at Buffalo. She is a 1999 recipient of the U.S. Department of Energy's Integrated Manufacturing Predoctoral Fellowship. Her research interests are in production systems, preventive maintenance, engineering design, and data mining.

Rakesh Nagi is an Associate Professor of Industrial Engineering at the University at Buffalo (SUNY). He received his Ph.D. (1991) and M.S. (1989) degrees in Mechanical Engineering from the University of Maryland at College Park, while he worked at the Institute for Systems Research and INRIA, France, and B.E. (1987) degree in Mechanical Engineering from University of Roorkee (now IIT-R), Roorkee, India. He is a recipient of SME's Milton C. Shaw Outstanding Young Manufacturing Engineer Award (1999), IIE's Outstanding Young Industrial Engineer Award in Academia (1999), and National Science Foundation's CAREER Award (1996). His papers have been published in journals including IIE Transactions, International Journal of Production Research, Journal of Manufacturing Systems, International Journal of Flexible Manufacturing Systems, Journal of Intelligent Manufacturing, Computers in Industry, Computer Integrated Manufacturing Systems, Operations Research, Annals of Operations Research, Computers and Operations Research, and Computers and Industrial Engineering. Dr. Nagi's major research thrust is in the area of production systems. His research interests are in Location theoretic approaches to Facilities Design, Agile Enterprises and Information-Based Manufacturing, and Just-In-Time production of assemblies.