Correlating XML Data Streams Using Tree-Edit Distance Embeddings

Minos Garofalakis
Bell Laboratories
Lucent Technologies
600 Mountain Avenue
Murray Hill, NJ 07974
minos@research.bell-labs.com

Amit Kumar
Bell Laboratories
Lucent Technologies
600 Mountain Avenue
Murray Hill, NJ 07974
amitk@research.bell-labs.com

ABSTRACT

We propose the first known solution to the problem of correlating, in small space, continuous streams of XML data through approximate (structure and content) matching, as defined by a general tree-edit distance metric. The key element of our solution is a novel algorithm for obliviously embedding tree-edit distance metrics into an L_1 vector space while guaranteeing an upper bound of $O(\log^2 n \log^* n)$ on the distance distortion between any data trees with at most n nodes. We demonstrate how our embedding algorithm can be applied in conjunction with known random sketching techniques to: (1) build a compact synopsis of a massive, streaming XML data tree that can be used as a concise surrogate for the full tree in approximate tree-edit distance computations; and, (2) approximate the result of tree-edit-distance similarity joins over continuous XML document streams. To the best of our knowledge, these are the first algorithmic results on low-distortion embeddings for tree-edit distance metrics, and on correlating (e.g., through similarity joins) XML data in the streaming model.

1. INTRODUCTION

The Extensible Markup Language (XML) is rapidly emerging as the new standard for data representation and exchange on the Internet. The simple, self-describing nature of the XML standard promises to enable a broad suite of next-generation Internet applications, ranging from intelligent web searching and querying to electronic commerce. In many respects, XML documents are instances of *semistructured data*: the underlying data model comprises an *ordered, labeled tree* of *element* nodes, where each element can be either an atomic data item or a composite data collection consisting of references (represented as edges) to child elements in the XML tree. Further, *labels* (or, *tags*) stored with XML data elements describe the actual semantics of the data.

The flexibility of the XML data model makes it a very natural and powerful tool for representing data from a wide variety of Internet data sources. Of course, given the typical autonomy of such sources, identical or similar data instances can be represented using different XML-document tree structures. For example, different online news sources may use distinct Document Type Descriptor (DTD) schemas to export their news stories, leading to different node labels and tree structures. Even when the same DTD is used, the resulting XML trees may not have the same structure, due to the presence of optional elements and attributes [16].

Given the presence of such structural differences and inconsistencies, it is obvious that correlating XML data across different sources needs to rely on *approximate* XML-document matching, where the approximation is quantified through an appropriate general *distance metric* between XML data trees. Such a metric for comparing ordered labeled trees has been developed by the combinatorial pattern matching community in the form of *tree-edit distance* [3]. Briefly, the tree-edit distance metric is the natural generalization of edit distance from the string domain; essentially, the tree-edit distance between two tree structures represents the minimum number of basic edit operations (node inserts, deletes, and relabels) needed to transform one tree to the other.

Tree-edit distance is a natural metric for correlating and discovering approximate matches in XML document collections (e.g., through an appropriately defined similarity-join operation). The problem becomes particularly challenging in the context of *streaming* XML data sources; that is, when such correlation queries need to be processed over continuous XML data streams that arrive and need to be processed on a continuous basis, without the benefit of several passes over a static, persistent data image. Algorithms for correlating such XML data streams would need to work under very stringent constraints, typically providing (approximate) results to user queries while: (a) looking at the relevant XML data *only once and in a fixed order* (determined by the stream-arrival pattern); and, (b) using a *small* amount of memory (typically, logarithmic or polylogarithmic in the size of the stream) [1, 2, 10, 13]. Of course, such streaming-XML techniques are more generally applicable in the context of huge, Terabyte XML databases, where performing multiple passes over the data to compute an exact result can be prohibitively expensive. In such scenarios, having *single-pass*, *space-efficient* XML query-processing algorithms that produce good-quality approximate answers offers a very viable

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PODS 2003, June 9-12, 2003, San Diego, CA.
Copyright 2003 ACM 1-58113-670-6/03/06 ...$5.00.

and attractive alternative [10].

Prior Work. Massive, continuous data streams arise naturally in a variety of different application domains, including network monitoring, retail-chain and ATM transaction processing, Web-server record logging, and so on. As a result, we are witnessing a recent surge of interest in data-stream computation, which has led to several (theoretical and practical) studies proposing novel one-pass algorithms with limited memory requirements for different problems; examples include: quantile and order-statistics computation [15, 14]; distinct-element counting [4, 8]; frequent itemset counting [6, 21]; estimating frequency moments, join sizes, and difference norms [1, 2, 11, 17]; and, computing one- or multi-dimensional histograms or Haar wavelet decompositions [13, 22]. All these papers rely on an approximate query-processing model, typically based on an appropriate underlying *synopsis* data structure. The synopses of choice for a number of the above-cited papers are based on the key idea of *pseudo-random sketches* which, essentially, can be thought of as simple, randomized linear projections of the underlying data item(s) (assumed to be points in some numeric vector space).

Recent work on XML-based publish/subscribe systems deals with XML document streams, but only in the context of simple, predicate-based *filtering* of individual documents [5]; clearly, the problem of efficiently correlating XML documents across one or more input streams gives rise to a drastically different set of issues. Guha et al. [16] discuss several different algorithms for performing tree-edit distance joins over XML databases. Their work introduce easier-to-compute bounds on the tree-edit distance metric and other heuristics that can significantly reduce the computational cost incurred due to all-pairs tree-edit distance computations. However, Guha et al. focus solely on *exact* join computation and their algorithms require *multiple passes* over the data; this obviously renders them inapplicable in a data-stream setting.

Our Contributions. All earlier work on correlating continuous data streams (through, e.g., join or norm computations) in small space has relied on the assumption of flat, relational data items over some appropriate *numeric vector space*; this is certainly the case with the sketch-based synopsis mechanism (discussed above), which has been the algorithmic tool of choice for most of these earlier research efforts. Unfortunately, this limitation renders earlier streaming results useless for directly dealing with streams of *structured objects* defined over a complex *metric space*, such as XML-document streams with a tree-edit distance metric. As a more concrete example, consider the problem of estimating the result size of a *similarity-join* operation that joins the trees in two input XML-document streams whose tree-edit distance is below some user/application-defined threshold. A space-efficient, one-pass algorithm for approximating such XML similarity joins would provide an invaluable tool, for instance, during data integration when we are trying to estimate the "degree of content similarity" between two distinct XML data sources as their document contents are streaming in.

In this paper, we propose the *first* known solution to the problem of approximating (in small space) the result of correlation queries based on tree-edit distance (like the similarity join described above) over continuous XML data streams. The centerpiece of our solution is a novel algorithm for effectively (i.e., "obliviously" [18]) *embedding* streaming XML and the tree-edit distance metric into a numeric vector space equipped with the standard L_1 distance norm, while guaranteeing a *worst-case* upper bound of $O(\log^2 n \log^* n)$ on the distance distortion between any data trees with at most n nodes.[1] Our embedding is completely deterministic and relies on parsing an XML tree into a hierarchy of special subtrees. Our parsing makes use of a deterministic coin-tossing process recently introduced by Cormode and Muthukrishnan [9] for embedding a variant of the string-edit distance into L_1; however, since we are dealing with general trees rather than flat strings, our embedding algorithm and its analysis are significantly more complex, and result in different bounds on the distance distortion.

We also demonstrate how our vector-space embedding construction can be combined with earlier sketching techniques [1, 10, 17] to obtain novel algorithms for: (1) constructing a small sketch synopsis of a massive, streaming XML data tree that can be used as a concise surrogate for the full tree in tree-edit distance computations; and, (2) estimating the result size of a tree-edit-distance similarity join over two streams of XML documents. To the best of our knowledge, our are the *first* algorithmic results on oblivious tree-edit distance embeddings, and on effectively correlating continuous, massive streams of XML data. We believe that our embedding algorithm also has other important applications; as an example, compared to an expensive, exact tree-edit distance computation algorithm that can require up to $O(n^4)$ time [3], our embedding can be used to provide an *approximate* distance (to within a $O(\log^2 n \log^* n)$ factor) in *near-linear*, i.e., $O(n \log^* n)$ time.

2. PRELIMINARIES

XML Data Model and Tree-Edit Distance. An XML document is essentially an *ordered, labeled tree* T, where each node in T represents an XML element and is characterized by a *label* taken from a fixed alphabet of string literals σ. Node labels capture the semantics of XML elements, and edges in T capture element nesting in the XML data. Without loss of generality, we assume that the alphabet σ captures all node labels, literals, and atomic values that can appear in an XML tree (e.g., based on the underlying DTD(s)); we also focus on the ordered, labeled tree structure of the XML data and ignore the raw-character data content inside nodes with string labels (`PCDATA`, `CDATA`, etc.). We use $|T|$ and $|\sigma|$ to denote the number of nodes in T and the number of symbols in σ, respectively.

Given two XML document trees T_1 and T_2, the *tree-edit distance* between T_1 and T_2 (denoted by $d(T_1, T_2)$) is defined as the minimum number of tree-edit operations to transform one tree into another. The standard set of tree-edit operations [3] includes: (1) *relabeling* (i.e., changing the label) of a tree node v; (2) *deleting* a tree node v (and moving all of v's children under its parent); and, (3) *inserting* a new node v under a node w and moving a *contiguous subsequence* of w's children (and their descendants) under the new node v. (Note that the node-insertion operation is essentially the

[1] All log's in this paper denote base-2 logarithms; $\log^* n$ denotes the number of log applications required to reduce n to a quantity that is ≤ 1, and is a *very slowly* increasing function of n.

complement of node deletion.) An example XML tree and tree-edit operation are depicted in Figure 1. In this paper, we consider a variant of the tree-edit distance metric, termed *tree-edit distance with subtree moves*, that, in addition to the above three standard edit operations, allows a subtree to be moved under a new node in the tree in one step. We believe that subtree moves make sense as a primitive edit operation in the context of XML data – identical substructures can appear in different locations (for example, due to a slight variation of the DTD), and rearranging such substructures should probably be considered as basic an operation as node insertion or deletion. In the remainder of this paper, the term "tree-edit distance" assumes the four primitive edit operations described above, namely node relabelings, deletions, insertions, and subtree moves.

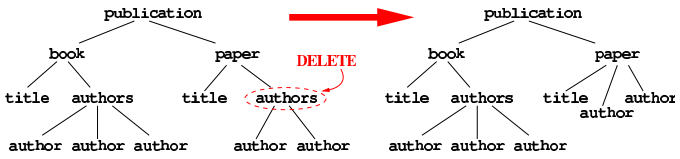

Figure 1: Example XML Tree and Tree-Edit Operation.

Data Streams and Basic Sketching. In a data-streaming environment, data-processing algorithms are allowed to see the incoming data records (e.g., relational tuples or XML documents) *only once* as they are streaming in from (possibly) different data sources [1, 2, 10]. Backtracking over the stream and explicit access to past data records is impossible. The data-processing algorithm is also allowed a small amount of memory, typically *logarithmic or polylogarithmic* in the data-stream size, in order to maintain concise *synopsis* data structures for the input stream(s). In addition to their small space requirement, these synopses should also be easily computable in a single pass over the data and with small per-record processing time. At any point in time, the algorithm can combine the maintained collection of synopses to produce an approximate result.

We focus on one particular type of stream synopses, namely *pseudo-random sketches*; sketches have provided effective solutions for several streaming problems, including join and multi-join processing [1, 2, 10], norm computation [11, 17], distinct-element counting [8], and histogram or Haar-wavelet construction [13, 22]. We describe the basics of sketching using a simple binary-join cardinality estimation query [1]. More specifically, assume that we want to estimate $Q = \texttt{COUNT}(R_1 \bowtie_A R_2)$, i.e., the cardinality of the join of two streaming relations R_1 and R_2 over an attribute (or, set of attributes) A, whose values we assume (without loss of generality) to range over $\{1, \ldots, N\}$. Letting $f_k(i)$ $(k = 1, 2;$ $i = 1, \ldots, N)$ denote the frequency of the i^{th} value in R_k, is is easy to see that $Q = \sum_{i=1}^{N} f_1(i) f_2(i)$; thus, Q can be computed exactly in $O(N)$ space. In their seminal work, Alon et al. [1, 2] show how to build a randomized estimate for Q in small, i.e., $O(\log N)$ space. Briefly, the key idea is to build a *sketch* (essentially, a randomized linear projection) of the distribution vector for each input stream. Such a sketch for R_k is simply defined as $X_k = \sum_{i=1}^{N} f_k(i)\xi_i$ $(k = 1, 2)$, where $\{\xi_i : i = 1, \ldots, N\}$ is a family of *uniform, four-wise independent* $\{-1, +1\}$ random variables. The key here is that the ξ random values can be constructed on-line using a seed of size only $O(\log N)$, and the maintenance of X_k over the R_k stream is extremely simple: start with $X_k = 0$ and just add ξ_i to X_k whenever the i^{th} value of A is observed in the stream. It is then easy to show that the product of the sketches $X_1 \cdot X_2$ is an unbiased estimate for Q, i.e., $E[X_1 \cdot X_2] = Q$. To boost the accuracy and confidence of the estimate, several independent copies of the above-described atomic sketching estimate can be maintained and combined using averaging and median-selection operations [1, 2].

3. OUR APPROACH: AN OVERVIEW

The key element of our methodology for correlating continuous XML data streams is a novel algorithm that *embeds* ordered, labeled trees and the tree-edit distance metric as points in a (numeric) multi-dimensional vector space equipped with the standard L_1 vector distance, while guaranteeing a small distortion of the distance metric. In other words, our techniques rely on mapping each XML tree T to a numeric vector $V(T)$ such that the tree-edit distances between the original trees are well-approximated by the L_1 vector distances of the tree images under the mapping; that is, for any two XML trees S and T, the L_1 distance $\|V(S) - V(T)\|_1 = \sum_j |V(S)[j] - V(T)[j]|$ gives a good approximation of the tree-edit distance $d(S, T)$.

Besides guaranteeing a small bound on the distance distortion, to be applicable in a data-stream setting, such an embedding algorithm needs to satisfy two additional requirements: (1) the embedding should require small space and time per data tree; and, (2) the embedding should be *oblivious*, that is the vector image $V(T)$ of a tree T cannot depend on other trees in the input stream(s). Our embedding algorithm satisfies all these requirements.

There is an extensive literature on low-distortion embeddings of metric spaces into normed vector spaces; for an excellent survey of the results in this area, please see the recent paper by Indyk [18]. A key result in this area is Bourgain's lemma proving that an arbitrary finite metric space is embeddable in an L_2 vector space with logarithmic distortion; unfortunately, Bourgain's technique is neither small space nor oblivious (i.e., it requires knowledge of the complete metric space), so there is no obvious way to apply it in a data-stream setting [18]. To the best of our knowledge, our algorithm gives the *first* oblivious, small space/time vector-space embedding for a complex tree-edit distance metric.

Given our algorithm for approximately embedding streaming XML trees and tree-edit distance in an L_1 vector space, known streaming techniques (like the sketching method discussed in Section 2) now become relevant. In this paper, we focus on two important applications of our results in the context of streaming XML, and propose novel algorithms for: (1) building a small sketch synopsis of a massive, streaming XML data tree; and, (2) approximating the size of a similarity join over XML streams. Once again, these are the first results on correlating (in small space) massive XML data streams based on the tree-edit distance metric.

4. THE EMBEDDING ALGORITHM

In this section, we describe our embedding algorithm for the tree-edit distance metric in detail and prove its small-

time and distance-distortion guarantees. We start by introducing some necessary notational conventions.

Consider an ordered, labeled tree T over alphabet σ, and let $n = |T|$. Also, let v be a node in T, and let $\mathbf{s}$ denote a *contiguous subsequence* of children of node v in T. If the nodes in $\mathbf{s}$ are all leaves, then we refer to $\mathbf{s}$ as a *contiguous leaf-child subsequence* of v. (A leaf child of v that is not adjacent to any other leaf child of v is called a *lone leaf child* of v.) We use $T[v, \mathbf{s}]$ to denote the subtree of T obtained as the union of all subtrees rooted at nodes in $\mathbf{s}$ and node v itself, retaining all node labels. We also use the notation $T'[v, \mathbf{s}]$ to denote exactly the same subtree as $T[v, \mathbf{s}]$, except that we do not associate any label with the root node v of the subtree. We define a *valid subtree* of T as any subtree of the form $T[v, \mathbf{s}]$, $T'[v, \mathbf{s}]$, or a path of degree-2 nodes (i.e., a *chain*) in T. Finally, we assume a hash function $h()$ that maps the set of all valid subtrees of T to new labels in a one-to-one fashion with high probability; such a function can be computed in small space/time, for example, using a simple modification of the Karp-Rabin string fingerprinting algorithm [19].

At a high level, our algorithm produces a *hierarchical parsing* of T into a multiset $\mathcal{T}(T)$ of special valid subtrees. The vector image $V(T)$ of T is essentially the "characteristic vector" for the multiset $\mathcal{T}(T)$ (over the space of all possible valid subtrees). Even though the dimensionality of $V(T)$ is, in general, exponential in n, our construction will guarantee that $V(T)$ is also very sparse: the total number of non-zero components in $V(T)$ is only $O(n)$.

Thus, the technical crux lies in the details of our hierarchical parsing process for T that produces the valid-subtree multiset $\mathcal{T}(T)$. We make use of a string-processing subroutine presented by Cormode and Muthukrishnan [9] that uses deterministic coin-tossing to find *landmarks* in an input string S, which are then used to split S into groups of 2 or 3 consecutive characters. A key property of their landmark-based grouping technique (termed `CM-Group` in this paper) is that, given a string of length k, the choice of a specific character x as a landmark (and, consequently, the chosen group of characters that contains x) depends only on the characters lying in a radius of $\log^* k + 5$ to the left and right of x [9]. Thus, a string-edit operation occurring outside this local neighborhood of a character x is guaranteed not to affect the group formed containing x. As we will see, this property is crucial in proving the distance-distortion bounds for our embedding algorithm.

Our algorithm constructs a hierarchical parsing of T in several phases. In phase i, the algorithm builds an ordered, labeled tree T^i that is obtained from tree of the previous phase T^{i-1} by contracting certain edges. (The initial tree T^0 is exactly the original input tree T.) Thus, each node $v \in T^i$ corresponds to a connected subtree of T — in fact, our algorithm guarantees that this subtree will be a *valid subtree* of T. Let $v(T)$ denote the valid subtree of T corresponding to node $v \in T^i$; the node label for v is determined by the hash-function value $h(v(T))$, where $h()$ is the valid-subtree naming function defined above.

The pseudo-code description of our embedding algorithm (termed TreeEmbed) is depicted in Figure 2. As described above, our algorithm builds a hierarchical parsing structure (i.e., a hierarchy of contracted trees T^i) over the input tree T, until the tree is contracted to a single node ($|T^i| = 1$). The multi-set $\mathcal{T}(T)$ of valid subtrees produced by our parsing for T contains all valid subtrees corresponding to all nodes of the final hierarchical parsing structure tagged with a phase label to distinguish between subtrees in different phases; that is, $\mathcal{T}(T)$ comprises all $< v(T^i), i >$ for all nodes $v \in T^i$ over all phases i (Step 18). Finally, we define the L_1 vector image $V(T)$ of T to be the "characteristic vector" of the multi-set $\mathcal{T}(T)$; in other words,

$$V(T)[< t, i >] \;:=\; \text{number of times the } < t, i > \text{ subtree-phase combination appears in } \mathcal{T}(T).$$

(We use the notation $V_i(T)$ to denote the restriction of $V(T)$ to only subtrees occurring at phase i.) A small example execution of our embedding algorithm is depicted pictorially in Figure 3.

The L_1 distance between the vector images of two trees S and T is defined in the standard manner, i.e., $\|V(T) - V(S)\|_1 = \sum_{x \in \mathcal{T}(T) \cup \mathcal{T}(S)} |V(T)[x] - V(S)[x]|$. In the remainder of this section, we prove our main theorem on the near-linear time complexity of our L_1 embedding algorithm and the logarithmic distortion bounds that our embedding guarantees for the tree-edit distance metric.

procedure TreeEmbed(T)
Input: Ordered, labeled tree T.
Output: Vector embedding $V(T)$ of T.
begin
1. $i := 0$; $T^0 := T$
2. **while** ($|T^i| > 1$) **do**
3. **for each** (maximal chain of degree-2 nodes in T^i) **do**
4. Use `CM-Group` to divide the chain into groups of 2 or 3 nodes
5. Contract each node group to form a new node of T^{i+1}
6. **endfor**
7. **for each** (node $v \in T^i$ with ≥ 2 children) **do**
8. **for each** (maximal contiguous leaf-child subsequence $\mathbf{s}$ of v) **do**
9. **if** ($|\mathbf{s}| \geq 2$) **then**
10. Consider $\mathbf{s}$ as a string and run `CM-Group` to divide $\mathbf{s}$ into groups of 2 or 3 nodes
11. Contract each node group to form a new node of T^{i+1}
12. **else if** ($|\mathbf{s}| = 1$ **and** $\mathbf{s} = \{w\}$ is the leftmost such leaf subsequence under v) **then**
13. Merge this leaf-child w into v to form a new node of T^{i+1}
14. **endfor**
15. **endfor**
16. $i := i + 1$
17. **endwhile**
18. $\mathcal{T}(T) := \{< v(T^i), i >: v \in T^i$ for all phases $i\}$
19. $V(T) :=$ "characteristic vector" of $\mathcal{T}(T)$ (see definition in text)
end

Figure 2: Our Tree-Embedding Algorithm.

Theorem 4.1. *The* TreeEmbed *algorithm constructs the vector image $V(T)$ of an input tree T in time $O(|T| \log^* |T|)$; further, the vector $V(T)$ contains at most $O(|T|)$ non-zero components. Finally, given two trees S and T with $n =$*

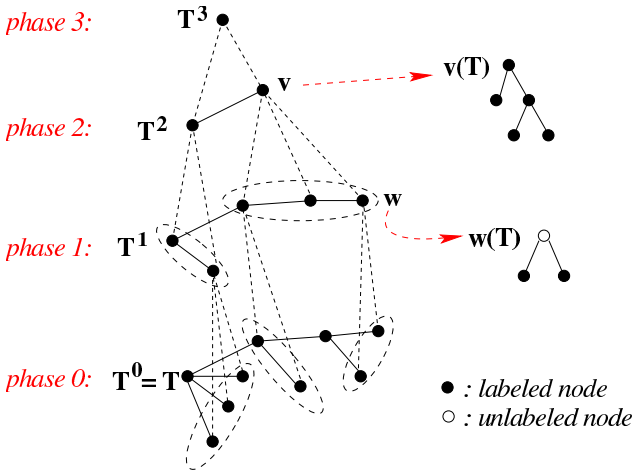

Figure 3: Example Hierarchical Tree Parsing.

$\max\{|S|, |T|\}$, *we have*

$$d(S,T) \leq 5 \cdot \|V(T) - V(S)\|_1 = O(\log^2 n \log^* n) \cdot d(S,T).$$

∎

We first prove the following lemma, which bounds the number of parsing phases; the key here is to show that the number of tree nodes goes down by a constant factor during each contraction phase. The detailed proof can be found in Appendix A.

LEMMA 4.2. *The number of phases for our* TREEEMBED *algorithm on an input tree T is $O(\log |T|)$.* ∎

Lemma 4.2 immediately implies that the total number of nodes in the entire hierarchical parsing structure for T is only $O(|T|)$. Thus, the vector image $V(T)$ built by our algorithm is a *very sparse* vector. To see this, note that the number of all possible ordered, labeled trees of size at most n that can be built using the label alphabet σ is $O((4|\sigma|)^n)$ (see, e.g., [20]); thus, by Lemma 4.2, the dimensionality needed for our vector image $V()$ to capture input trees of size n is $O((4|\sigma|)^n \log n)$. However, *for a given tree T*, only $O(|T|)$ of these dimensions can contain non-zero counts. Lemma 4.2, in conjunction with the fact that the `CM-Group` procedure runs in time $O(k \log^* k)$ for a string of size k [9], also implies that our TREEEMBED algorithm runs in $O(|T| \log^* |T|)$ time on input T.[2] The following two subsections establish the distance-distortion bounds stated in Theorem 4.1.

4.1 Upper Bound Proof

Suppose we are given a tree T with n nodes and let Δ denote the quantity $\log^* n + 5$. Consider a (valid) subtree of T the form $T'[v, \mathbf{s}]$ for some contiguous set of children $\mathbf{s}$ of v (recall that the root of $T'[v, \mathbf{s}]$ has no label). Let us delete $T'[v, \mathbf{s}]$ from T. Let the resulting tree be T_2, and let T_1 denote $T'[v, \mathbf{s}]$. Thus, we have broken T into T_1 and T_2.

We now compare the following two vectors. The first vector $V(T)$ is obtained by applying our parsing procedure to T. For the second vector, we apply our procedure to each of the trees T_1 and T_2 separately and then add the corresponding vectors $V(T_1)$ and $V(T_2)$ component-wise – call this vector $V(T_1 + T_2)$. Our goal is to prove the following theorem.

THEOREM 4.3. *The L_1 distance between $V(T)$ and $V(T_1 + T_2)$ is at most $O(\log^2 n \log^* n)$.* ∎

Let us first see how this result implies the upper bound stated in Theorem 4.1.

Proof of the Upper Bound in Theorem 4.1: It is sufficient to consider the case when the tree-edit distance between S and T is 1. First, assume that T is obtained from S by deleting a leaf node v. Let the parent of v be w. Define $\mathbf{s} = \{v\}$. Delete $S'[w, \mathbf{s}]$ from S. This splits S into T and $S'[w, \mathbf{s}]$ – call this S_1. Theorem 4.3 implies that $\|V(S) - (V(T) + V(S_1))\|_1 \leq O(\log^2 n \log^* n)$. But, it is easy to see that the vector $V(S_1)$ has only two non-zero components, both equal to 1. Thus, $\|(V(T) + V(S_1)) - V(T)\|_1 \leq 2$. By the triangle inequality, $\|V(S) - V(T)\|_1$ is $O(\log^2 n \log^* n)$.

Since insertion of a leaf node is the inverse of a leaf-node deletion, the same holds for this case as well. Let w be a node in S and $\mathbf{s}$ be a contiguous set of children of w. Suppose T is obtained from S by moving the subtree $S'[v, \mathbf{s}]$, i.e., delete this subtree and make it a child of another node x in S.[3]

Let S_1 denote $S'[v, \mathbf{s}]$. Let S_2 denote the tree obtained by deleting S_1 from S. The theorem above implies that $\|V(S) - (V(S_1 + S_2))\|_1$ is $O(\log^2 n \log^* n)$. But we can picture $S_1 \cup S_2$ also as the forest obtained by deleting S_1 from T. Thus, $\|V(T) - V(S_1 + S_2)\|_1$ is also $O(\log^2 n \log^* n)$. Triangle inequality now implies the result.

Finally, suppose we delete a node v from S. Let the parent of v be w. All children of v now become children of w. We can think of this process as follows. Let $\mathbf{s}$ be the children of v. First we move $S'[v, \mathbf{s}]$ and make it a child of w. Now, v is a leaf node, so we are just deleting a leaf node now. Thus, the result for this case follows from the arguments above for deleting a leaf node and moving a subtree. ∎

Thus, it suffices to prove Theorem 4.3. Our proof proceeds along the following lines. We define an *influence region* for each tree T^i in our hierarchical parsing. Initially (i.e., phase 0), this region is just the node v at which we deleted the T_1 subtree. But, this region grows as we proceed to next phases. We then argue that if we ignore this influence region and the corresponding region in $T_1 + T_2$, then the two trees look very similar (in any phase). Thus, if we can bound the rate at which this region grows we can also bound the L_1 distance between the two vectors. The reason why this

[2] An immediate implication of these results is that we can use our embedding algorithm to compute the approximate (to within a $O(\log^2 n \log^* n)$ factor) tree-edit distance between T and S in $O(n \log^* n)$ (i.e., *near-linear*) time. The time complexity of exact tree-edit distance computation (without subtree moves) is significantly higher: $O(|T||S|d_T d_S)$, where d_T (d_S) is the depth of T (S) [3].

[3] This is a slightly "generalized" subtree move, since it allows for a contiguous (sub)sequence of sibling subtrees to be moved in one step. However, it is easy to see that it can be simulated with only three simpler edit operations, namely, a node insertion, a single-subtree move, and a node deletion. Thus, our results trivially carry over to the case of "single-subtree move" edit operations.

approach should work is the following : when we change T at some point, nodes far away from it remain unaffected in the following sense – the set of nodes in which a particular node in phase i will be grouped when parsing the tree T^i remains unchanged. This is a direct consequence of the properties of the `CM-Group` procedure used for grouping nodes in each phase.

We now proceed with the proof. Define $(T_1 + T_2)^i$ as the tree corresponding to $(T_1 + T_2)$ at the beginning of phase i. We say that a node $x \in T^{i+1}$ *contains* a node $x' \in T^i$ if the set of nodes in T^i which were merged to form x contains x'. Any node w in T^i corresponds to a subgraph $w(T)$ in T. It is clear that $w(T)$ is always a connected subgraph of T. It is also clear that if w and w' are two different nodes in T^i, then $w(T)$ and $w'(T)$ are disjoint.

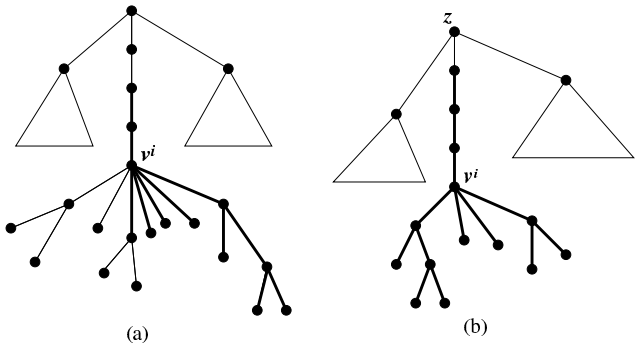

Figure 4: (a) The subtree induced by the bold edges corresponds to the nodes in M^i. (b) Node z becomes the center of N^i.

For each tree T^i, we shall *mark* certain nodes. This intuitively corresponds to the influence region we mentioned above. Let M^i be the set of marked nodes in T^i (see Figure 4(a) for an example). M^i has the following structure: There is a *central* node v^i. M^i will always contain the node v, i.e., the node in T^i which contains v is in M^i. M^i may contain some ancestors A^i of v^i — but all these nodes (except perhaps v^i) must be of degree 2 only and should form a connected path. M^i may also contain some descendants of v^i.

Certain nodes of T^i will be denoted as *corner nodes* – intuitively, these are nodes whose parsing will be affected when they are shrunk down to a leaf node.

The key idea here is that M^i captures the set of those nodes in T^i which have been *influenced* by the change we made at node v. Now, in the next phase, the changes in M^i will affect some more nodes. Thus, we now try to determine which nodes M^i can affect, i.e., if the change at v has influenced all nodes in M^i in phase i, which are the nodes in T^i whose parsing can change as a result of this. In order to capture this, we define another set of nodes N^i in T^i – these will intuitively correspond to the set of nodes whose parsing can be affected by the changes in M^i.

First add all nodes in M^i to N^i. Let w be the highest node in M^i (so it is an ancestor of the center v^i). If M^i contains all descendants of w, then add the parent of node w, call it z, to N^i (see Figure 4(b)). Otherwise, if some descendants of w are not in M^i, then define z to be same as v^i. (The idea here is that the grouping of w's parent in the next phase can only change if the entire subtree under w has changed, e.g., the subtree has just been moved there; otherwise, w's parent cannot be grouped in the next phase in any case.) Node z will be the center of the "extended" influence region N^i. We also add the following nodes to N^i (see Figure 5 for examples) :

(i) Suppose u is a leaf child of (the center) z and there is another child u' of z such that the following conditions are satisfied : $u' \in M^i$ or u' is a corner leaf node, the set of nodes $\mathbf{s}(u, u')$ between u and u' are leaves, and $|\mathbf{s}(u, u')| \leq \Delta$. Then, add u to N^i. (In particular, note that any leaf child of z which is a corner node gets added to N^i.)

(ii) Let u be the leftmost lone leaf child of z which is not already in M^i (if such a child exists). Then, add u to N^i.

(iii) Let w be the highest node in M^i (so it is an ancestor of the center v^i). Let u be an ancestor of w. Suppose it is the case that all nodes between u and w, except perhaps w, have degree 2, and the length of the path joining u and w is at most Δ. Then, add u to N^i.

(iv) Suppose there is a child u of z such that all descendants of siblings of u (other than u itself) are already in M^i. Let u' be the lowest descendant of u which is in M^i. If u'' is any descendant of u' such that the path joining them contains degree-2 nodes only (including the endpoints), and has length at most Δ, then we add u'' to N^i.

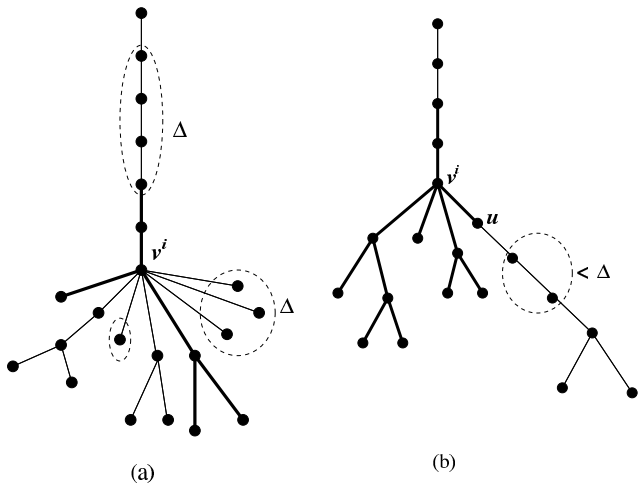

Figure 5: (a) The nodes in dotted circles get added to N^i due to rules (i), (ii), and (iii). (b) The nodes in dotted circle get added to N^i due to rule (iv) – note that all descendants of v^i which are not descendants of u are in M^i.

Let us briefly describe why we need these four rules. We basically want to include all those nodes in N^i whose parsing can be affected if we delete or modify the nodes in M^i. The

first three rules, in conjunction with the properties of our parsing, are easily seen to capture this fact. The last rule is a little more subtle. Suppose u is a child of z. Further, assume all descendants of z except perhaps those of u are in M^i. Remember that all nodes in M^i have been modified, so they may not have been present at all in the original tree. But, if we just *ignore* M^i, then z becomes a node of degree 2 only, which means that it would affect how u and its degree-2 descendants are parsed . This fact is captured by rule (iv).

We now consider the case of *corner nodes*. Suppose z has at least two children. When we parse T^i, we may merge a leaf child u of z into z. If so, then we mark the two immediate siblings of u as corner nodes (see Figure 6). Again, the idea here is that the merging of u into z can affect the parsing of these sibling nodes when they are shrunk down to leaves.

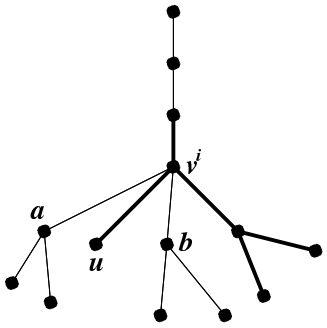

Figure 6: Because node u moves up to v^i, nodes a and b become corner nodes.

Having described N^i, let us now define M^{i+1}, i.e., the influence region at the next level of our parsing. M^{i+1} is the set of those nodes in T^{i+1} which contain a node of N^i. The center of M^{i+1} is the node which contains the center of N^i, i.e., z. Furthermore, any node in T^{i+1} which contains a corner node will again be a corner node.

Initially, define $M^0 = \{v\}$. The children of v which are immediately on the left and on the right of the removed subsequence s are marked as corner nodes. Based on the above rules, it is easy to see that M^i and N^i are always connected subtrees of T^i.

It is important to note that the "extended" influence region N^i is defined in such a manner that the parsing of all nodes in $T^i - N^i$ will not be affected by the changes in M^i. This fact will become clear as we proceed with the proofs in this section. We now define a corresponding set, P^i, in $(T_1+T_2)^i$. The remainder of the section is devoted to proving that the nodes in $T^i - M^i$ and $(T_1 + T_2)^i - P^i$ can be matched in some manner, so that each pair of matched nodes correspond to *identical* subtrees in T and $(T_1 + T_2)$, respectively.

The set P^i in $(T_1 + T_2)^i$ is defined as follows. P^i always contains the root of ${T_1}^i$. Further, a node $u \in (T_1 + T_2)^i$ is in P^i, if and only if there exists a $u' \in M^i$ such that $u(T_1 + T_2) \cap u'(T)$ is non-empty ($u(T_1 + T_2)$, as expected, denotes the subtree corresponding to u in $T_1 \cup T_2$).

We maintain the following invariant : Given any node

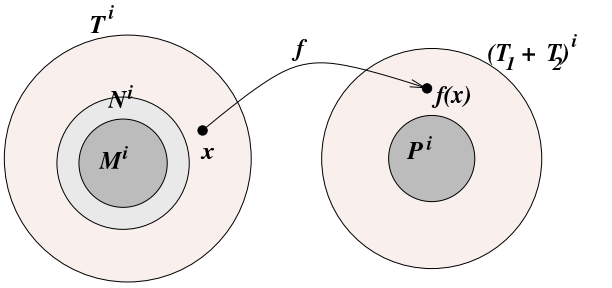

Figure 7: f maps from $T^i - M^i$ to $(T_1 + T_2)^i - P^i$.

$x \in T^i - M^i$, there will be a node $y \in (T_1 + T_2)^i - P^i$ such that $x(T)$ and $y(T_1 + T_2)$ are *identical* subtrees. We denote y by $f(x)$. Conversely, given a node $y \in (T_1 + T_2)^i - P^i$, there is a node $x \in T^i - M^i$ such that $x(T) = y(T_1 + T_2)$. Thus, f is a one-to-one, onto mapping from $T^i - M^i$ to $(T_1 + T_2)^i - P^i$ (Figure 7). In other words, if we ignore M^i and P^i from T^i and $(T_1 + T_2)^i$ respectively, then the two remaining forests are *identical*.

Clearly, our invariant is true in the beginning. Suppose the hypothesis is true for T^i and $(T_1 + T_2)^i$. We now need to prove it for T^{i+1} and $(T_1 + T_2)^{i+1}$. Fix a node w in $T^i - N^i$, and let w' be the corresponding node in $(T_1 + T_2)^i - P^i$ (i.e., $w' = f(w)$). Suppose w is contained in a node q in T^{i+1} and w' in node $q' \in (T_1 + T_2)^{i+1}$.

LEMMA 4.4. *If $q(T)$ and $q'(T_1 + T_2)$ are identical subtrees for any node $w \in T^i - N^i$, then the invariant holds for T^{i+1} and $(T_1 + T_2)^{i+1}$ as well.* ∎

Proof: We have to show the following facts. If x is a node in $T^{i+1} - M^{i+1}$, then there exists a node $y \in (T_1 + T_2)^{i+1} - P^{i+1}$ such that $y(T_1 + T_2) = x(T)$. Conversely, given a node $y \in (T_1 + T_2)^{i+1} - P^{i+1}$, there is a node $x \in T^{i+1} - M^{i+1}$ such that $x(T) = y(T_1 + T_2)$.

Suppose the condition in the lemma holds. Let x be a node in $T^{i+1} - M^{i+1}$. Let x' be a node in T^i such that x contains x'. Clearly, $x' \notin N^i$, otherwise x will be in M^{i+1}. Let $y' = f(x')$. Let y be the node in $(T_1 + T_2)^{i+1}$ which contains y'. By the hypothesis of the lemma, $x(T)$ and $y(T_1 + T_2)$ are identical trees. It remains to check that $y \notin P^{i+1}$. Since $y(T_1 + T_2) = x(T)$, $y(T_1 + T_2)$ is disjoint from $z(T)$ for any $z \in T^{i+1}, z \neq x$. Since $x \notin M^{i+1}$, $y \in (T_1 + T_2)^{i+1} - P^{i+1}$.

Let us prove the converse now. Suppose $y \in (T_1 + T_2)^{i+1} - P^{i+1}$. Let y' be a node in $(T_1 + T_2)^i$ such that y contains y'. If $y' \in P^i$, then there is a node $x' \in M^i$ such that $x'(T) \cap y'(T_1 + T_2) \neq \emptyset$. Let x be the node in T^{i+1} which contains x'. Since $x' \in N^i$, $x \in M^{i+1}$. Now, $x(T) \cap y(T_1 + T_2) \supseteq x'(T) \cap y'(T_1 + T_2) \neq \emptyset$. But then y should be in P^{i+1}, a contradiction. Therefore, $y' \notin P^i$. By the invariant for T^i, there is a node $x' \in T^i - M^i$ such that $y' = f(x')$.

Let x be the node in T^{i+1} containing x'. Again, if $x' \in N^i$, then $x \in M^{i+1}$. But then $x(T) \cap y(T_1 + T_2) \supseteq x'(T) \cap y'(T_1 + T_2)$, which is non-empty because $x'(T) = y'(T_1 + T_2)$. This would imply that $y \in P^{i+1}$. So $x' \notin N^i$. But then, by the hypothesis of the lemma, $x(T) = y(T_1 + T_2)$. Further, x cannot be in M^{i+1}, otherwise y will be in P^{i+1}. Thus, the lemma is true. ∎

It is, therefore, sufficient to prove that $q(T) = q'(T_1 + T_2)$. This is what we seek to do next. Our proof uses a case-by-case analysis of how node w gets parsed in T^i. For each case, we demonstrate that w' will also get parsed in exactly the same manner in the tree $(T_1 + T_2)^i$. In the interest of space and continuity, we defer the details of this proof to the full version of this paper [12].

Thus, we have established the fact that, if we look at the vectors $V(T)$ and $V(T_1 + T_2)$, the nodes corresponding to phase i of $V(T)$ which are not present in $V(T_1 + T_2)$ are a subset of M^i. Our next step is to bound the size of M^i.

LEMMA 4.5. *The influence region M^i for phase i consists of $O(i \log^* n)$ nodes.* ∎

Proof: Note that the number of nodes of degree 2 that are added to M^i (rules (iii) and (iv)) are at most 2Δ. So, adding over first i stages of our algorithm the number of such nodes in M^i can be at most $O(i \log^* n)$. So, we need to bound the number of nodes that get added due to rules (i) and (ii).

We now want to count the number of leaf children of the center node v^i which are in M^i. Let k_i be the number of children of v^i which become leaves for the first time in T^i and are marked as corner nodes. Let C^i be the nodes in M^i which were added as the leaf children of the center node of $T^{i'}$, for some $i' < i$. Then, we claim that C^i can be partitioned into at most $\sum_{j=1}^{i-1} k_j + 1$ contiguous sets such that each set has at most 4Δ elements. We prove this by induction on i. So, suppose it is true for T^i.

Consider such a contiguous set of leaves in C^i – call it C_1^i, $|C_1^i| \leq 4\Delta$. We may add Δ leaves of v^i on either side of C_1^i which are contiguous with C_1^i to the set N^i. Thus, this set may grow to a size of 6Δ contiguous leaves. But when we parse this set, we will reduce it by at least half. Thus, this set will now contain at most 3Δ leaves (which is at most 4Δ).

Therefore, each of the $\sum_{j=1}^{i-1} k_j + 1$ contiguous sets in C^i correspond to a contiguous set in T^{i+1} of size at most 4Δ.

Now, we may add other leaves of v^i to N^i. This can happen only if a corner node becomes a leaf. Δ consecutive leaves on either side of this node will be added to N^i. Thus, we may add k_i more such sets of consecutive leaves to N^i. This completes our inductive argument.

But note that each phase adds at most 2 corner nodes. So, $\sum_{j=1}^{i} k_j \leq 2i$. This shows that the number of nodes in C^i is $O(i \log^* n)$. This proves the result. ∎

We now need to bound the nodes in $(T_1 + T_2)^i$ which are not in T^i. But this can be done in exactly analogous manner if we switch the roles of T and $T_1 + T_2$ in the proofs above. Thus, we can define a subset Q^i of $(T_1 + T_2)^i$ and a 1-1 map g from $(T_1 + T_2)^i - Q^i$ to a subset of T^i such that $g(w)(T) = w(T_1 + T_2)$ for every $w \in (T_1 + T_2)^i - Q^i$. Further we can show in a similar manner that $|Q^i| \leq O(\log^2 n \log^* n)$.

We are now ready to complete the proof of Theorem 4.3.

Proof of Theorem 4.3: Fix a phase i. Consider those subtrees t such that $V_i(T)[< t, i >] \geq V_i(T_1 + T_2)[< t, i >]$. In other words, t appears more frequently in the parsed tree T^i than in $(T_1 + T_2)^i$. Let the set of such subtrees be denoted by $\mathcal{S}$. We first observe that

$$|M^i| \geq \sum_{t \in \mathcal{S}} V_i(T)[< t, i >] - V_i(T_1 + T_2)[< t, i >].$$

Indeed, consider a tree $t \in \mathcal{S}$. Let V_1 be the set of vertices u in T^i such that $u(T) = t$. Similarly, define the set V_2 in $(T_1 + T_2)^i$. So, $|V_1| - |V_2| = V_i(T)[< t, i >] - V_i(T_1 + T_2)[< t, i >]$. Now, the function f must map a vertex in $V_1 - M^i$ to a vertex in V_2. Since f is 1-1, $V_1 - M^i$ can have at most $|V_2|$ nodes. In other words, M^i must contain $|V_1 - V_2|$ nodes from V_1. Adding this up for all such subtrees in $\mathcal{S}$ gives us the inequality above.

We can write a similar inequality for Q^i. Adding these up, we get

$$|M^i| + |Q^i| \geq \sum_t |V_i(T)[< t, i >] - V_i(T_1 + T_2)[< t, i >]|,$$

where the sum is over all subtrees. If we add over all values of i, we get the desired result. ∎

4.2 Lower Bound Proof

Our proof follows along the lower-bound proof of Cormode and Muthukrishnan [9], in that it does not make use of any special properties of our hierarchical tree parsing; instead, we only assume that the parsing structure built on top of the data tree is of bounded degree k (in our case, of course, $k = 3$). The idea is then to show how, given two data trees S and T, we can use the "credit" from the L_1 difference of their vector embeddings $||V(T) - V(S)||_1$ to transform S into T. As in [9], our proof is constructive and shows how the overall parsing structure for S (including S itself at the leaves) can be transformed into that for T; the transformation is performed level-by-level in a bottom-up fashion (starting from the leaves of the parsing structure). (The distance-distortion lower bound for our embedding is an immediate consequence of Lemma 4.6 with $k = 3$.)

LEMMA 4.6. *Assuming a hierarchical parsing structure with degree at most k ($k \geq 2$), the overall parsing structure for tree S can be transformed into exactly that of tree T with at most $(2k - 1)||V(T) - V(S)||_1$ tree-edit operations (node inserts, deletes, relabels, and subtree moves).* ∎

Proof: As in [9], we first perform a top-down pass over the parsing structure of S, marking all nodes x whose subgraph appears in the both parse-tree structures, making sure that the number of marked x nodes at level (i.e., phase) i of the parse tree does not exceed $V_i(T)[x]$ (we use x instead of $v(x)$ to also denote the valid subtree corresponding to x in order to simplify the notation). Descendants of marked nodes are also marked. Marked nodes are "protected" during the parse-tree transformation process described below, in the sense that we do not allow an edit operation to split a marked node.

We proceed bottom-up over the parsing structure for S in $O(\log n)$ rounds (where $n = \max\{|S|, |T|\}$), ensuring that after the end of round i we have created an S_i such that $||V_i(T) - V_i(S_i)||_1 = 0$. The base case (i.e., level 0) deals with simple node labels and creates S_0 in a fairly straightforward way: for each label a, if $V_0(S)[a] > V_0(T)[a]$ then we delete $(V_0(S)[a] - V_0(T)[a])$ unmarked copies of a; otherwise, if $V_0(S)[a] < V_0(T)[a]$, then we add $(V_0(T)[a] - V_0(S)[a])$ leaf nodes labeled a at some location of S. In each case, we perform $|V_0(S)[a] - V_0(T)[a]|$ edit operations which is exactly the contribution of label a to $||V_0(T) - V_0(S)||_1$. It is easy to see that, at the and of the above process, we have $||V_0(T) - V_0(S_0)||_1 = 0$.

Inductively, assume that, when we start the transformation at level i, we have enough nodes at level $i-1$; that is, $||V_{i-1}(T) - V_{i-1}(S_{i-1})||_1 = 0$. We show how to create S_i using at most $(2k-1)||V_i(T) - V_i(S_i)||_1$ subtree-move operations. Consider a node x at level i (again, to simplify the notation, we also use x to denote the corresponding valid subtree). If $V_i(S)[x] > V_i(T)[x]$, then we have exactly $V_i(T)[x]$ marked x nodes at level i of S's parse tree that we will not alter; the remaining copies will be split to form other level-i nodes as described next. If $V_i(S)[x] < V_i(T)[x]$, then we need to build an extra $(V_i(T)[x] - V_i(S)[x])$ copies of the x node at level i. We demonstrate how each such copy can be built by using $\leq (2k-1)$ subtree move operations in order to bring together $\leq k$ level-$(i-1)$ nodes to form x (note that the existence of these level-$(i-1)$ nodes is guaranteed by the fact that $||V_{i-1}(T) - V_{i-1}(S_{i-1})||_1 = 0$). Since $(V_i(T)[x] - V_i(S)[x])$ is exactly the contribution of x to $||V_i(T) - V_i(S_i)||_1$, the overall transformation for level i requires at most $(2k-1)||V_i(T) - V_i(S_i)||_1$ edit operations.

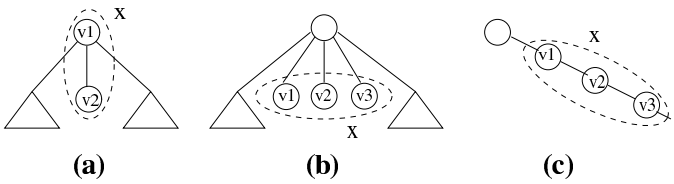

Figure 8: Forming a Level-i Node x.

To see how we form the x node at level i note that, based on our embedding algorithm, there are three distinct cases for the formation of x from level-$(i-1)$ nodes, as depicted in Figure 8(a-c). In case (a), x is formed by "folding" the (no-siblings) leftmost leaf child v_2 of a node v_1 into its parent; we can create the scenario depicted in Figure 8(a) easily with two subtree moves: one to remove any potential subtree rooted at the level-$(i-1)$ node v_2 (we can place it under v_2's original parent at the level-$(i-1)$ tree), and one to move the (leaf) v_2 under the v_1 node. Similarly, for the scenarios depicted in cases (b,c), we basically need at most k subtree moves to turn the nodes involved into leaves, and at most $k-1$ additional moves to move these leaves into the right formation around one of these $\leq k$ nodes. Thus, we can create each copy of x with $\leq (2k-1)$ subtree move operations. At the end of this process, we have $||V_i(T) - V_i(S_i)||_1 = 0$. Note that we do not care *where* in the level-i tree we create the x node; the exact placement will be taken care of at higher levels of the parsing structure. This completes the proof. ∎

5. SKETCHING A MASSIVE, STREAMING XML DATA TREE

In this section, we describe how our tree-edit distance embedding algorithm can be used to obtain a small, pseudo-random *sketch* synopsis of a massive XML data tree in the streaming model. This sketch synopsis requires only small (logarithmic) space, and it can be used as a much smaller surrogate for the entire data tree in approximate tree-edit distance computations with guaranteed error bounds on the quality of the approximation based on the distortion bounds guaranteed from our embedding. Most importantly, as we show in this section, the properties of our embedding algorithm are the key that allows us to build this sketch synopsis in small space as nodes of the tree are streaming by without ever backtracking on the data.

More specifically, consider the problem of embedding a data tree T of size n into a vector space, but this time assume that T is truly massive (i.e., n far exceeds the amount of available storage). Instead, we assume that we see the nodes of T as a continuous data stream in some apriori determined order. In the theorem below, we assume that the nodes of T arrive in the order of a *preorder* (i.e., depth-first and left-to-right) traversal of T. The theorem demonstrates that the vector $V(T)$ constructed for T by our L_1 embedding algorithm can then be constructed in space $O(d \log^2 n \log^* n)$, where d denotes the depth of T. The sketch of T is essentially a sketch of the $V(T)$ vector (denoted by $\texttt{sketch}(V(T))$) that can be used for L_1 distance calculations in the embedding vector space. Such an L_1 sketch of $V(T)$ can be obtained (in small space) using the existing sketching algorithms of Indyk [17].

THEOREM 5.1. *A sketch* $\texttt{sketch}(V(T))$ *to allow approximate tree-edit distance computations can be computed over the stream of nodes in the preorder traversal of an n-node XML data tree T using $O(d \log^2 n \log^* n)$ space and $O(\log d \log^2 n (\log^* n)^2)$ time per node , where d denotes the depth of T. Then, assuming sketch vectors of size $O(\log \frac{1}{\delta})$ and for an appropriate combining function $f()$, $f(\texttt{sketch}(S), \texttt{sketch}(T))$ gives an estimate of the tree-edit distance $d(S,T)$ to within a relative error of $O(\log^2 n \log^* n)$ with probability of at least $1-\delta$.* ∎

The proof of Theorem 5.1 hinges on the fact that, based on our proof in Section 4.1, given a node v on a root-to-leaf path of T and for each of the $O(\log n)$ levels of the parsing structure above v, we only need to retain a local neighborhood (i.e., influence region) of nodes of size at most $O(\log n \log^* n)$ to determine the effect of adding an incoming subtree under T. The $O(d)$ multiplicative factor is needed since, as the tree is streaming in in preorder, we do not really know where a new node will attach itself to T; thus, we have to maintain $O(d)$ such influence regions. Given that most real-life XML data trees are reasonably "bushy", we expect that, typically, $d << n$, or $d = O(\text{polylog}(n))$. The $f()$ combining function is basically a median-selection over the absolute component-wise differences of the two `sketch` vectors [17]. A detailed proof of Theorem 5.1 can be found in the full version of this paper [12].

6. APPROXIMATE SIMILARITY JOINS OVER XML DOCUMENT STREAMS

We now consider the problem of computing (in limited space) the cardinality of an approximate tree-edit-distance similarity join over two continuous data streams of XML documents $\mathcal{S}_1$ and $\mathcal{S}_2$. Note that this is a distinctly different streaming problem from the one examined in Section 5: we now assume massive, continuous streams of *short* XML documents that we want to join based on tree-edit distance; thus, the limiting factor is no longer the size of an individual data tree (which is assumed small and constant) but

rather the number of trees in the stream(s). The documents in each $\mathcal{S}_i$ stream can arrive *in any order*, and our goal is to produce an accurate estimate for the similarity-join cardinality $|\texttt{SimJoin}(\mathcal{S}_1, \mathcal{S}_2, \tau)| = |\{(S,T) \in \mathcal{S}_1 \times \mathcal{S}_2 : d(S,T) \leq \tau\}|$; that is, the number of pairs in $\mathcal{S}_1 \times \mathcal{S}_2$ that are within a tree-edit distance of τ of each other (where the similarity threshold τ is a user/application-defined parameter). Such a space-efficient, one-pass approximate similarity join algorithm would obviously be very useful in processing huge XML databases, integrating streaming XML data sources, and so on.

Once again, the first key step is to utilize our tree-edit distance embedding algorithm on each streaming document tree $T \in \mathcal{S}_i$ $(i = 1, \ldots, 2)$ to construct a (low-distortion) image $V(T)$ of T as a point in an appropriate multi-dimensional vector space. We then obtain a lower-dimensional L_1-sketch of $V(T)$ that approximately preserves L_1 distances in the original vector space, as described by Indyk [17]. Our tree-edit distance similarity join has now essentially been transformed into an L_1-distance similarity join in the embedding, low-dimensional vector space. The final step then performs an additional level of sketching over the stream of points in the embedding L_1 vector space in order to build a randomized, sketch-based estimate for $|\texttt{SimJoin}(\mathcal{S}_1, \mathcal{S}_2, \tau)|$. The following theorem shows how an *atomic* sketch-based estimate can be constructed in small space over the streaming XML-data trees; to boost accuracy and probabilistic confidence, several independent atomic-estimate instances can be used (as in [1, 2, 10]).

THEOREM 6.1. *Let $|\texttt{SimJoin}(\mathcal{S}_1, \mathcal{S}_2, \tau)|$ denote the the cardinality of the tree-edit distance similarity join between two XML document streams $\mathcal{S}_1$ and $\mathcal{S}_2$, where document distances are approximated to within a factor of $O(\log^2 b \log^* b)$ with constant probability, and b is a (constant) upper bound on the size of each document tree. An atomic, sketch-based estimate for $|\texttt{SimJoin}(\mathcal{S}_1, \mathcal{S}_2, \tau)|$ can be constructed in $O(b + \log \frac{1}{\delta} \log N)$ space and $O(\frac{1}{\delta} + b \log^* b)$ time per document, where δ controls the accuracy of the distance estimates and N denotes the length of the input stream(s).* ∎

Proof: Our algorithm for producing an atomic sketch estimate for the similarity join cardinality uses two distinct levels of sketching. Assume an input tree T (in one of the input streams). The first level of sketching uses our L_1 embedding algorithm in conjunction with the L_1-sketching technique of Indyk [17] (i.e., with 1-stable (Cauchy) random variates) to map T to an $O(\log \frac{1}{\delta})$-dimensional vector of sketching values $\texttt{sketch}(V(T))$ in a manner similar to that described in Theorem 5.1. This mapping of an input tree T to a point in an $O(\log \frac{1}{\delta})$-dimensional vector space can be done in space $O(b + \log \frac{1}{\delta} \log N)$: this covers the $O(b)$ space to store and parse the tree, and the $O(\log N)$ space required to generate the random variates for each of the $O(\log \frac{1}{\delta})$ sketch-value computations (and store the sketch values themselves). (Note that $O(\log N)$ space is sufficient since we know that there are at most $O(Nb)$ non-zero components in all the $V(T)$ vectors in the entire data stream.) A key property of this mapping is that the L_1 distances of the $V(T)$ vectors are approximately preserved in this new $O(\log \frac{1}{\delta})$-dimensional vector space with *constant probability* [17].

The second level of sketching in our construction will produce a pseudo-random sketch of the point-distribution (in the embedding vector space) for each input data stream. To deal with an L_1 τ-similarity join, the basic equi-join sketching technique discussed in Section 2 needs to be appropriately adapted. The key idea here is to view each incoming "point" $\texttt{sketch}(V(T))$ in *one of the two data streams*, say $\mathcal{S}_1$, as an L_1 region of points (i.e., a multi-dimensional hypercube) of radius τ centered around $\texttt{sketch}(V(T))$ in the embedding $O(\log \frac{1}{\delta})$-dimensional vector space when building a sketch synopsis for stream $\mathcal{S}_1$. Essentially, this means that when T (i.e., $\texttt{sketch}(V(T))$) is seen in the $\mathcal{S}_1$ input, instead of simply adding the random variate $\xi_{\vec{i}}$ (where, the index $\vec{i} = \texttt{sketch}(V(T))$) to the atomic sketch estimate X_{S_1} for S_1, we update $X_{\mathcal{S}_1}$ by adding $\sum_{\vec{j} \in N(\vec{i},\tau)} \xi_{\vec{j}}$, where $N(\vec{i}, \tau)$ denotes the L_1 neighborhood of radius τ of $\vec{i} = \texttt{sketch}(V(T))$ in the embedding vector space (i.e., $N(\vec{i}, \tau) = \{\vec{j} : \|\vec{i} - \vec{j}\|_1 \leq \tau\}$). Note that this special processing is only carried out on the $\mathcal{S}_1$ stream; the sketch $X_{\mathcal{S}_2}$ for the second XML stream $\mathcal{S}_2$ is updated in the standard manner. It is then fairly simple to prove (as in [1, 10]) that the product $X_{\mathcal{S}_1} \cdot X_{\mathcal{S}_2}$ gives an unbiased, atomic sketching estimate for the cardinality of the L_1 τ-similarity join of $\mathcal{S}_1$ and $\mathcal{S}_2$ in the embedding $O(\log \frac{1}{\delta})$-dimensional vector space.

In terms of processing time per document, note that, in addition to time cost of our embedding process, the first level of sketching can be done in small time using the techniques discussed by Indyk [17]. The second level of sketching can also be implemented using standard sketching techniques, with the difference that (for one of the two streams) updating would require summation of ξ variates over an L_1 neighborhood of radius τ in an $O(\log \frac{1}{\delta})$-dimensional vector space. This basically means that the per-document sketching cost would increase by a multiplicative factor of $O(\tau^{O(\log(1/\delta))}) = O(1/\delta)$ in the worst-case (in fact, efficiently range-summable sketching variables, as in [11], can be used to reduce this multiplicative factor to $O(\log(1/\delta))$). ∎

Theorem 6.1 only guarantees that our estimation algorithm approximates tree-edit distances with *constant* probability. This is due to the fact that using the L_1 sketching technique of Indyk as a tool for L_1 dimensionality reduction can only guarantee a mapping that is approximately non-expansive with constant probability [17]; in other words, this means that a constant fraction of the points in the τ-neighborhood of a given point could be missed. (The high-probability result of Theorem 5.1 relies on median-selection rather than L_1 norm computation in the $O(\log \frac{1}{\delta})$-dimensional space, and median-selection cannot be used efficiently for similarity-join computations.) Furthermore, the very recent results of Charikar and Sahai [7] prove that *no sketching method* (based on randomized linear projections) can provide a high-probability dimensionality-reduction tool for L_1; in other words, there is no analogue of the Johnson-Lindenstrauss (JL) lemma for the L_1 norm. Thus, there seems to be no obvious way to strengthen Theorem 6.1 with *high-probability* distance estimates.

The following corollary shows that high-probability estimates are possible if we allow for an extra $O(\sqrt{b})$ multiplicative factor in the distance distortion. The idea here is to use L_2 vector norms to approximate L_1 norms exploiting the fact that each $V(T)$ vector has at most $O(b)$ non-zero components, and then use standard, high-probability L_2 dimensionality reduction (e.g., through JL). Of course, a different approach that could give stronger results would be to try to

embed tree-edit distance *directly into* L_2, but this remains an open problem.

COROLLARY 6.2. *The tree-edit distances for the estimation of* $|\texttt{SimJoin}(\mathcal{S}_1, \mathcal{S}_2, \tau)|$ *in Theorem 6.1 can be estimated with high probability to within a factor of* $O(\sqrt{b}\log^2 b \log^* b)$. ∎

7. CONCLUSIONS

In this paper, we have presented the first algorithmic results on the problem of effectively correlating (in small space) massive XML data streams based on approximate tree-edit distance computations. Our solution relies on a novel algorithm for obliviously embedding XML trees as points in an L_1 vector space while guaranteeing a logarithmic worst-case upper bound on the distance distortion. We have combined our embedding algorithm with pseudo-random sketching techniques to obtain novel, small-space algorithms for building concise sketch synopses and approximating similarity joins over streaming XML data. Our embedding result also has other important algorithmic applications, e.g., as a tool for very fast, approximate tree-edit distance computations.

8. REFERENCES

[1] Noga Alon, Phillip B. Gibbons, Yossi Matias, and Mario Szegedy. "Tracking Join and Self-Join Sizes in Limited Storage". In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Philadeplphia, Pennsylvania, May 1999.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. "The Space Complexity of Approximating the Frequency Moments". In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, pages 20–29, Philadelphia, Pennsylvania, May 1996.

[3] Alberto Apostolico and Zvi Galil, editors. *"Pattern Matching Algorithms"*. Oxford University Press, 1997.

[4] Ziv Bar-Yossef, T.S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. "Counting distinct elements in a data stream". In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM'02)*, Cambridge, Massachusetts, September 2002.

[5] Chee-Yong Chan, Pascal Felber, Minos Garofalakis, and Rajeev Rastogi. "Efficient Filtering of XML Documents with XPath Expressions". In *Proceedings of the Eighteenth International Conference on Data Engineering*, San Jose, California, February 2002.

[6] Moses Charikar, Kevin Chen, and Martin Farach-Colton. "Finding Frequent Items in Data Streams". In *Proceedings of the International Colloquium on Automata, Languages, and Programming*, Malaga, Spain, July 2002.

[7] Moses Charikar and Amit Sahai. "Dimension Reduction in the l_1 Norm". In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, Vancouver, Canada, November 2002.

[8] Graham Cormode, Mayur Datar, Piotr Indyk, and S. Muthukrishnan. "Comparing Data Streams Using Hamming Norms (How to Zero In)". In *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China, August 2002.

[9] Graham Cormode and S. Muthukrishnan. "The String Edit Distance Matching Problem with Moves". In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, California, January 2002.

[10] Alin Dobra, Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. "Processing Complex Aggregate Queries over Data Streams". In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Madison, Wisconsin, June 2002.

[11] Joan Feigenbaum, Sampath Kannan, Martin Strauss, and Mahesh Viswanathan. "An Approximate L^1-Difference Algorithm for Massive Data Streams". In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, New York City, New York, October 1999.

[12] Minos Garofalakis and Amit Kumar. "Correlating XML Data Streams Using Tree-Edit Distance Embeddings". Bell Labs Technical Memorandum, March 2003.

[13] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. "Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries". In *Proceedings of the 27th International Conference on Very Large Data Bases*, Roma, Italy, September 2001.

[14] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. "How to Summarize the Universe: Dynamic Maintenance of Quantiles". In *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China, August 2002.

[15] Michael Greenwald and Sanjeev Khanna. "Space-Efficient Online Computation of Quantile Summaries". In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, Santa Barbara, California, May 2001.

[16] Sudipto Guha, H.V. Jagadish, Nick Koudas, Divesh Srivastava, and Ting Yu. "Approximate XML Joins". In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Madison, Wisconsin, June 2002.

[17] Piotr Indyk. "Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computation". In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 189–197, Redondo Beach, California, November 2000.

[18] Piotr Indyk. "Algorithmic Aspects of Geometric Embeddings". In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, Las Vegas, Nevada, October 2001.

[19] Richard M. Karp and Michael O. Rabin. "Efficient Randomized Pattern-Matching Algorithms". *IBM Journal of Research and Development*, 31(2):249–260, March 1987.

[20] Donald E. Knuth. *"The Art of Computer Programming (Vol. 1 / Fundamental Algorithms)"*. Reading, Mass. : Addison-Wesley Pub. Co., 1973.

[21] Gurmeet Singh Manku and Rajeev Motwani. "Approximate Frequency Counts over Data Streams". In *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China, August 2002.

[22] Nitin Thaper, Sudipto Guha, Piotr Indyk, and Nick Koudas. "Dynamic Multidimensional Histograms". In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Madison, Wisconsin, June 2002.

APPENDIX

A. BOUNDING THE NUMBER OF PHASES

Proof of Lemma 4.2: For a tree H, we say that a node $v \in H$ is a *leaf child* of a node $u \in H$ if v is a leaf and a child of u. Suppose v is a leaf child of u. Define $\mathbf{s}_v$ as the maximal contiguous set of leaf children of u which contains v. Define a *chain* to be a path in H which goes from a node to its descendant such that all nodes in this path have degree 2.

For a tree H, we partition its vertex set into several sets as follows. Let $A(H)$ be the non-root nodes which either have degree two or are leaf children of non-root nodes of degree two. Observe that the non-leaf nodes in $A(H)$ have only one child. Define $B(H)$ as the nodes with two or more children. The root is also added to $B(H)$. So $A(H)$ and $B(H)$ contain all the non-leaf nodes of T^i. It remains to consider leaves in H which have at least one sibling or which are children of the root. Let $D_1(H)$ be the set of all such leaves u for which $|\mathbf{s}_u| \geq 2$. Let $D_2(H)$ be the set of all such leaves for which $|\mathbf{s}_u| = 1$ and u is the leftmost such child of its parent. Define $D_3(H)$ to be the remaining leaves of H, i.e., leaves u for which $|\mathbf{s}_u| = 1$ but u is not the leftmost such child of its parent. For notational convenience, we shall also denote $|A(H)|$ by $A(H)$ and similarly for other sets.

We first prove the following ancillary lemma.

LEMMA A.1. *For any rooted tree H with at least 2 nodes, $D_3(H) \leq D_2(H) + A(H)/2 - 1$.* ∎

Proof: We prove this by induction on the number of nodes in H. Suppose H has only two nodes. Then $D_3(H) = 0, D_2(H) = 1, A(H) = 0$. So the lemma is true for the base case.

Suppose it is true for all rooted trees with less than n nodes. Suppose H has n nodes. Let r be the root of H. First consider the case when r has only one child, call it s. Let H' be the subtree rooted at s. By induction, $D_3(H') \leq D_2(H') + A(H')/2 - 1$. Clearly, $D_3(H) = D_3(H')$. Is $D_2(H)$ equal to $D_2(H')$? The only case when a node u can occur in $D_2(H')$ but not in $D_2(H)$ is when s has only one child, u, which also happens to be a leaf. Clearly, $u \in D_2(H')$ because its parent is the root. But this is not the case in H. So $u \notin D_2(H)$. But if this is the case then both s and r are in $A(H) - A(H')$. So, $D_2(H) + A(H)/2 = D_2(H') + A(H')/2$. Thus the lemma is true in this case as well.

Now consider the case when r has at least 2 children. We shall construct several subtrees, each of these will be rooted at r (but will contain only a subset of the descendants of r). Let $u_1, \ldots, u_k$ be the leaf children of r such that $\mathbf{s}_{u_i} = \{u_i\}$. So, $u_1 \in D_2(H)$ whereas all other $u_i \in D_3(H)$. We define subtrees $H_1, \ldots, H_{k+1}$ as follows. H_i is the set of all descendants of r (including r) which lie to the right of u_{i-1} but to the left of u_i (as special cases, H_1 is the set of nodes to the left of u_1 and H_{k+1} as the set of nodes to the right of u_k). It may happen that H_1 or H_{k+1} do not contain any nodes (except the root), but all other subtrees will have at least one node other than the root r. By induction,

$$D_3(H_i) \leq D_2(H_i) + A(H_i)/2 - 1$$

for all H_i except perhaps H_1 and H_{k+1} (in case they do not contain any nodes except the root). Adding these inequalities, we get

$$\sum_i D_3(H_i) \leq \sum_i D_2(H_i) + \sum_i A(H_i)/2 - (k-1).$$

Note that we have $k-1$ only in the right hand side because H_1 and H_{k+1} may not contribute to this sum.

Now, if $u \in A(H_i)$, then $u \in A(H)$ as well. So, $A(H) = \sum_i A(H_i)$. Suppose $u \in D_2(H_i)$. Let the parent of u be w. Note that $w \neq r$. Indeed, suppose $w = r$. Since $\mathbf{s}_u$ contains a leaf node other than u it must be the case that u is adjacent to one of the nodes $u_1, \ldots, u_k$. But there can not be a leaf node adjacent to $u_1, \ldots, u_k$. So $w \neq r$. But then $u \in D_2(H)$ as well. Conversely, suppose $u \in D_2(H)$. Either $u = u_1$ or the parent of u is in one of the subtrees H_i. In the latter case $u \in D_2(H_i)$. Thus, $D_2(H) = \sum_i D_2(H_i) + 1$.

Finally, we can argue in a similar manner that $D_3(H_i) \subset D_3(H)$. Further, if $u \in D_3(H)$, then either $u \in \{u_2, \ldots, u_k\}$ or $u \in D_3(H_i)$. Thus, $D_3(H) = \sum_i D_3(H_i) + k - 1$. Putting everything together, we see that

$$\begin{aligned} D_3(H) &= \sum_i D_3(H_i) + k - 1 \\ &\leq \sum_i D_2(H_i) + \sum_i A(H_i)/2 \\ &= D_2(H) + A(H)/2 - 1 \end{aligned}$$

This proves the lemma. ∎

We show that the number of nodes goes down by a constant factor after each phase. Recall that T^i is the tree at the beginning of phase i.

We claim that

$$B(T^{i+1}) \leq B(T^i), \quad B(T^{i+1}) + A(T^{i+1}) \leq B(T^i) + A(T^i)/2.$$

Indeed all nodes which have degree at least 3 (i.e., at least 2 children) in T^{i+1} must have had degree at least 3 in T^i as well. This proves the first inequality. So any node in $B(T^{i+1})$ corresponds to a unique node in $B(T^i)$. Now consider a node u in $A(T^{i+1})$. Two cases can happen depending on how u was formed. One case is that u was formed by collapsing some degree 2 nodes from $A(T^i)$ — in this case u corresponds to at least 2 nodes from $A(T^i)$. The other case is that there was a vertex $w \in B(T^i)$ and we collapsed a leaf child of w into w to get u. In this case u corresponds to a unique node of $B(T^i)$. This proves the second inequality.

In phase i, the leaves in $D_1(T^i)$ will be reduced to at least half their size. Those in $D_2(T^i)$ will merge with their parents. The leaves in $D_3(T^i)$ do not change. So we need to bound their size. The lemma above implies that

$$D_3(T^i) \leq D_2(T^i) + A(T^i)/2.$$

From this we get

$$D_3(T^i) \leq 2/3(D_2(T^i)/2 + D_3(T^i) + A(T^i)/4).$$

Thus, the number of leaves in T^{i+1} is at most $D_1(T^i)/2 + D_2(T^i)/3 + 2/3 D_3(T^i)/3 + A^i/6$, which is at most $2/3 D(T^i) + A(T^i)/6$, where $D(T^i)$ is the number of leaves in T^i. So the total number of nodes in T^{i+1} is at most $B(T^i) + c(A(T^i) + D(T^i))$, where c is some constant less than 1.

Now,

$$\begin{aligned} B(T^i) + c(A(T^i) + D(T^i)) &= \frac{1+c}{2} B(T^i) + \frac{1-c}{2} B(T^i) + \\ &\quad + c(A(T^i) + D(T^i)) \\ &\leq \frac{1+c}{2} B(T^i) + \frac{1+c}{2} \left(A(T^i) + D(T^i) \right) \end{aligned}$$

because $B(T^i) \leq A(T^i) + D(T^i)$ (the number of nodes of degree more than 2 is at most the number of leaves in any tree). Thus, the number of nodes in T^{i+1} goes down by a constant factor. This completes the proof. ∎