XML Stream Processing Using Tree-Edit Distance Embeddings

MINOS GAROFALAKIS
Bell Labs, Lucent Technologies
and
AMIT KUMAR
Indian Institute of Technology

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 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. 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. Experimental results from an empirical study with both synthetic and real-life XML data trees validate our approach, demonstrating that the average-case behavior of our embedding techniques is much better than what would be predicted from our theoretical worst-case distortion bounds. 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.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Query processing*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*Combinatorial algorithms*

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: XML, data streams, data synopses, approximate query processing, tree-edit distance, metric-space embeddings

1. INTRODUCTION

The Extensible Markup Language (XML) is rapidly emerging as the new standard for data representation and exchange on the Internet. The simple,

A preliminary version of this article appeared in *Proceedings of the 22nd Anuual ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (San Diego, CA, June) [Garofalakis and Kumar 2003].
Authors' addresses: M. Garofalakis, Bell Labs, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974; email: minos@research.bell-labs.com; A. Kumar, Department of Computer Science and Engineering, Indian Institute of Technology, Hauz Khas, New Delhi-110016, India; email: amitk@cse.iitd.ernet.in.
Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 2005 ACM 0362-5915/05/0300-0279 $5.00

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, rather than simply specifying how elements are to be displayed (as in HTML). Thus, XML data is tree-structured and self-describing.

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 [Guha et al. 2002].

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* [Apostolico and Galil 1997; Zhang and Shasha 1989]. In a nutshell, the tree-edit distance metric is the natural generalization of edit distance from the string domain; thus, 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).[1] The problem becomes particularly challenging in the context of *streaming* XML data sources, that is, when such correlation queries must be evaluated 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) [Alon et al. 1996, 1999;

[1]Specific semantics associated with XML node labels and tree-edit operations can be captured using a generalized, *weighted* tree-edit distance metric that associates different weights/costs with different operations. Extending the algorithms and results in this article to weighted tree-edit distance is an interesting open problem.

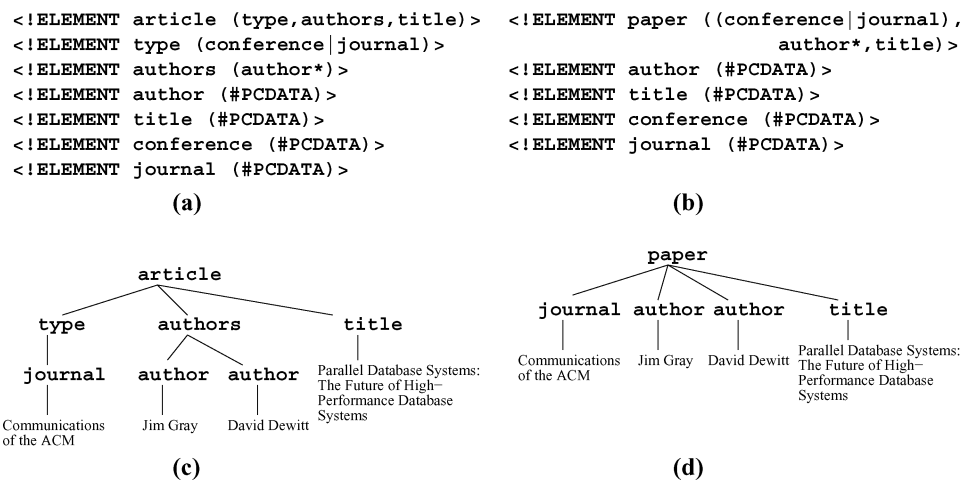


Fig. 1. Example DTD fragments (a) and (b) and XML Document Trees (c) and (d) for autonomous bibliographic Web sources.

Dobra et al. 2002; Gilbert et al. 2001]. 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 and attractive alternative [Babcock et al. 2002; Garofalakis et al. 2002].

Example 1.1. Consider the problem of integrating XML data from two autonomous, bibliographic Web sources WS_1 and WS_2. One of the key issues in such data-integration scenarios is that of *detecting (approximate) duplicates* across the two sources [Dasu and Johnson 2003]. For autonomously managed XML sources, such duplicate-detection tasks are complicated by the fact that the sources could be using different DTD structures to describe their entries. As a simple example, Figures 1(a) and 1(b) depict the two different DTD fragments employed by WS_1 and WS_2 (respectively) to describe XML trees for academic publications; clearly, WS_1 uses a slightly different set of tags (i.e., `article` instead of `paper`) as well as a "deeper" DTD structure (by adding the `type` and `authors` structuring elements).

Figures 1(c) and 1(d) depict two example XML document trees T_1 and T_2 from WS_1 and WS_2, respectively; even though the two trees have structural differences, it is obvious that T_1 and T_2 represent the same publication. In fact, it is easy to see that T_1 and T_2 are within a tree-edit distance of 3 (i.e., one relabel and two delete operations on T_1). Approximate duplicate detection across WS_1 and WS_2 can be naturally expressed as a *tree-edit distance similarity join* operation that returns the pairs of trees $(T_1, T_2) \in WS_1 \times WS_2$ that are within a tree-edit distance of τ, where the user/application-defined similarity threshold τ is set to a value ≥ 3 to perhaps account for other possible differences in the joining tree structures (e.g., missing or misspelled coauthor

names). A single-pass, space-efficient technique for approximating such similarity joins as the document trees from the two XML data sources are streaming in would provide an invaluable data-integration tool; for instance, estimates of the similarity-join result size (i.e., the number of approximate duplicate entries) can provide useful indicators of the degree of *overlap* (i.e., "content similarity") or *coverage* (i.e., "completeness") of autonomous XML data sources [Dasu and Johnson 2003; Florescu et al. 1997].

1.1 Prior Work

Techniques for *data reduction* and *approximate query processing* for both relational and XML databases have received considerable attention from the database research community in recent years [Acharya et al. 1999; Chakrabarti et al. 2000; Garofalakis and Gibbons 2001; Ioannidis and Poosala 1999; Polyzotis and Garofalakis 2002; Polyzotis et al. 2004; Vitter and Wang 1999]. The vast majority of such proposals, however, rely on the assumption of a static data set which enables either several passes over the data to construct effective *data synopses* (such as histograms [Ioannidis and Poosala 1999] or Haar wavelets [Chakrabarti et al. 2000; Vitter and Wang 1999]); clearly, this assumption renders such solutions inapplicable in a data-stream setting. 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 [Greenwald and Khanna 2001; Gilbert et al. 2002b]; distinct-element counting [Bar-Yossef et al. 2002; Cormode et al. 2002a]; frequent itemset counting [Charikar et al. 2002; Manku and Motwani 2002]; estimating frequency moments, join sizes, and difference norms [Alon et al. 1996, 1999; Dobra et al. 2002; Feigenbaum et al. 1999; Indyk 2000]; and, computing one- or multidimensional histograms or Haar wavelet decompositions [Gilbert et al. 2002a; Gilbert et al. 2001; Thaper et al. 2002]. All these articles rely on an approximate query-processing model, typically based on an appropriate underlying *stream-synopsis* data structure. (A different approach, explored by the Stanford STREAM project [Arasu et al. 2002], is to characterize subclasses of queries that can be computed *exactly* with bounded memory.) The synopses of choice for a number of the above-cited data-streaming articles are based on the key idea of *pseudorandom 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 has dealt with XML document streams, but only in the context of simple, predicate-based *filtering* of individual documents [Altinel and Franklin 2000; Chan et al. 2002; Diao et al. 2003; Gupta and Suciu 2003; Lakshmanan and Parthasarathy 2002]; more recent work has also considered possible transformations of the XML documents in order to produce customized output [Diao and Franklin 2003]. 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. [2002] discussed several different algorithms for performing tree-edit distance joins over XML databases. Their work introduced 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. focused solely on *exact* join computation and their algorithms require *multiple passes* over the data; this obviously renders them inapplicable in a data-stream setting.

1.2 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.

In this article, we propose the *first* known solution to the problem of approximating (in small space) the result of correlation queries based on tree-edit distance (such as the tree-edit distance similarity joins described in Example 1.1) over continuous XML data streams. The centerpiece of our solution is a novel algorithm for effectively (i.e., "obliviously" [Indyk 2001]) *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.[2] 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 [2002] for embedding a variant of the string-edit distance (that, in addition to standard string edits, includes an atomic "substring move" operation) 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.[3]

We also demonstrate how our vector-space embedding construction can be combined with earlier sketching techniques [Alon et al. 1999; Dobra et al. 2002; Indyk 2000] 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

[2]All log's in this article 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.

[3]Note that other known techniques for approximating string-edit distance based on the decomposition of strings into q-grams [Ukkonen 1992; Gravano et al. 2001] only give *one-sided* error guarantees, essentially offering *no guaranteed upper bound* on the distance distortion. For instance, it is not difficult to construct examples of very distinct strings with nearly identical q-gram sets (i.e., arbitrarily large distortion). Furthermore, to the best of our knowledge, the results in Ukkonen [1992] have not been extended to the case of trees and tree-edit distance.

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. Finally, we present results from an empirical study of our embedding algorithm with both synthetic and real-life XML data trees. Our experimental results offer some preliminary validation of our approach, demonstrating that the average-case behavior of our techniques over realistic data sets is much better than what our theoretical worst-case distortion bounds would predict, and revealing several interesting characteristics of our algorithms in practice. To the best of our knowledge, ours 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. For instance, *exact* tree-edit distance computation is typically a computationally-expensive problem that can require up to $O(n^4)$ time (for the conventional tree-edit distance metric [Apostolico and Galil 1997; Zhang and Shasha 1989]), and is, in fact, $\mathcal{NP}$-hard for the variant of tree-edit distance considered in this article (even for the simpler case of flat strings [Shapira and Storer 2002]). In contrast, our embedding scheme can be used to provide an *approximate* tree-edit distance (to within a guaranteed $O(\log^2 n \log^* n)$ factor) in *near-linear*, that is, $O(n \log^* n)$, time.

1.3 Organization

The remainder of this article is organized as follows. Section 2 presents background material on XML, tree-edit distance and data-streaming techniques. In Section 3, we present an overview of our approach for correlating XML data streams based on tree-edit distance embeddings. Section 4 presents our embedding algorithm in detail and proves its small-time and low distance-distortion guarantees. We then discuss two important applications of our algorithm for XML stream processing, namely (1) building a sketch synopsis of a massive, streaming XML data tree (Section 5), and (2) approximating similarity joins over streams of XML documents (Section 6). We present the results of our empirical study with synthetic and real-life XML data in Section 7. Finally, Section 8 outlines our conclusions. The Appendix provides ancillary lemmas (and their proofs) for the upper bound result.

2. PRELIMINARIES

2.1 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

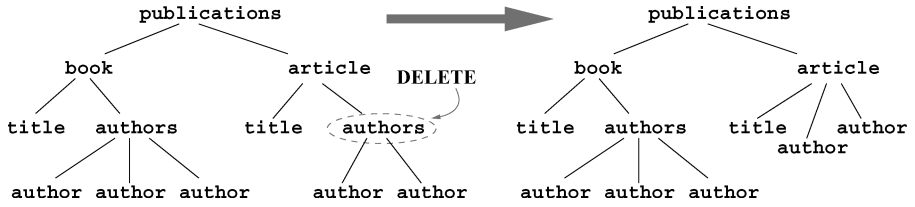


Fig. 2. Example XML tree and tree-edit operation.

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 [Apostolico and Galil 1997; Zhang and Shasha 1989] 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 complement of node deletion.) An example XML tree and tree-edit operation are depicted in Figure 2. In this article, 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 article, the term *tree-edit distance* assumes the four primitive edit operations described above, namely, node relabelings, deletions, insertions, and subtree moves.[4]

2.2 Data Streams and Basic Pseudorandom 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 [Alon et al. 1996, 1999; Dobra et al. 2002]. Backtracking over the stream and explicit access to past data records are 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.

[4]The problem of designing efficient (i.e., "oblivious"), guaranteed-distortion embedding schemes for the standard tree-edit distance metric remains open; of course, this is also true for the much simpler standard *string-edit distance* metric (i.e., without "substring moves") [Cormode and Muthukrishnan 2002].

We focus on one particular type of stream synopses, namely, *pseudorandom sketches*; sketches have provided effective solutions for several streaming problems, including join and multijoin processing [Alon et al. 1996, 1999; Dobra et al. 2002], norm computation [Feigenbaum et al. 1999; Indyk 2000], distinct-element counting [Cormode et al. 2002a], and histogram or Haar-wavelet construction [Gilbert et al. 2001; Thaper et al. 2002]. We describe the basics of pseudorandom sketching schemes using a simple binary-join cardinality estimation query [Alon et al. 1999]. More specifically, assume that we want to estimate $Q = \texttt{COUNT}(R_1 \bowtie_A R_2)$, that is, the cardinality of the binary equijoin of two streaming relations R_1 and R_2 over a (numeric) attribute (or, set of attributes) A, whose values we assume (without loss of generality) to range over $\{1, \ldots, N\}$. (Note that, by the definition of the equijoin operator, the two join attributes have identical value domains.) Letting $f_k(i)$ $(k = 1, 2;$ $i = 1, \ldots, N)$ denote the frequency of the ith value in R_k, is is easy to see that $Q = \sum_{i=1}^{N} f_1(i) f_2(i)$. Clearly, estimating this join size exactly requires at least $\Omega(N)$ space, making an exact solution impractical for a data-stream setting.

In their seminal work, Alon et al. [Alon et al. 1996, 1999] proposed a randomized technique that can offer strong probabilistic guarantees on the quality of the resulting join-size estimate while using space that can be significantly smaller than N. Briefly, the key idea is to (1) build an *atomic sketch* X_k (essentially, a randomized linear projection) of the distribution vector for each input stream R_k $(k = 1, 2)$ (such a sketch can be easily computed over the streaming values of R_k in only $O(\log N)$ space) and (2) use the atomic sketches X_1 and X_2 to define a random variable X_Q such that (a) X_Q is an *unbiased* (i.e., correct on expectation) randomized estimator for the target join size, so that $E[X_Q] = Q$, and (b) X_Q's variance (Var$[X_Q]$) can be appropriately upper-bounded to allow for probabilistic guarantees on the quality of the Q estimate. More formally, this random variable X_Q is constructed on-line from the two data streams as follows:

—Select a family of *four-wise independent binary random variates* $\{\xi_i : i = 1, \ldots, N\}$, where each $\xi_i \in \{-1, +1\}$ and $P[\xi_i = +1] = P[\xi_i = -1] = 1/2$ (i.e., $E[\xi_i] = 0$). Informally, the four-wise independence condition means that, for any 4-tuple of ξ_i variates and for any 4-tuple of $\{-1, +1\}$ values, the probability that the values of the variates coincide with those in the $\{-1, +1\}$ 4-tuple is exactly $1/16$ (the product of the equality probabilities for each individual ξ_i). The crucial point here is that, by employing known tools (e.g., orthogonal arrays) for the explicit construction of small sample spaces supporting four-wise independence, such families can be efficiently constructed on-line using only $O(\log N)$ space [Alon et al. 1996].

—Define $X_Q = X_1 \cdot X_2$, where the atomic sketch X_k is defined simply as $X_k = \sum_{i=1}^{N} f_k(i)\xi_i$, for $k = 1, 2$. Again, note that each X_k is a simple randomized linear projection (inner product) of the frequency vector of $R_k.A$ with the vector of ξ_i's that can be efficiently generated from the streaming values of A as follows: start a counter with $X_k = 0$ and simply add ξ_i to X_k whenever the ith value of A is observed in the R_k stream.

The quality of the estimation guarantees can be improved using a standard *boosting technique* that maintains several independent identically distributed (iid) instantiations of the above process, and uses averaging and median-selection operators over the X_Q estimates to boost accuracy and probabilistic confidence [Alon et al. 1996]. (Independent instances can be constructed by simply selecting independent random seeds for generating the families of four-wise independent ξ_i's for each instance.) We use the term *(atomic) AMS sketch* to describe a randomized linear projection computed in the above-described manner over a data stream. Letting SJ_k $(k = 1, 2)$ denote the *self-join size* of $R_k.A$ (i.e., $\mathrm{SJ}_k = \sum_{i=1}^{N} f_k(i)^2$), the following theorem [Alon et al. 1999] shows how sketching can be applied for estimating binary-join sizes in limited space. (By standard Chernoff bounds [Motwani and Raghavan 1995], using median-selection over $O(\log(1/\delta))$ of the averages computed in Theorem 2.1 allows the confidence in the estimate to be boosted to $1 - \delta$, for any pre-specified $\delta < 1$.)

THROEM 2.1 [ALON ET AL. 1999]. *Let the atomic AMS sketches X_1 and X_2 be as defined above. Then, $E[X_Q] = E[X_1X_2] = Q$ and $\mathrm{Var}(X_Q) \leq 2 \cdot \mathrm{SJ}_1 \cdot \mathrm{SJ}_2$. Thus, averaging the X_Q estimates over $O(\frac{\mathrm{SJ}_1 \cdot \mathrm{SJ}_2}{Q^2\epsilon^2})$ iid instantiations of the basic scheme, guarantees an estimate that lies within a relative error of at most ϵ from Q with constant probability $> 1/2$.*

It should be noted that the space-usage bounds stated in Theorem 2.1 capture the *worst-case* behavior of AMS-sketching-based estimation—empirical results with synthetic and real-life data sets have demonstrated that the average-case behavior of the AMS scheme is much better [Alon et al. 1999]. More recent work has led to improved AMS-sketching-based estimators with provably better space-usage guarantees (that actually match the lower bounds shown by Alon et al. [1999]) [Ganguly et al. 2004], and has demonstrated that AMS-sketching techniques can be extended to effectively handle one or more complex *multi-join* aggregate SQL queries over a collection of relational streams [Dobra et al. 2002, 2004].

Indyk [2000] discussed a different type of pseudorandom sketches which are, once again, defined as randomized linear projections $X_k = \sum_{i=1}^{N} f_k(i)\xi_i$ of a streaming input frequency vector for the values in R_k, but using random variates $\{\xi_i\}$ drawn from a *p-stable distribution* (which can again be generated in small space, i.e., $O(\log N)$ space) in the X_k computation. The class of p-stable distributions has been studied for some time (see, e.g., Nolan [2004]; [Uchaikin and Zolotarev 1999])—they are known to exist for any $p \in (0, 2]$, and include well-known distribution functions, for example, the Cauchy distribution (for $p = 1$) and the Gaussian distribution (for $p = 2$). As the following theorem demonstrates, such *p-stable sketches* can provide accurate probabilistic estimates for the L_p-difference norm of streaming frequency vectors in small space, for any $p \in (0, 2]$.

THEOREM 2.2 [INDYK 2000]. *Let $p \in (0, 2]$, and define the p-stable sketch for the R_k stream as $X_k = \sum_{i=1}^{N} f_k(i)\xi_i$, where the $\{\xi_i\}$ variates are drawn from a p-stable distribution $(k = 1, 2)$. Assume that we have built $l = O(\frac{\log(1/\delta)}{\epsilon^2})$ iid*

pairs of p-stable sketches $\{X_1^j, X_2^j\}$ $(j = 1, \ldots, l)$*, and define*

$$X \;=\; \text{median}\{|X_1^1 - X_2^1|, \ldots, |X_1^l - X_2^l|\}.$$

Then, X lies within a relative error of at most ϵ *of the* L_p*-difference norm* $||f_1 - f_2||_p = [\sum_i |f_1(i) - f_2(i)|^p]^{1/p}$ *with probability* $\geq 1 - \delta$.

More recently, Cormode et al. [2002a] have also shown that, with small values of p (i.e., $p \to 0$), p-stable sketches can provide very effective estimates for the Hamming (i.e., L_0) norm (or, the number of distinct values) over continuous streams of updates.

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) multidimensional 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 in the stream; 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) (since we cannot explicitly store or backtrack to past stream items). 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 article by Indyk [2001]. 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 [Indyk 2001]. 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 methods discussed in Section 2.2) now become relevant. In this article, 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.

3.1 Technical Roadmap

The development of the technical material in this article is organized as follows. Section 4 describes our embedding algorithm for the tree-edit distance metric (termed TREEEMBED) in detail. In a nutshell, TREEEMBED constructs a hierarchical parsing of an input XML tree by iteratively *contracting edges* to produce successively smaller trees; our parsing makes repeated use of a recently proposed label-grouping procedure [Cormode and Muthukrishnan 2002] for contracting chains and leaf siblings in the tree. The bulk of Section 4 is devoted to proving the *small-time and low distance-distortion guarantees* of our TREEEMBED algorithm (Theorem 4.2). Then, in Section 5, we demonstrate how our embedding algorithm can be combined with the 1-stable sketching technique of Indyk [2000] to build a small sketch synopsis of a massive, streaming XML tree that can be used as a concise surrogate for the tree in approximate tree-edit distance computations. Most importantly, we show that the properties of our embedding allow us to parse the tree and build this sketch *in small space* and *in one pass*, as nodes of the tree are streaming by without ever backtracking on the data (Theorem 5.1). Finally, Section 6 shows how to combine our embedding algorithm with both 1-stable and AMS sketching in order to estimate (in limited space) the result size of an approximate tree-edit-distance similarity join over two continuous streams of XML documents (Theorem 6.1).

4. OUR TREE-EDIT DISTANCE EMBEDDING ALGORITHM

4.1 Definitions and Overview

In this section, we describe our embedding algorithm for the tree-edit distance metric (termed TREEEMBED) in detail, and prove its small-time and low distance-distortion guarantees. We start by introducing some necessary definitions and 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*) possibly ending in leaf node in T.

At a high level, our TREEEMBED algorithm produces a *hierarchical parsing* of T into a multiset $\mathcal{T}(T)$ of special valid subtrees by stepping through a number of *edge-contraction phases* producing successively smaller trees. A key component

of our solution (discussed later in this section) is the recently proposed deterministic coin tossing procedure of Cormode and Muthukrishnan [2002] for grouping symbols in a string—TREEEMBED employs that procedure repeatedly during each contraction phase to merge tree nodes in a chain as well as sibling leaf nodes. 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). Our analysis shows that the number of edge-contraction phases in T's parsing is $O(\log n)$, and that, even though the dimensionality of $V(T)$ is, in general, exponential in n, our construction guarantees that $V(T)$ is also *very sparse:* the total number of nonzero components in $V(T)$ is only $O(n)$. Furthermore, we demonstrate that our TREEEMBED algorithm runs in *near-linear*, that is, $O(n \log^* n)$ time. Finally, we prove the upper and lower bounds on the distance distortion guaranteed by our embedding scheme.

4.2 The Cormode-Muthukrishnan Grouping Procedure

Clearly, the technical crux lies in the details of our hierarchical parsing process for T that produces the valid-subtree multiset $\mathcal{T}(T)$. A basic element of our solution is the string-processing subroutine presented by Cormode and Muthukrishnan [2002] that uses deterministic coin tossing to find *landmarks* in an input string S, which are then used to split S into groups of two or three consecutive symbols. A landmark is essentially a symbol y (say, at location j) of the input string S with the following key property: if S is transformed into S' by an edit operation (say, a symbol insertion) at location l far away from j (i.e., $|l - j| >> 1$), then the Cormode-Muthukrishnan string-processing algorithm ensures that y is still designated as a landmark in S'. Due to space constraints, we do not give the details of their elegant landmark-based grouping technique (termed `CM-Group` in the remainder of this article) in our discussion—they can be found in Cormode and Muthukrishnan [2002]. Here, we only summarize a couple of the key properties of `CM-Group` that are required for the analysis of our embedding scheme in the following theorem.

THEOREM 4.1 [CORMODE AND MUTHUKRISHNAN 2002]. *Given a string of length k, the* `CM-Group` *procedure runs in time $O(k \log^* k)$. Furthermore, the closest landmark to any symbol x in the string is determined by at most $\log^* k + 5$ consecutive symbols to the left of x, and at most five consecutive symbols to the right of x.*

Intuitively, Theorem 4.1 states that, for any given symbol x in a string of length k, the group of (two or three) consecutive symbols chosen (by `CM-Group`) to include x depends only on the symbols lying in a radius of at most $\log^* k + 5$ to the left and right of x. Thus, a string-edit operation occurring outside this local neighborhood of symbol x is guaranteed not to affect the group formed containing x. As we will see, this property of the `CM-Group` procedure is crucial in proving the distance-distortion bounds for our TREEEMBED algorithm. Similarly, the $O(k \log^* k)$ complexity of `CM-Group` plays an important role in determining the running time of TREEEMBED.

4.3 The TreeEmbed Algorithm

As mentioned earlier, our TreeEmbed 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 the 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, by construction, our TreeEmbed algorithm guarantees that this subtree will be a *valid subtree* of T. Let $v(T)$ denote the valid subtree of T corresponding to node $v \in T^i$. Determining the node label for v uses 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; thus, the label of $v \in T^i$ is defined as the hash-function value $h(v(T))$. As we demonstrate in Section 7.1, such a valid-subtree-naming function can be computed in small space/time using an adaptation of the Karp-Rabin string fingerprinting algorithm [Karp and Rabin 1987]. Note that the existence of such an efficient naming function is crucial in guaranteeing the small space/time properties for our embedding algorithm since maintaining the exact valid subtrees $v(T)$ is infeasible; for example, near the end of our parsing, such subtrees are of size $O(|T|)$.[5]

The pseudocode description of our TreeEmbed embedding algorithm is depicted in Figure 3. 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 multiset $\mathcal{T}(T)$ 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 the hierarchical tree parsing in our embedding algorithm is depicted pictorially in Figure 4.

The L_1 distance between the vector images of two trees S and T is defined in the standard manner, that is, $\|V(T) - V(S)\|_1 = \sum_{x \in \mathcal{T}(T) \cup \mathcal{T}(S)} |V(T)[x] - V(S)[x]|$. 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.

[5] An implicit assumption made in our running-time analysis of TreeEmbed (which is also present in the complexity analysis of `CM-Group` in Cormode and Muthukrishnan [2002]—see Theorem 4.1) is that the fingerprints produced by the naming function $h()$ fit in *a single memory word* and, thus, can be manipulated in constant (i.e., $O(1)$) time. If that is not the case, then an additional multiplicative factor of $O(\log |T|)$ must be included in the running-time complexity to account for the length of such fingerprints (see Section 7.1).

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 (possibly ending in a leaf node)) **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 **s** of v) **do**
9. **if** ($|\mathbf{s}| \geq 2$) **then**
10. Consider **s** as a string and run `CM-Group` to divide **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 detailed definition in text
end

Fig. 3. Our tree-embedding algorithm.

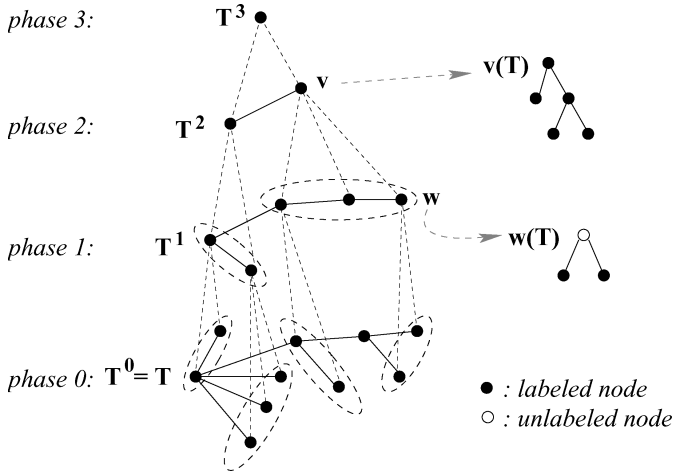


Fig. 4. Example of hierarchical tree parsing.

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* $O(|T|)$ *nonzero components. Finally, given two trees* S *and* T *with* $n = \max\{|S|, |T|\}$*, we have*

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

It is important to note here that, for certain special cases (i.e., when T is a simple chain or a "star"), our TREEEMBED algorithm essentially degrades to

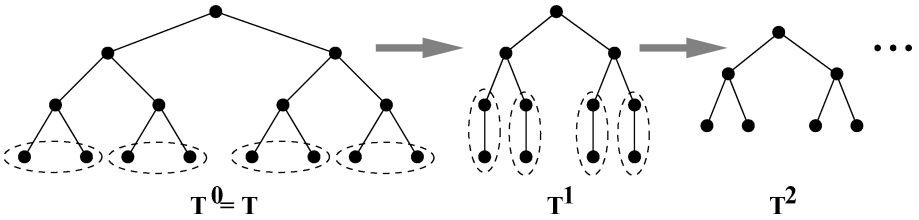


Fig. 5. Example of parsing steps for the special case of a full binary tree.

the string-edit distance embedding algorithm of Cormode and Muthukrishnan [2002]. This, of course, implies that, for such special cases, their even tighter $O(\log n \log^* n)$ bounds on the worst-case distance distortion are applicable. As another special-case example, Figure 5 depicts the initial steps in the parsing of a *full binary tree* T; note that, after two contraction phases, our parsing essentially reduces a full binary tree of depth h to one of depth $h-1$ (thus decreasing the size of the tree by a factor of about 1/2).

As a first step in the proof of Theorem 4.1, we demonstrate 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 of our embedding algorithm (Steps 3–16).

LEMMA 4.3. *The number of phases for our* TREEEMBED *algorithm on an input tree* T *is* $O(\log |T|)$.

PROOF. We partition the node set of T into several subsets as follows. First, define

$$A(T) = \{v \in T : v \text{ is a nonroot node with degree 2 (i.e., with only one child) } \textit{or } v \text{ is a leaf child of a non-root node of degree 2}\}, \text{ and}$$
$$B(T) = \{v \in T : v \text{ is a node of degree} \geq 3 \text{ (i.e., with at least two children) } \textit{or } v \text{ is the root node of } T\}.$$

Clearly, $A(T) \cup B(T)$ contains all internal (i.e., nonleaf) nodes of T; in particular, $A(T)$ contains all nodes appearing in (degree-2) chains in T (including potential leaf nodes at the end of such chains). Thus, the set of remaining nodes of T, say $L(T)$, comprises only leaf nodes of T which have at least one sibling or are children of the root. Let v be a leaf child of some node u, and let $\mathbf{s}_v$ denote the maximal contiguous set of leaf children of u which contains v. We further partition the leftover set of leaf nodes $L(T)$ as follows:

$$L_1(T) = \{v \in L(T) : |\mathbf{s}_v| \geq 2\},$$
$$L_2(T) = \{v \in L(T) : |\mathbf{s}_v| = 1 \text{ and } v \text{ is the leftmost such child of its parent}\}, \text{ and}$$
$$L_3(T) = L(T) - L_1(T) - L_2(T)$$
$$= \{v \in L(T) : |\mathbf{s}_v| = 1 \text{ and } v \textit{ is not} \text{ the leftmost such child of its parent}\}.$$

For notational convenience, we also use $A(T)$ to denote the set cardinality $|A(T)|$, and similarly for other sets. We first prove the following ancillary claim.

CLAIM 4.4. *For any rooted tree T with at least two nodes, $L_3(T) \leq L_2(T) + A(T)/2 - 1$.*

PROOF. We prove this claim by induction on the number of nodes in T. Suppose T has only two nodes. Then, clearly, $L_3(T) = 0$, $L_2(T) = 1$, and $A(T) = 0$. Thus, the claim is true for the base case.

Suppose the claim is true for all rooted trees with less than n nodes. Let T have n nodes and let r be the root of T. First, consider the case when r has only one child node (say, s), and let T' be the subtree rooted at s. By induction, $L_3(T') \leq L_2(T') + A(T')/2 - 1$. Clearly, $L_3(T) = L_3(T')$. Is $L_2(T)$ equal to $L_2(T')$? It is not hard to see that the only case when a node u can occur in $L_2(T')$ but not in $L_2(T)$ is when s has only one child, u, which also happens to be a leaf. In this case, obviously, $u \in L_2(T')$ (since it is the sole leaf child of the root), whereas in T u is the end-leaf of a chain node, so it is counted in $A(T)$ and, thus, $u \notin L_2(T)$. On the other hand, it is easy to see that both s and r are in $A(T) - A(T')$ in this case, so that $L_2(T) + A(T)/2 = L_2(T') + A(T')/2$. Thus, the claim is true in this case as well.

Now, consider the case when the root node r of T has at least two children. We construct several smaller subtrees, each of which is rooted at r (but contains only a subset of r's descendants). Let $u_1, \ldots, u_k$ be the leaf children of r such that $\mathbf{s}_{u_i} = \{u_i\}$ (i.e., have no leaf siblings); thus, by definition, $u_1 \in L_2(T)$, whereas $u_i \in L_3(T)$ for all $i = 2, \ldots, k$. We define the subtrees $T_1, \ldots, T_{k+1}$ as follows. For each $i = 1, \ldots, k+1$, T_i is the set of all descendants of r (including r itself) that lie to the right of leaf u_{i-1} and to the left of leaf u_i (as special cases, T_1 is the subtree to the left of u_1 and T_{k+1} is the subtree to the right of u_k). Note that T_1 and T_{k+1} my not contain any nodes (other than the root node r), but, by the definition of u_i's, all other T_i subtrees are guaranteed to contain at least one node other than r. Now, by induction, we have that

$$L_3(T_i) \leq L_2(T_i) + A(T_i)/2 - 1$$

for all subtrees T_i, except perhaps for T_1 and T_{k+1} (if they only comprise a sole root node, in which case, of course, the L_2, L_3, and A subsets above are all empty). Adding all these inequalities, we have

$$\sum_i L_3(T_i) \;\leq\; \sum_i L_2(T_i) + \sum_i A(T_i)/2 - (k-1), \tag{1}$$

where we only have $k-1$ on the right-hand side since T_1 and T_{k+1} may not contribute a -1 to this summation.

Now, it is easy to see that, if $u \in A(T_i)$, then $u \in A(T)$ as well; thus, $A(T) = \sum_i A(T_i)$. Suppose $u \in L_2(T_i)$, and let w denote the parent of u. Note that w cannot be the root node r of T, T_i. Indeed, suppose that $w = r$; then, since $u \in \{u_1, \ldots, u_k\}$, $\mathbf{s}_u$ contains a leaf node other than u which is also not in T_i (since $u \in L_2(T_i)$)). But then, it must be the case that u is adjacent to one of the leaves $u_1, \ldots, u_k$, which is impossible; thus, $w \neq r$ which, of course, implies that $u \in L_2(T)$ as well. Conversely, suppose that $u \in L_2(T)$; then, either $u = u_1$ or the parent of u is in one of the subtrees T_i. In the latter case, $u \in D_2(H_i)$. Thus, $L_2(H) = \sum_i L_2(T_i) + 1$.

Finally, we can argue in a similar manner that, for each $i = 1, \ldots, k+1$, $L_3(T_i) \subset L_3(T)$. Furthermore, if $u \in L_3(T)$, then either $u \in \{u_2, \ldots, u_k\}$ or $u \in L_3(T_i)$. Thus, $L_3(T) = \sum_i L_3(T_i) + k - 1$. Putting everything together, we have

$$\begin{aligned} L_3(T) &= \sum_i L_3(T_i) + k - 1 \\ &\leq \sum_i L_2(T_i) + \sum_i A(T_i)/2 \qquad \text{(by Inequality (1))} \\ &= L_2(T) + A(T)/2 - 1. \end{aligned}$$

This completes the inductive proof argument. □

With Claim 4.4 in place, we now proceed to show that the number of nodes in the tree goes down by a constant factor after each contraction phase of our parsing. Recall that T^i is the tree at the beginning of the $(i+1)$th phase, and let $L'(T^{i+1}) \subseteq L(T^{i+1})$ denote the subset of leaf nodes in $L(T^{i+1})$ that are created by contracting a chain in T^i. We claim that

$$B(T^{i+1}) \leq B(T^i) \quad \text{and} \quad B(T^{i+1}) + A(T^{i+1}) + L'(T^{i+1}) \leq B(T^i) + \frac{A(T^i)}{2}. \quad (2)$$

Indeed, it is easy to see that all nodes with degree at least three (i.e., $\geq$ two children) in T^{i+1} must have had degree at least three in T^i as well; this obviously proves the first inequality. Furthermore, note that any node in $B(T^{i+1})$ corresponds to a unique node in $B(T^i)$. Now, consider a node u in $A(T^{i+1}) \cup L'(T^{i+1})$. There are two possible cases depending on how node u is formed. In the first case, u is formed by collapsing some degree-2 (i.e., chain) nodes (and, possibly, a chain-terminating leaf) in $A(T^i)$—then, by virtue of the `CM-Group` procedure, u corresponds to at least two distinct nodes of $A(T^i)$. In the second case, there is a node $w \in B(T^i)$ and a leaf child of w that is collapsed into w to get u—then, u corresponds to a unique node of $B(T^i)$. The second inequality follows easily from the above discussion.

During the $(i+1)$th contraction phase, the number of leaves in $L_1(T^i)$ is clearly reduced by at least one-half (again, due to the properties of `CM-Group`). Furthermore, note that all leaves in $L_2(T^i)$ are merged into their parent nodes and, thus, disappear. Now, the leaves in $L_3(T^i)$ do not change; so, we need to bound the size of this leaf-node set. By Claim 4.4, we have that $L_3(T^i) \leq L_2(T^i) + A(T^i)/2$—adding $2 \cdot L_3(T^i)$ on both sides and multiplying across with 1/3, this inequality gives

$$L_3(T^i) \leq \frac{L_2(T^i)}{3} + \frac{2}{3} L_3(T^i) + \frac{A(T^i)}{6}.$$

Thus, the number of leaf nodes in $L(T^{i+1}) - L'(T^{i+1})$ can be upper-bounded as follows:

$$L(T^{i+1}) - L'(T^{i+1}) \leq \frac{L_1(T^i)}{2} + \frac{L_2(T^i)}{3} + \frac{2}{3} L_3(T^i) + \frac{A(T^i)}{6} \leq \frac{2}{3} L(T^i) + \frac{A(T^i)}{6}.$$

Combined with Inequality (2), this implies that the total number of nodes in T^{i+1} is

$$\begin{aligned} A(T^{i+1}) + B(T^{i+1}) + L(T^{i+1}) &\leq \frac{A(T^i)}{2} + B(T^i) + \frac{2}{3}L(T^i) + \frac{A(T^i)}{6} \\ &\leq B(T^i) + \frac{2}{3}(A(T^i) + L(T^i)). \end{aligned}$$

Now, observe that $B(T^i) \leq A(T^i) + L(T^i)$ (the number of nodes of degree more than two is at most the number of leaves in any tree)—the above inequality then gives

$$\begin{aligned} A(T^{i+1}) + B(T^{i+1}) + L(T^{i+1}) &\leq \frac{5}{6}B(T^i) + \frac{2}{3}(A(T^i) + L(T^i)) + \frac{1}{6}B(T^i) \\ &\leq \frac{5}{6}(A(T^i) + B(T^i) + L(T^i)). \end{aligned}$$

Thus, when going from tree T^i to T^{i+1}, the number of nodes goes down by a constant factor $\leq \frac{5}{6}$. This obviously implies that the number of parsing phases for our TreeEmbed algorithm is $O(\log |T|)$, and completes the proof. □

The proof of Lemma 4.3 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., Knuth [1973]); thus, by Lemma 4.3, 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 nonzero counts. Lemma 4.3, in conjunction with the fact that the `CM-Group` procedure runs in time $O(k \log^* k)$ for a string of size k (Theorem 4.1), also implies that our TreeEmbed algorithm runs in $O(|T| \log^* |T|)$ time on input T. The following two subsections establish the distance-distortion bounds stated in Theorem 4.1.

An immediate implication of the above results is that we can use our embedding algorithm to compute the approximate (to within a guaranteed $O(\log^2 n \log^* n)$ 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 is significantly higher: conventional tree-edit distance (without subtree moves) is solvable in $O(|T||S|d_T d_S)$ time (where, d_T (d_S) is the depth of T (respectively, S)) [Apostolico and Galil 1997; Zhang and Shasha 1989], whereas in the presence of subtree moves the problem becomes $\mathcal{NP}$-hard even for the simple case of flat strings [Shapira and Storer 2002].

4.4 Upper-Bound Proof

Suppose we are given a tree T with n nodes and let Δ denote the quantity $\log^* n + 5$. As a first step in our proof, we demonstrate that showing the upper-bound result in Theorem 4.2 can actually be reduced to a simpler problem, namely, that of bounding the L_1 distance between the vector image of T and the vector image of a 2-tree forest created when removing a valid subtree from

T. More formally, consider a (valid) subtree of T of the form $T'[v, \mathbf{s}]$ for some contiguous subset 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, and let T_2 denote the resulting subtree; furthermore, let T_1 denote the deleted subtree $T'[v, \mathbf{s}]$. Thus, we have broken T into a 2-tree forest comprising $T_1 = T'[v, \mathbf{s}]$ and $T_2 = T - T_1$ (see the leftmost portion of Figure 8 for an example).

We now compare the following two vectors. The first vector $V(T)$ is obtained by applying our TreeEmbed parsing procedure to T. For the second vector, we apply TreeEmbed to each of the trees T_1 and T_2 individually, and then add the corresponding vectors $V(T_1)$ and $V(T_2)$ component-wise—call this vector $V(T_1 + T_2) = V(T_1) + V(T_2)$. (Throughout this section, we use $(T_1 + T_2)$ to denote the 2-tree forest composed of T_1 and T_2.) Our goal is to prove the following theorem.

THEOREM 4.5. *The L_1 distance between vectors $V(T)$ and $V(T_1 + T_2)$ is at most $O(\log^2 n \log^* n)$.*

Let us first see how this result directly implies the upper bound stated in Theorem 4.2.

PROOF OF THE UPPER BOUND IN THEOREM 4.2. It is sufficient to consider the case when the tree-edit distance between S and T is 1 and show that, in this case, the L_1 distance between $V(S)$ and $V(T)$ is $\leq O(\log^2 n \log^* n)$. First, assume that T is obtained from S by deleting a leaf node v. Let the parent of v be w. Define $\mathbf{s} = \{v\}$, and delete $S'[w, \mathbf{s}]$ from S. This splits S into T and $S'[w, \mathbf{s}]$—call this S_1. Theorem 4.5 then implies that $\|V(S) - V(T + S_1)\|_1 = \|V(S) - (V(T) + V(S_1))\|_1 \leq O(\log^2 n \log^* n)$. But, it is easy to see that the vector $V(S_1)$ only has three nonzero components, all equal to 1; this is since S_1 is basically a 2-node tree that is reduced to a single node after one contraction phase of TreeEmbed. Thus, $\|V(S_1)\|_1 = \|(V(T) + V(S_1)) - V(T)\|_1 \leq 3$. Then, a simple application of the triangle inequality for the L_1 norm gives $\|V(S) - V(T)\|_1 \leq O(\log^2 n \log^* n)$. Note that, since insertion of a leaf node is the inverse of a leaf-node deletion, the same holds for this case as well.

Now, let v be a node in S and $\mathbf{s}$ be a contiguous set of children of v. Suppose T is obtained from S by moving the subtree $S'[v, \mathbf{s}]$, that is, deleting this subtree and making it a child of another node x in S.[6] Let S_1 denote $S'[v, \mathbf{s}]$, and let S_2 denote the tree obtained by deleting S_1 from S. Theorem 4.5 implies that $\|V(S) - V(S_1 + S_2)\|_1 \leq O(\log^2 n \log^* n)$. Note, however, that we can also picture $(S_1 + S_2)$ as the forest obtained by deleting S_1 from T. Thus, $\|V(T) - V(S_1 + S_2)\|_1$ is also $\leq O(\log^2 n \log^* n)$. Once again, the triangle inequality for L_1 easily implies the result.

Finally, suppose we delete a nonleaf 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. At this point, v is a leaf node, so we are just deleting a leaf node now. Thus,

[6]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.

the result for this case follows easily from the arguments above for deleting a leaf node and moving a subtree. □

As a consequence, it is sufficient to prove Theorem 4.5. Our proof proceeds along the following lines. We define an *influence region* for each tree T^i in our hierarchical parsing ($i = 0, \ldots, O(\log n)$)—the intuition here is that the influence region for T^i captures the complete set of nodes in T^i *whose parsing could have been affected* by the change (i.e., the splitting of T into $(T_1 + T_2)$). Initially (i.e., tree T^0), this region is just the node v at which we deleted the T_1 subtree. But, obviously, this region grows as we proceed to subsequent phases in our parsing. We then argue that, if we ignore this influence region in T^i and the corresponding region in the parsing of the (T_1+T_2) forest, then the resulting sets of valid subtrees look very similar (in any phase i). Thus, if we can *bound the rate at which this influence region grows* during our hierarchical parsing, we can also bound the L_1 distance between the two resulting characteristic vectors. The key intuition behind bounding the size of the influence region is as follows: when we effect a change at some node v of T, nodes far away from v in the tree remain *unaffected*, in the sense that the subtree in which such nodes are grouped during the next phase of our hierarchical parsing remains unchanged. As we will see, this fact hinges on the properties of the `CM-Group` procedure used for grouping nodes during each phase of TreeEmbed (Theorem 4.1). The discussion of our proof in the remainder of this section is structured as follows. First, we formally define influence regions, giving the set of rules for "growing" such regions of nodes across consecutive phases of our parsing. Second, we demonstrate that, for any parsing phase i, if we ignore the influence regions in the current (i.e., phase-$(i + 1)$) trees produced by TreeEmbed on input T and $(T_1 + T_2)$, then we can find a *one-to-one, onto mapping* between the nodes in the remaining portions of the current T and $(T_1 + T_2)$ that pairs up identical valid subtrees. Third, we bound the size of the influence region during each phase of our parsing. Finally, we show that the upper bound on the L_1 distance of $V(T)$ and $V(T_1 + T_2)$ follows as a direct consequence of the above facts.

We now proceed with the proof of Theorem 4.5. Define $(T_1 + T_2)^i$ as the 2-tree forest corresponding to $(T_1 + T_2)$ at the beginning of the $(i + 1)$th parsing phase. 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 are merged to form x contains x'. As earlier, any node w in T^i corresponds to a valid subtree $w(T)$ of T; furthermore, it is easy to see that if w and w' are two distinct nodes of T^i, then the $w(T)$ and $w'(T)$ subtrees are disjoint. (The same obviously holds for the parsing of each of T_1, T_2.)

For each tree T^i, we *mark* certain nodes; intuitively, this node-marking defines the *influence region* of T^i mentioned above. Let M^i be the set of marked nodes (i.e., influence region) in T^i (see Figure 6(a) for an example). The generic structure of the influence region M^i satisfies the following: (1) M^i is a *connected subtree* of T^i that always contains the node v (at which the T_1 subtree was removed), that is, the node in T^i which contains v (denoted by v^i) is always in M^i; (2) there is a *center node* $c^i \in M^i$, and M^i may contain some ancestor nodes of c^i—but all such ancestors (except perhaps for c^i itself) must be of degree 2 only, and should form a connected path; and (3) M^i may also contain some

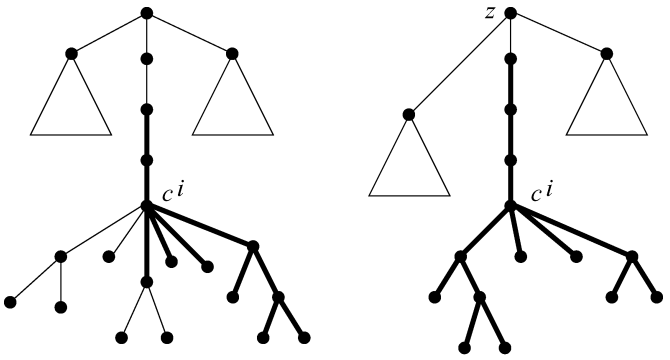


Fig. 6. (a) The subtree induced by the bold edges corresponds to the nodes in M^i. (b) Node z becomes the center of N^i.

descendants of the center node c^i. Finally, certain (unmarked) nodes in $T^i - M^i$ are identified as *corner nodes*—intuitively, these are nodes whose parsing will be affected when they are shrunk down to a leaf node.

Once again, the key idea is that the influence region M^i captures the set of those nodes in T^i whose parsing in TREEEMBED *may have been affected* by the change we made at node v. Now, in the next phase, the changes in M^i can potentially affect some more nodes. Thus, we now try to determine which nodes M^i can affect; that is, assuming the change at v has influenced all nodes in M^i, which are the nodes in T^i whose parsing (during phase $(i+1)$) can change as a result of this. To capture this newly affected set of nodes, we define an *extended influence region* N^i in T^i—this intuitively corresponds to the (worst-case) subset of nodes in T^i whose parsing can potentially be affected by the changes in M^i.

First, add all nodes in M^i to N^i. We define the *center node* z of the extended influence region N^i as follows. We say that a descendant node u of v^i (which contains v) in T^i is a *removed descendant of* v^i if and only if its corresponding subtree $u(T)$ in the base tree T is entirely contained within the removed subtree $T[v, \mathbf{s}]$. (Note that, initially, $v^0 = v$ is trivially a removed descendant of v^0.) Now, let w be the highest node in M^i—clearly, w is an ancestor of the current center node c^i as well as the v^i node in T^i. If all the descendants of w are either in M^i or are removed descendants of v^i, then define the center z to be the parent of node w, and add z to N^i (see Figure 6(b)); otherwise, define the center z of N^i to be same as c^i. The idea here is that the grouping of w's parent in the next phase can change only if the entire subtree under w has been affected by the removal of the $T'[v, \mathbf{s}]$ subtree. Otherwise, if there exist nodes under w in T^i whose parsing remains unchanged and that have not been deleted by the subtree removal, then the mere existence of these nodes in T^i means that it is impossible for TREEEMBED to group w's parent in a different manner during the next phase of the $(T_1 + T_2)$ parsing in any case. Once the center node z of N^i has been fixed, we also add nodes to N^i according to the following set of rules (see Figures 7(a) and (b) for examples).

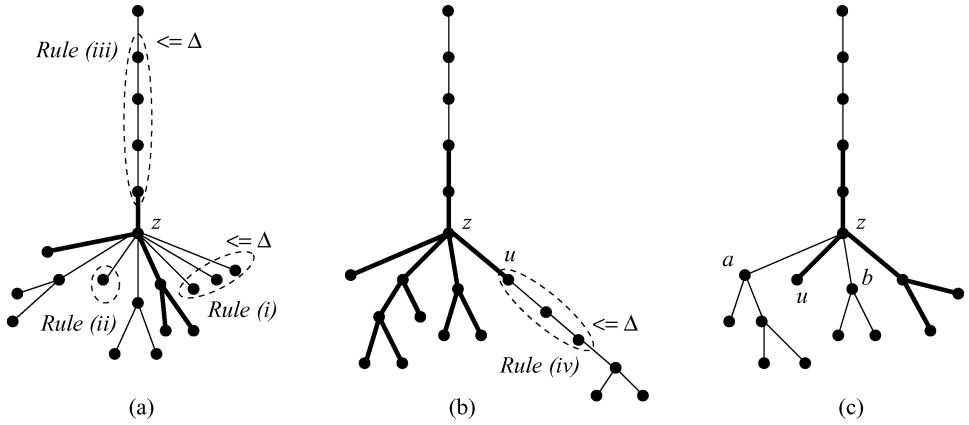


Fig. 7. (a) The nodes in dotted circles get added to N^i due to Rules (i), (ii), and (iii). (b) The nodes in the dotted circle get added to N^i due to Rule (iv)—note that all descendants of the center z which are not descendants of u are in M^i. (c) Node u moves up to z, turning nodes a and b into corner nodes.

(i) Suppose u is a leaf child of the (new) center z or the v^i node in T^i; furthermore, assume there is some sibling u' of u 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 the center z which is not already in M^i (if such a child exists); then, add u to N^i. Similarly, for the v^i node in T^i, let u be a leaf child of v^i such that one of the following conditions is satisfied: (a) u is the leftmost lone leaf child of v^i when considering only the removed descendants of v^i; or (b) u is the leftmost lone leaf child of v^i when ignoring all removed descendants of v^i. Then, add u to N^i.

(iii) Let w be the highest node in $M^i \cup \{z\}$ (so it is an ancestor of the center node z). 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 the center z or the v^i node in T^i such that one of the following conditions is satisfied: (a) u is not a removed descendant of v^i and all descendants of all siblings of u (other than u itself) are either already in M^i or are removed descendants of v^i; or (b) u is a removed descendant of v^i (and, hence, a child of v^i) and all removed descendants of v^i which are not descendants of u are in M^i. Then, 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 end-points), and has length at most Δ, then add u'' to N^i.

Let us briefly describe why we need these four rules. We basically want to make sure that we include all those nodes in N^i whose parsing can potentially

be affected if we delete or modify the nodes in M^i (given, of course, the removal of the $T'[v, \mathbf{s}]$ subtree). The first three rules, in conjunction with the properties of our TREEEMBED parsing, are easily seen to capture this fact. The last rule is a little more subtle. Suppose u is a child of z (so that we are in clause (a) of Rule (iv)); furthermore, assume that all descendants of z except perhaps those of u are either already in M^i or have been deleted with the removal of $T'[v, \mathbf{s}]$. Remember that all nodes in M^i have been modified due to the change effected at v, so they may not be present at all in the corresponding picture for $(T_1 + T_2)$ (i.e., the $(T_1+T_2)^i$ forest). But, if we just *ignore* M^i and the removed descendants of v^i, then z becomes a node of degree 2 only, which would obviously affect how u and its degree-2 descendants are parsed in $(T_1+T_2)^i$ (compared to their parsing in T^i). Rule (iv) is designed to capture exactly such scenarios; in particular, note that clauses (a) and (b) in the rule are meant to capture the potential creation of such degree-2 chains in the remainder subtree T_2^i and the deleted subtree T_1^i, respectively.

We now consider the rule for marking *corner nodes* in T^i. Once again, the intuition is that certain (unaffected) nodes in $T^i - M^i$ (actually, in $T^i - N^i$) are marked as corner nodes so that we can "remember" that their parsing will be affected when they are shrunk down to a leaf. Suppose the center node z has at least two children, and a leftmost lone leaf child u—note that, by Rule (ii), $u \in N^i$. If any of the two immediate siblings of u are not in N^i, then we mark them as corner nodes (see Figure 7(c)). The key observation here is that, when parsing T^i, u is going to be merged into z and disappear; however, we need to somehow "remember" that a (potentially) affected node u was there, since its existence could affect the parsing of its sibling nodes when they are shrunk down to leaves. Marking u's immediate siblings in T^i as corner nodes essentially achieves this effect.

Having described the (worst-case) extended influence region N^i in T^i, let us now define M^{i+1}, that is, the influence region at the next level of our parsing of T. M^{i+1} is precisely 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 node z of N^i; furthermore, any node in T^{i+1} which contains a corner node is again marked as a corner node.

Initially, define $M^0 = \{v\}$ (and, obviously, $v^0 = c^0 = v$). Furthermore, if v has a child node immediately on the left (right) of the removed child subsequence $\mathbf{s}$, then that node as well as the leftmost (respectively, rightmost) node in $\mathbf{s}$ are marked as corner nodes. The reason, of course, is that these ≤ 4 nodes may be parsed in a different manner when they are shrunk down to leaves during the parsing of T_1 and T_2. Based on the above set of rules, it is easy to see that M^i and N^i are always connected subtrees of T^i. It is also important to note that the extended influence region N^i is defined in such a manner that the parsing of all nodes in $T^i - N^i$ cannot be affected by the changes in M^i. This fact should become clear as we proceed with the details of the proofs in the remainder of this section.

Example 4.6. Figure 8 depicts the first three phases of a simple example parsing for T and $(T_1 + T_2)$, in the case of a 4-level full binary tree T that

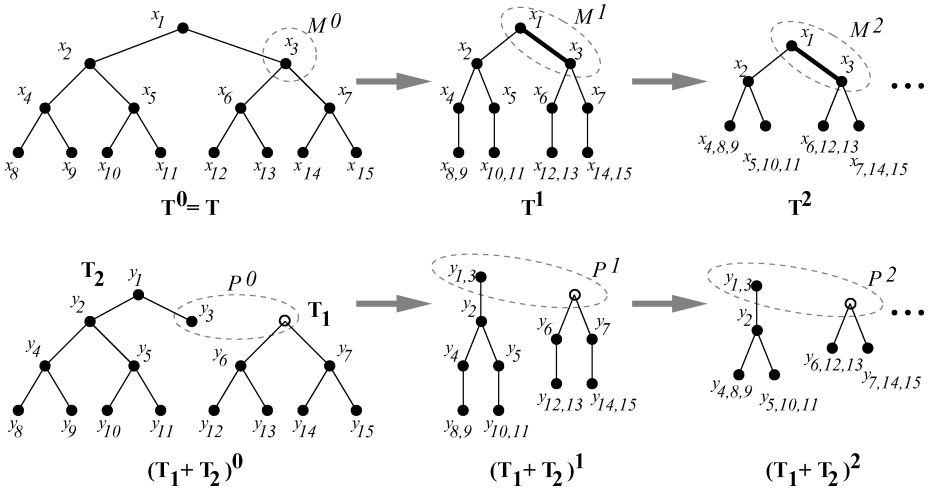


Fig. 8. Example of TREEEMBED parsing phases for T and $(T_1 + T_2)$ in the case of a full binary tree, highlighting the influence regions M^i in T^i and the corresponding P^i regions in $(T_1 + T_2)^i$ ("**o**" denotes an unlabeled node).

is split by removing the right subtree of the root (i.e., $T_1 = T'[x_3, \{x_6, x_7\}]$, $T_2 = T - T_1$). We use subscripted x's and y's to label the nodes in T^i and $(T_1 + T_2)^i$ to emphasize the fact that these tree nodes are parsed independently by TREEEMBED; furthermore, we employ the subscripts to capture the original subtrees of T and $(T_1 + T_2)$ represented by nodes in later phases of our parsing. Of course, it should be clear that x and y nodes with identical subscripts refer to *identical (valid) subtrees* of the original tree T; for instance, both $x_{4,8,9} \in T^2$ and $y_{4,8,9} \in T_2^2$ represent the same subtree $T[x_4, \{x_8, x_9\}] = \{x_4, x_8, x_9\}$ of T.

As depicted in Figure 8, the initial influence region of T is simply $M^0 = \{x_3\}$ (with $v^0 = c^0 = x_3$). Since, clearly, all descendants of x_3 are removed descendants of v_0, the center z for the extended influence region N^0 moves up to the parent node x_1 of x_3 (and none of our other rules are applicable); thus, $N^0 = \{x_1, x_3\}$ and, obviously, $M^1 = \{x_1, x_3\}$. This is crucial since (as shown in Figure 8), due to the removal of T_1, nodes y_1 and y_3 are processed in a very different manner in the remainder subtree T_2^0 (i.e., y_3 is merged up into y_1 as its leftmost lone leaf child). Now, for T^1, none of our rules for extending the influence region apply and, consequently, $N^1 = M^2 = \{x_1, x_3\}$. The key thing to note here is that, for each parsing phase i, ignoring the nodes in the influence region M^i (and the "corresponding" nodes in $(T_1 + T_2)^i$), the remaining nodes of T^i and $(T_1 + T_2)^i$ have been parsed in an *identical* manner by TREEEMBED (and correspond to an identical subset of valid subtrees in T); in other words, their corresponding characteristic vectors in our embedding are *exactly the same*. We now proceed to formalize these observations.

Given the influence region M^i of T^i, we define a corresponding node set, P^i, in the $(T_1 + T_2)^i$ forest. In what follows, we prove 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* valid subtrees in T and $(T_1 + T_2)$,

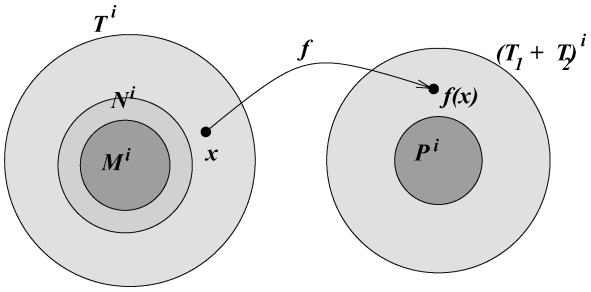


Fig. 9. f maps from $T^i - M^i$ to $(T_1 + T_2)^i - P^i$.

respectively. The node set P^i in $(T_1 + T_2)^i$ is defined as follows (see Figure 8 for examples). P^i always contains the root node of T_1^i. Furthermore, a node $u \in (T_1 + T_2)^i$ is in P^i, if and only if there exists a node $u' \in M^i$ such that the intersection $u(T_1 + T_2) \cap u'(T)$ is nonempty (as expected, $u(T_1 + T_2)$ denotes the valid subtree corresponding to u in $(T_1 + T_2)$). We demonstrate that our solution always maintains the following invariant.

INVARIANT 4.7. *Given any node $x \in T^i - M^i$, there exists a node $y = f(x)$ in $(T_1 + T_2)^i - P^i$ such that $x(T)$ and $y(T_1 + T_2)$ are* identical *valid subtrees on the exact same subset of nodes in the original tree T. Conversely, given a node $y \in (T_1 + T_2)^i - P^i$, there exists a node $x \in T^i - M^i$ such that $x(T) = y(T_1 + T_2)$.*

Thus, there always exists a *one-to-one, onto mapping* f from $T^i - M^i$ to $(T_1 + T_2)^i - P^i$ (Figure 9). 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 of valid subtrees in this phase are *identical*.

Example 4.8. Continuing with our binary-tree parsing example in Figure 8, it is easy to see that, in this case, the mapping $f : T^i - M^i \longrightarrow (T_1 + T_2)^i - P^i$ simply maps every x node in $T^i - M^i$ to the y node in $(T_1 + T_2)^i - P^i$ with the same subscript that, obviously, corresponds to exactly the same valid subtree of T; for instance, $y_{10,11} = f(x_{10,11})$ and both nodes correspond to the same valid subtree $T'[x_5, \{x_{10}, x_{11}\}]$. Thus, the collections of valid subtrees for $T^i - M^i$ and $(T_1 + T_2)^i - P^i$ are identical (i.e., the L_1 distance of their corresponding characteristic vectors is zero); this implies that, for example, the contribution of T^1 and $(T_1 + T_2)^1$ to the difference of the embedding vectors $V(T)$ and $V(T_1 + T_2)$ is upper-bounded by $|M^1| = 2$.

Clearly, Invariant 4.7 is true in the beginning (i.e., $M^0 = \{v\}$, $P^0 = \{v, \text{root}(T_1)\}$). Suppose our invariant remains true for T^i and $(T_1 + T_2)^i$. We now need to prove it for T^{i+1} and $(T_1 + T_2)^{i+1}$. As previously, let $N^i \supseteq M^i$ be the extended influence region in T^i. 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 node $q \in T^{i+1}$ and w' is contained in node $q' \in (T_1 + T_2)^{i+1}$.

LEMMA 4.9. *Given a node w in $T^i - N^i$, let q, q' be as defined above. If $q(T)$ and $q'(T_1 + T_2)$ are identical subtrees for any node $w \in T^i - N^i$, then Invariant 4.7 holds for T^{i+1} and $(T_1 + T_2)^{i+1}$ as well.*

PROOF. We have to demonstrate 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 would be in M^{i+1}. Let $y' = f(x')$, and 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 subtrees. 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 $z \in T^{i+1}, z \neq x$. By the definition of the P^i node sets, since $x \notin M^{i+1}$, we have that $y \in (T_1 + T_2)^{i+1} - P^{i+1}$.

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 (by definition) there exists 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 nonempty 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, for any pair of nodes $w \in T^i - N^i$, $w' = f(w) \in (T_1 + T_2)^i - P^i$, the corresponding encompassing nodes $q \in T^{i+1}$ and $q' \in (T_1 + T_2)^{i+1}$ map to identical valid subtrees, that is, $q(T) = q'(T_1 + T_2)$. This is what we seek to do next. Our proof uses a detailed, 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 forest $(T_1 + T_2)^i$. In the interest of space and continuity, we defer the details of this proof to the Appendix.

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 guaranteed to be a subset of M^i. Our next step is to bound the size of M^i.

LEMMA 4.10. *The influence region M^i for tree T^i consists of at most $O(i \log^* n)$ nodes.*

PROOF. Note that, during each parsing phase, Rule (iii) adds at most Δ nodes of degree at most 2 to the extended influence region N^i. It is not difficult to see that Rule (iv) also adds at most 4Δ nodes of degree at most 2 to N^i during each phase; indeed, note that, for instance, there is *at most one* child node u of z which is not in M^i and satisfies one of the clauses of Rule (iv). So, adding over the first i stages of our algorithm the number of such nodes in M^i can be at most $O(i \log^* n)$. Thus, we only need to bound the number of nodes that get added to the influence region due to Rules (i) and (ii).

We now want to count the number of leaf children of the center node c^i which are in M^i. Let k_i be the number of children of c^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 $1 + \sum_{j=1}^{i-1} k_j$ 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, where $|C_1^i| \leq 4\Delta$. We may add up to Δ consecutive leaf children of c^i on either side of C_1^i to the extended influence region N^i. Thus, this set may grow to a size of 6Δ contiguous leaves. But when we parse this set (using `CM-Group`), we reduce its size by at least half. Thus, this set will now contain at most 3Δ leaves (which is at most 4Δ). Therefore, each of the $1 + \sum_{j=1}^{i-1} k_j$ contiguous sets in C^i correspond to a contiguous set in T^{i+1} of size at most 4Δ.

Now, we may add other leaf children of c^i to N^i. This can happen only if a corner node becomes a leaf. In this case, at most Δ consecutive leaves on either side of this node are added to N^i (by Rule (i)); thus, we may add k_i more such sets of consecutive leaves to N^i. This completes our inductive argument.

But note that, in any phase, at most two new corner nodes (i.e., the immediate siblings of the center node's leftmost lone leaf child) can be added. (And, of course, we also start out with at most four nodes marked as corners inside and next to the removed child subsequence $\mathbf{s}$.) So, $\sum_{j=1}^{i} k_j \leq 2i+2$. This shows that the number of nodes in C^i is $O(i \log^* n)$. The contribution toward M^i of the leaf children of the v^i node can also be upper bounded by $O(i \log^* n)$ using a very similar argument. This completes the proof. □

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 one-to-one, onto mapping 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 $w \in (T_1 + T_2)^i - Q^i$. Furthermore, we can show in a similar manner that $|Q^i| \leq O(i \log^* n)$.

We are now ready to complete the proof of Theorem 4.5.

PROOF OF THEOREM 4.5. 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 one-to-one, $V_1 - M^i$ 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 summation is over all subtrees t. Adding over all parsing phases i, we have

$$\|V(T) - V(T_1 + T_2)\|_1 \;\leq\; \sum_{i=1}^{O(\log n)} O(i \log^* n) \;=\; O(\log^2 n \log^* n).$$

This completes our proof argument. □

4.5 Lower-Bound Proof

Our proof follows along the lower-bound proof of Cormode and Muthukrishnan [2002], 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 Cormode and Muthukrishnan [2002], 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 4.11 with $k = 3$.[7])

LEMMA 4.11. *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 Cormode and Muthukrishnan [2002], 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

[7]It is probably worth noting at this point that the subtree-move operation is needed only to establish the distortion lower-bound result in this section; that is, the upper bound shown in Section 4.1 holds for the *standard* tree-edit distance metric as well.

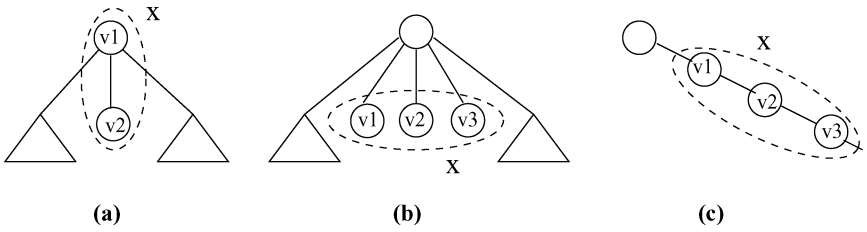


Fig. 10. Forming a level-i node x.

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 end 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.

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 Figures 10(a)–10(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 10(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) and (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, pseudorandom *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. (Note, for example, that this is exactly the ordering of XML elements produced by the event-based SAX parsing interface (`sax.sourceforge.net/`).) 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 1-stable sketching algorithms of Indyk [2000] (see Theorem 2.2).

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* $f()$, $f(\texttt{sketch}(V(S)), \texttt{sketch}(V(T)))$ *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 4.4, 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 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 (Theorem 2.2). The details of the proof for Theorem 5.1 follow easily from the above discussion and the results of Indyk [2000].

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 from 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 $T \in \mathcal{S}_i$ $(i = 1, 2)$ to construct a (low-distortion) image $V(T)$ of T as a point in an appropriate multidimensional vector space. We then obtain a lower-dimensional vector of 1-stable sketches of $V(T)$ that approximately preserves L_1 distances in the original vector space, as described by Indyk [2000]. 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 AMS sketching over the stream of points in the embedding L_1 vector space in order to build a randomized, sketch-based estimate for $|\texttt{SimJoin}(\mathcal{S}_1, \mathcal{S}_2, \tau)|$.[8] 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 Alon et al. [1996, 1999]; Dobra et al. [2002]; see also Theorem 2.1).

THEOREM 6.1. *Let* $|\texttt{SimJoin}(\mathcal{S}_1, \mathcal{S}_2, \tau)|$ *denote 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. Define* $k = k(\delta, \epsilon) = O(\frac{\log(1/\delta)}{\epsilon})^{O(1/\epsilon)}$*. An atomic, sketch-based estimate for* $|\texttt{SimJoin}(\mathcal{S}_1, \mathcal{S}_2, \tau)|$ *can be constructed in* $O(b + k(\delta, \epsilon) \log N)$ *space and* $O(b \log^* b + k(\delta, \epsilon) \log N)$ *time per document, where* δ, ϵ *are constants* < 1 *that control the accuracy of the distance estimates and* N *denotes the length of the input stream(s).*

[8] Assuming constant-sized trees, a straightforward approach to our similarity-join problem would be to exhaustively build all trees within a τ-radius of an incoming tree, and then just sketch (the fingerprints of) these trees directly using AMS for the similarity-join estimate. The key problem with such a "direct" approach is the computational cost per incoming tree: given a tree T with b nodes and an edit-distance radius of τ, the cost of the brute-force enumeration of all trees in the τ-neighborhood of T would be at least $O(b^\tau)$, which is probably prohibitive (except for very small values of b and τ).

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 [2000] (i.e., with 1-stable (Cauchy) random variates) to map T to a lower-dimensional vector of $O(k(\delta,\epsilon))$ iid sketching values $\texttt{sketch}(V(T))$. This mapping of an input tree T to a point in an $O(k(\delta,\epsilon))$-dimensional vector space can be done in space $O(b+k(\delta,\epsilon)\log N)$: this covers the $O(b)$ space to store and parse the tree,[9] and the $O(\log N)$ space required to generate the 1-stable random variates for each of the $O(k(\delta,\epsilon))$ 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 $O(Nb)$ nonzero 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 $O(k(\delta,\epsilon))$-dimensional vector space with *constant probability*, as stated in the following theorem from Indyk [2000].

THEOREM 6.2 (INDYK 2000). *Let f_1 and f_2 denote N dimensional numeric vectors rendered as a stream of updates, and let $\{X_1^j, X_2^j : j = 1,\ldots,k\}$ denote $k = k(\delta,\epsilon) = O(\frac{\log(1/\delta)}{\epsilon})^{O(1/\epsilon)}$ iid pairs of 1-stable sketches $X_l^j = \sum_{i=1}^{N} f_l(i)\xi_i^j$; also, define $\overline{X}_l$ as the k-dimensional vector $(X_l^1,\ldots,X_l^k)$ ($l = 1,2$; $\{\xi_i^j\}$ are 1-stable (Cauchy) random variates). Then, the L_1-difference norm of the k-dimensional sketch vectors $\|\overline{X}_1 - \overline{X}_2\|_1$ satisfies*

(1) $\|\overline{X}_1 - \overline{X}_2\|_1 \geq \|f_1 - f_2\|_1$ *with probability* $\geq 1-\delta$; *and*
(2) $\|\overline{X}_1 - \overline{X}_2\|_1 \leq (1+\epsilon)\cdot\|f_1 - f_2\|_1$ *with probability* $\geq \epsilon$.

Intuitively, Theorem 6.2 states that, if we use 1-stable sketches as a *dimensionality-reduction* tool for L_1 (that is, for mapping a point in a high, O(N)-dimensional L_1-normed space to a lower, k-dimensional L_1-normed space, instead of using median selection as in Theorem 2.2), then we can only provide weaker, asymmetric guarantees on the L_1 distance distortion. In short, we can guarantee small distance contraction with high probability (i.e., $1-\delta$), but we can guarantee small distance expansion only with *constant probability* (i.e., ϵ). (Note that the exact manner in which the δ,ϵ parameters control the error and confidence in the approximate L_1-distance estimates is formally stated in Theorem 6.2.) The reason for using this version of Indyk's results in our similarity-join scenario is that, as mentioned earlier, we need to perform an (approximate) streaming similarity-join computation over the mapped space of sketch vectors, which appears to be infeasible when the median-selection operator is used.

The second level of sketching in our construction will produce a pseudorandom AMS sketch (Section 2.2) 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 AMS-sketching technique discussed in Section 2.2 needs

[9]Of course, for large trees, the small-space optimizations of Section 5 can be used (assuming pre-order node arrivals).

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 multidimensional hypercube) of radius τ centered around $\texttt{sketch}(V(T))$ in the embedding $O(k(\delta,\epsilon))$-dimensional vector space when building an AMS-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 $\xi_{\bar{i}}$ (where, the index $\bar{i} = \texttt{sketch}(V(T))$) to the atomic AMS-sketch estimate $X_{\mathcal{S}_1}$ for S_1, we update $X_{\mathcal{S}_1}$ by adding $\sum_{\bar{j}\in n(\bar{i},\tau)} \xi_{\bar{j}}$, where $n(\bar{i},\tau)$ denotes the L_1 neighborhood of radius τ of $\bar{i} = \texttt{sketch}(V(T))$ in the embedding vector space (i.e., $n(\bar{i},\tau) = \{\bar{j} : \|\bar{i} - \bar{j}\|_1 \leq \tau\}$). Note that this special processing is only carried out on the $\mathcal{S}_1$ stream; the AMS-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 show (see Section 2.2) 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 $O(k(\delta,\epsilon))$-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 (1-stable) sketching can be done in small time using the techniques discussed by Indyk [2000]. The second level of (AMS) sketching can also be implemented using standard AMS-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 $O(k(\delta,\epsilon))$-dimensional vector space. Thus, a naive, brute force technique that simply iterates over all these variates would increase the per-document sketching cost by a multiplicative factor of $O(|n(\bar{i},\tau)|) = O(\tau^{k(\delta,\epsilon)}) \approx O((1/\delta)^{k(\delta,\epsilon)})$ in the worst case; however, *efficiently range-summable* sketching variates, as in Feigenbaum et al. [1999], can be used to reduce this multiplicative factor to only $O(\log|n(\bar{i},\tau)|) = O(k(\delta,\epsilon))$. □

Again, note that, by Indyk's L_1dimensionality-reduction result (Theorem 6.2), Theorem 6.1 only guarantees that our estimation algorithm approximates tree-edit distances with *constant probability*. In other words, this means that a constant fraction of the points in the τ-neighborhood of a given point could be missed. Furthermore, the very recent results of Charikar and Sahai [2002] 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 [Johnson and Lindenstrauss 1984] 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 $O(b)$ nonzero components, and then use standard, high-probability L_2 dimensionality reduction (e.g., through the JL construction). 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 6.3. *The tree-edit distances for the estimation of the similarity-join cardinality* $|\texttt{SimJoin}(\mathcal{S}_1, \mathcal{S}_2, \tau)|$ *in Theorem 6.1 can be estimated with high probability to within a factor of* $O(\sqrt{b}\log^2 b \log^* b)$.

7. EXPERIMENTAL STUDY

In this section, we present the results of an empirical study that we have conducted using the oblivious tree-edit distance embedding algorithm developed in this article. Several earlier studies have verified (both analytically and experimentally) the effectiveness of the pseudorandom sketching techniques employed in Sections 5 and 6 in approximating join cardinalities and different vector norms; see, for example, Alon et al. [1996, 1999]; Cormode et al. [2002a, 2002b]; Dobra et al. [2002]; Gilbert et al. [2002a]; Indyk et al. [2000], Indyk [2000]; Thaper et al. [2002]. Thus, the primary focus of our experimental study here is to quantify the average-case behavior of our embedding algorithm (TREEEMBED) in terms of the observed tree-edit distance distortion on realistic (both synthetic and real-life) XML data trees. As our findings demonstrate, the average-case behavior of our TREEEMBED algorithm is indeed significantly better than that predicted by the theoretical (worst-case) distortion bounds shown earlier in this article. Furthermore, our experimental results reveal several other properties and characteristics of our embedding scheme with interesting implications for its potential use in practice. Our implementation was carried out in C++; all experiments reported in this section were performed on a 1-GHz Intel Pentium-IV machine with 256 MB of main memory running RedHat Linux 9.0.

7.1 Implementation, Testbed, and Methodology

7.1.1 *Implementation Details: Subtree Fingerprinting.* A key point in our implementation was the use of Karp-Rabin (KR) probabilistic fingerprints [Karp and Rabin 1987] for assigning hash labels $h(t)$ to valid subtrees t of the input tree T in a one-to-one manner (with high probability). The KR algorithm was originally designed for strings so, in order to use it for trees, our implementation makes use of the flattened, parenthesized string representation of valid subtrees of T to obtain the corresponding tree fingerprint (treating parentheses as special delimiter labels in the underlying alphabet). An important property of the KR string-fingerprinting scheme is its ability to easily produce the fingerprint $h(s_1 s_2)$ of the concatenation of two strings s_1 and s_2 given only their individual fingerprints $h(s_1)$ and $h(s_2)$ [Karp and Rabin 1987]. This is especially important in the context of our data-stream processing algorithms since, clearly, we cannot afford to retain entire subtrees of the original (streaming) XML data tree T in order to compute the corresponding fingerprint in the current phase of our hierarchical tree parsing—the result would be space requirements linear in $|T|$ for each parsing phase. Thus, we need to be able to compute the fingerprints of valid subtrees corresponding to nodes $v \in T^i$ using only the fingerprints from the nodes in T^{i-1} that were contracted by TREEEMBED to obtain node v. This turns out to be nontrivial since, unlike the string case where the only possible options are left or right concatenation, TREEEMBED can merge the underlying subtrees of T in several different ways (Figures 3 and 4).

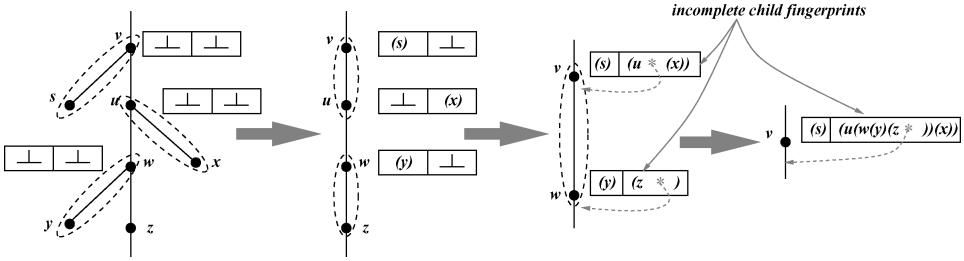


Fig. 11. Example of subtree fingerprint propagation. ("⊥" denotes an `empty` fingerprint and "∗" separates the left and right part of an `incomplete` fingerprint.)

The solution we adopted in our TREEEMBED implementation is based on the idea of maintaining, for each node in the current phase $v \in T^i$, a *collection of subtree fingerprints* corresponding to the child subtrees of v in the original data tree T. Briefly, all of v's fingerprints start out in an `empty` state, and a fingerprint becomes `complete` (meaning that it contains the complete fingerprint for the corresponding child subtree) once the last node along that branch is merged into node v. Child fingerprints of v can also be in an `incomplete` state, meaning that the subtree along the corresponding branch has only been *partially* merged into v; in order to correctly merge in the remaining subtree, an `incomplete` fingerprint consists of both a left and a right part that will eventually enclose the fingerprint propagated up by the rest of the subtree. The key to our solution is, of course, that we can always compute the fingerprint of a valid subtree at phase i by *simple concatenations* of the fingerprints from nodes in phase $i-1$. (Note that a sequence of `complete` child fingerprints can always be concatenated to save space, if necessary.) Figure 11 illustrates the key ideas in our subtree fingerprinting scheme following a simple example scenario of edge contractions. To simplify the exposition, the figure uses parenthesized nodelabel strings instead of actual numeric KR fingerprints of these strings; again, the key here is that fingerprints for new nodes (obtained through contractions in the current phase) are computed by simply concatenating existing KR fingerprints. Fingerprinting and merging for subtrees rooted at *unlabeled* nodes (Figure 4) can also be easily handled in our scheme.

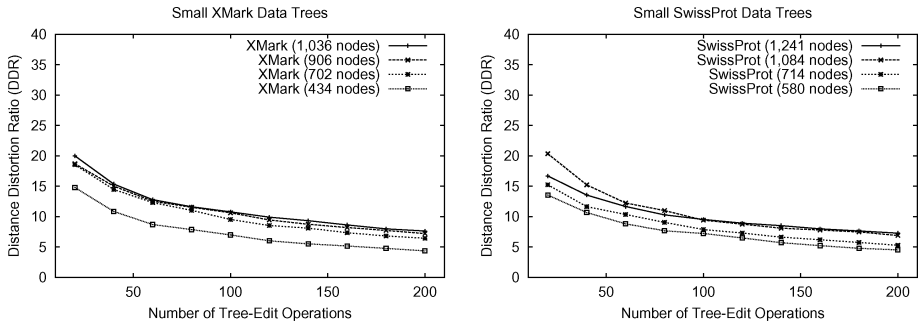
The KR-fingerprinting scheme maps each string in an input collection of strings to a number in the range $[0, p]$, where p is a prime number large enough to ensure that distinct input strings are mapped to distinct numbers with sufficiently high probability. Given that the total number of valid subtrees created during our hierarchical parsing of an input XML tree T is guaranteed to be only $O(|T|)$, we chose the prime p for our subtree fingerprinting to be $p = \Theta(|T|^2)$—this clearly suffices to ensure high-probability one-to-one fingerprints in our scheme.

7.1.2 *Experimental Methodology.* One of the main metrics used in our study to gauge the effectiveness of our tree-edit distance embedding scheme is the *distance-distortion ratio* which is defined, for a given pair of XML data trees S and T, as the quantity $\mathtt{DDR}(S,T) = \frac{\|V(S)-V(T)\|_1}{d(S,T)}$ (where $d(S,T)$ is the tree-edit distance of S and T and $V(S)$, $V(T)$ are the vector embeddings computed

by our TREEEMBED algorithm). Based on our initial experimental results, we also decided to employ a heuristic, *normalized distance-distortion ratio* metric NormDDR(S, T) in which the L_1 vector distance $\|V(S) - V(T)\|_1$ is normalized by the maximum of the depths of the parse trees (produced by TREEEMBED) for S and T; in other words, letting $\rho(S)$, $\rho(T)$ denote the number of TREEEMBED parsing phases for S and T (respectively), we define $\texttt{NormDDR}(S, T) = \frac{\texttt{DDR}(S,T)}{\max\{\rho(S),\rho(T)\}}$. (We discuss the rationale behind our NormDDR metric later in this section.)

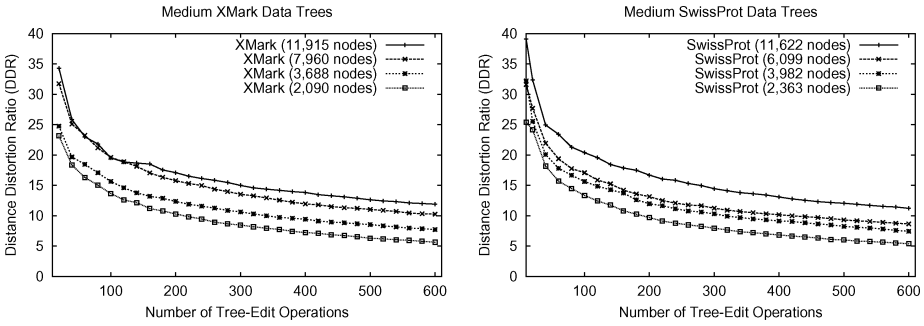
Unfortunately, the problem of computing the exact tree-edit distance with subtree-move operations (i.e., $d(S, T)$ above) turns out to be $\mathcal{NP}$-hard—this is a direct implication of the recent $\mathcal{NP}$-hardness result of Shapira and Storer [2002] for the simpler string-edit distance problem in the presence of substring moves. Furthermore, to the best of our knowledge, no other efficient approximation algorithms have been proposed for our tree-edit distance computation problem. Given the intractability of exact $d(S, T)$ computation and the lack of other viable alternatives (the sizes of our data sets preclude any brute-force, exhaustive technique), we decided to base our experimental methodology on the idea of performing *random tree-edit perturbations* on input XML trees. Briefly, given an XML tree T, we apply a script rndEdits() of random tree-edit operations (inserts, deletes, relabels, and subtree moves) on randomly selected nodes of T to obtain a perturbed tree rndEdits(T). Special care is taken in the creation of the rndEdits() edit-script in order to avoid redundant operations. Specifically, the key idea is to grow the rndEdits() script incrementally, storing a signature for each randomly chosen (node, operation) combination inside a set data structure; then, once a new random (node, operation) pair is selected, we employ our stored set of signatures together with a simple set of rules to check that the new edit operation is not redundant before entering it into rndEdits(). Examples of such redundant-operation checks include the following: (1) do not relabel the same node more than once, (2) do not move the same subtree more than once, (3) do not delete a previously inserted node, (4) do not insert a node in exactly the same location as a previously-deleted node, and so on. Even though our set of rules is not guaranteed to eliminate all possible redundancies in rndEdits(), we have found it to be quite effective in practice. Finally, we compute an (approximate) distance-distortion ratio DDR$(T,$ rndEdits$(T))$, where $d(T,$ rndEdits$(T))$ is approximated as $d(T,$ rndEdits$(T)) \approx |$rndEdits()$|$, that is, the number of tree-edit operations in our random script—since we explicitly try to avoid redundant edits, this is bound to be a reasonably good approximation of the true tree-edit distance (with moves) between the original and modified tree.

7.1.3 *Data Sets.* We used both synthetic and real-life XML data trees of varying sizes in our empirical study. These trees were obtained from (1) XMark [Schmidt et al. 2002], a synthetic XML data benchmark intended to model the activities of an on-line auction site (www.xml-benchmark.org/), and (2) SwissProt, a real-life XML data set comprising curated protein sequences and accompanying annotations (us.expasy.org/sprot/). We controlled the size of the XMark data trees using the "scaling factor" input to the XMark data generator. SwissProt is a fairly large real-life XML data collection (of total size over 165 MB)—in order to control the size of our input SwissProt trees, we used a



(a) (b)

Fig. 12. TREEEMBED distance distortion ratios for small (a) `XMark`, and (b) `SwissProt` data trees.



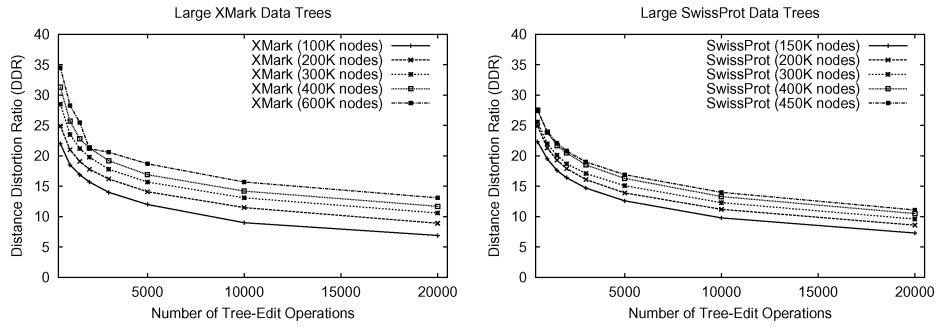
(a) (b)

Fig. 13. TREEEMBED distance distortion ratios for medium (a) `XMark`, and (b) `SwissProt` data trees.

simple sampling procedure that randomly selects a subset of top-level `<Entry>` nodes from `SwissProt`'s full tree with a certain sampling probability (where, of course, larger sampling probabilities imply larger generated subtrees). For both data sets, we partitioned the set of input data trees into three broad classes: (1) a *small* class comprising trees with sizes approximately between 400 and 1200 nodes; (2) a *medium* class with trees of sizes approximately between 2000 and 20,000 nodes; and (3) a *large* class with trees of sizes approximately between 100,000 and 600,000 nodes. The number of random tree-edit operations in our edit scripts (|`rndEdits()`|) was typically varied between 20–200 for small trees, 20–600 for medium trees, and 200–20, 000 for large trees; in order to smooth out randomization effects, our results were averaged over five distinct runs of our algorithms using different random seeds for generating the random tree-edit script. The numbers presented in the following section are indicative of our results on all data sets tested.

7.2 Experimental Results

7.2.1 TREEEMBED *Distance Distortions for Varying Data-Set Sizes.* The plots in Figures 12, 13, and 14 depict several observed tree-edit distance-distortion ratios obtained through our TREEEMBED algorithm for (a) `XMark` and



(a) (b)

Fig. 14. TREEEMBED distance distortion ratios for large (a) `XMark`, and (b) `SwissProt` data trees.

(b) `SwissProt`, in the case of small, medium, and large data trees (respectively). We plot the distance-distortion ratio as a function of the number of random tree edits in our edit script; thus, based on our discussion in Section 7.1, the x axis in our plots essentially corresponds to the true tree-edit distance value between the original and modified input trees. Our numbers clearly show that the distortions imposed by our L_1 vector embedding scheme on the true tree-edit distance typically vary between a factor of 4–20 on small inputs, a factor of 5–30 on medium inputs, and a factor of 10–35 on the large `XMark` and `SwissProt` trees. It is important to note that these experimental distortion ratios are obviously much better (by an *order of magnitude or more*) than what the pessimistic worst-case bounds in our analysis would predict for TREEEMBED. More specifically, based on the size of the trees (n) in our experiments, it is easy to verify that our worst-case distortion bound of $\log^2 n \log^* n$ (even ignoring all the constant factors in our analysis and those in Cormode and Muthukrishnan [2002]) gives values in the (approximate) ranges 230–300 (for small trees), 360–600 (for medium trees), and 850–1,100 (for large trees); our experimental distortion numbers are clearly much better.

An additional interesting finding in *all* of our experiments (with both `XMark` and `SwissProt` data) is that our tree-edit distance estimates based on the L_1 difference of the embedding vector images *consistently overestimate (i.e., expand)* the actual distance; in other words, for all of our experimental runs, $\texttt{DDR}(T, S) \geq 1$. Furthermore, note that the range of our experimental (over)estimation errors appears to grow quite slowly over a wide range of values for the tree-size parameter n (for instance, when moving from trees with $n \approx 4,000$ nodes to trees with $n \approx 600,000$ nodes). These observations along with a closer examination of some of our experimental results and the specifics of our TREEEMBED embedding procedure motivate the introduction of our *normalized* distance-distortion ratio metric (discussed below).

7.2.2 *A Heuristic for Normalizing the L_1 Difference: The* `NormDDR` *Metric.* Our experimental distance-distortion ratio numbers clearly demonstrate that our TREEEMBED algorithm satisfies the theoretical worst-case distortion guarantees shown in this article, typically improving on these worst-case bounds by

well over an order of magnitude on synthetic and real-life data. Still, it is not entirely clear how to interpret the importance of these numbers for real-life, practical XML-processing scenarios. Distance overestimation ratios in the range of 5–30 are obviously quite high and could potentially lead to poor sketching-based query estimates (e.g., for a streaming XML similarity join). Based on our experimental observations and the details of our TREEEMBED algorithm, we now propose a simple heuristic rule for *normalizing* the L_1 difference of the image vectors that could potentially be used to provide more useful tree-edit distance estimates.

Consider an input tree T to our TREEEMBED procedure, and let $\rho(T)$ denote the number of phases in the parsing of T. Now, assume that we effect a single edit operation (e.g., a node relabel) at the bottom level (i.e., tree $T^0 = T$) of our parsing to convert T to a new tree S. It is not difficult to see that this one edit operation is going to "hit" (i.e., affect the corresponding valid subtree of) *at least one node* at each of the $\rho(T)$ parsing phases of T, thus resulting in an L_1 difference $\|V(T) - V(S)\|_1$ in the order of $\rho(T)$. In other words, even though $d(T, S) = 1$, just by going through the different parsing phases, the effect of that single edit operation on T is amplified by a factor of $O(\rho(T))$ in the resulting L_1 distance. Generalizing from this simple scenario, consider a situation where T is modified by a relatively small number of edit operations (with respect to the size of T) applied to nodes randomly spread throughout T. The key observation here is that, since we have a small number of changes at locations spread throughout T, the effects of these changes on the different parsing phases of T will remain pretty much independent until near the end of the parsing; in other words, the nodes "hit"/affected by different edit operations will not be merged until the very late stages of our hierarchical parsing. Thus, under this scenario, we would once again expect the original edit distance to be amplified by a factor of $O(\rho(T))$ in the resulting L_1 distance.

A closer examination of some of our experimental results validated the above intuition. Remember that our `rndEdits()` script does in fact choose the target nodes for tree-edit operations randomly throughout the input tree T; furthermore, as expected, the impact of the parse-tree depth $\rho(T)$ on the approximate tree-edit distance estimates is more evident when the number of edit operations in `rndEdits()` is relatively small compared to the size of T. This obviously explains the clear downward trend for the distance-distortion ratios in Figures 12–14.

Based on the above discussion, we propose normalizing the L_1 distance of the image vectors in our embedding by the maximum parse-tree depth; that is, we estimate $d(S, T)$ using the ratio $\frac{\|V(S)-V(T)\|_1}{\max\{\rho(S),\rho(T)\}}$. Figure 15 depicts our experimental numbers for the corresponding *normalized distance-distortion ratio* $\texttt{NormDDR}(S, T) = \frac{\texttt{DDR}(S,T)}{\max\{\rho(S),\rho(T)\}}$ for several `XMark` and `SwissProt` data trees of varying sizes. Clearly, the normalized L_1 distance gives us much better tree-edit distance estimates in our experimental setting, typically ranging between a factor of 0.5 and 2.0 of the true tree-edit distance. Such distortions could be acceptable for several real-life application scenarios, especially when dealing

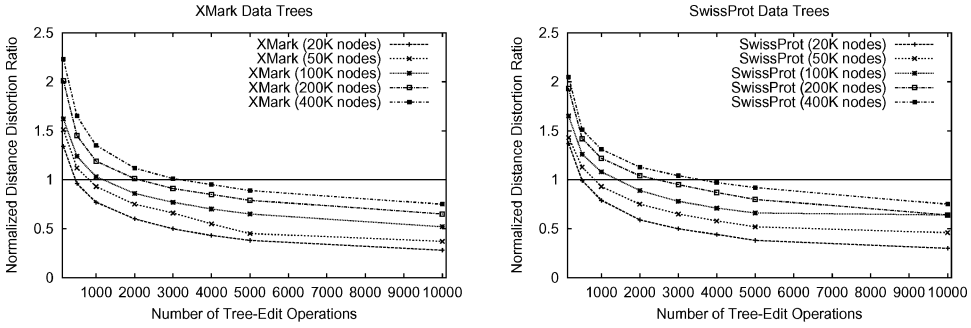


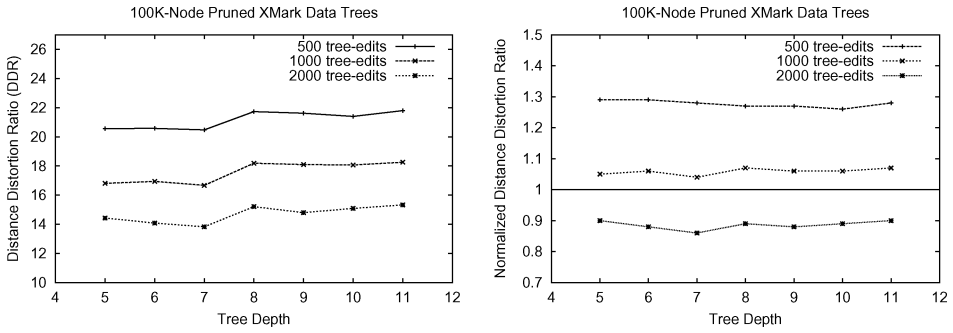
Fig. 15. TREEEMBED normalized distance distortion ratios for (a) `XMark`, and (b) `SwissProt` data trees.

with data collections with well-defined, well-separated structural clusters of XML documents (as we typically expect to be the case in practice).

Of course, we should stress that normalizing the L_1 distance estimate by the parse-tree depth is only a heuristic solution that is *not* directly supported by the theoretical analysis of TREEEMBED (Section 4). This heuristic may work well for the case of a small number of randomly-spread edit operations; however, when such operations are "clustered" in T or their number is fairly large with respect to $|T|$, dividing by $\rho(T)$ may result in significantly underestimating the actual tree-edit distance (see the clear trend in Figure 15). Still, our normalization heuristic may prove useful in certain scenarios, for example, when dealing with streams of large XML documents that, based on some prior knowledge, cannot be radically different from each other (i.e., they are all within an edit-distance radius which is much smaller than the document sizes).

7.2.3 *Effect of Tree Depth.* `SwissProt` is a fairly shallow XML data set (of maximum depth ≤ 5); thus, to study the potential effect of tree depth on the estimation accuracy of our embedding we concentrate solely on trees produced from the `XMark` data generator. More specifically, our methodology is as follows. We generate large (400,000-node) `XMark` data trees and, for a given value of the tree-depth parameter, we prune all nodes below that depth. Then, we make sure that the resulting pruned trees T at different depths all have the same approximate target size t using the following iterative rule: while $|T|$ is larger (smaller) than t pick a random node x in T and delete (respectively, replicate) its subtree at x's parent (making, of course, sure that the tree resulting from this operation is not too far from our target size and that the depth of the tree does not change). Finally, we run our `rndEdits()` scripts on these pruned trees with varying numbers of specified edit operations, and measure the observed normalized and unnormalized distance-distortion ratios for each depth value.

The plots in Figure 16 depict the observed unnormalized and normalized distance-distortion ratios as a function of tree depth for a pruned-tree target size of 100,000 nodes and for different numbers of tree-edit operations. Our experimental numbers clearly indicate that the estimation accuracy of our embedding



(a) (b)

Fig. 16. TREEEMBED unnormalized (a) and normalized (b) distance distortion ratios as a function of tree depth for 100,000-node (pruned) XMark data trees.

scheme does not have any direct dependence on the depth of the input tree(s)—the key experimental parameters affecting the quality of our estimates appear to be the size of the tree and the number of tree-edit operations.

7.2.4 *Using* TREEEMBED *for Approximate Document Ranking.* We now experimentally explore a different potential use for our XML-tree signatures in the context of approximate XML-document ranking based on the tree-edit distance similarity metric. In this setup, we are given a target XML document T and a number of incoming XML documents that are within different tree-edit distance ranges from T. The goal is to quickly *rank* incoming documents based on their tree-edit distance from T, such that if $d(S_1, T) < d(S_2, T)$ then S_1 is ranked "higher" than S_2. Since computing the exact tree-edit distances can be very expensive computationally, we would like to have efficient, easy-to-compute tree-edit distance estimates that can be used to *approximately* rank incoming documents. Our idea is to use TREEEMBED to produce L_1 vector signatures for both the target document T and each incoming document S_i, and use the L_1 distances $\|V(T) - V(S_i)\|_1$ for the approximate ranking of S_i's. The key observation here, of course, is that, for effective document ranking, it is crucial for our estimation techniques to preserve the *relative ranking* of individual tree-edit distances (rather than to accurately estimate each distance). Our experimental results demonstrate that our embedding schemes could provide a useful tool in this context.

For our document-ranking experiments, we vary the size of the target document T between 10,000 and 200,000 nodes. For a given target T and tree-edit distance d, we generate 40 different trees S_i at distance d from T (using different runs of our `rndEdits()` script). We vary the tree-edit distance d in three distinct ranges (10–50, 100–500, and 1000–3000) and, for a given value of d, we measure the observed range of (a) L_1 distances $\|V(T) - V(S_i)\|_1$, and (b) normalized L_1 distances $\frac{\|V(T)-V(S_i)\|_1}{\max\{\rho(T),\rho(S_i)\}}$, over the corresponding set of 40 S_i trees. Our experimental results for 50,000-node XMark and SwissProt data trees are shown in Table I. Note that in almost all cases, the approximate tree-edit distance ranges provided by our two L_1-distance metrics for the S_i sets (1) are

Table I. Approximate Document-Ranking Results: 50K-Node XMark and SwissProt Data Trees

	XMark Data		SwissProt Data	
$d(T, S_i)$	L_1 Distance	Normalized L_1 Distance	L_1 Distance	Normalized L_1 Distance
10	386–822	22.7–48.3	325–562	20.3–35.1
20	766–1083	45–63.7	688–883	43–55.2
30	994–1417	58.4–83.3	901–1212	56.3–75.7
40	1318–1580	77.5–92.9	1258–1542	78.6–96.4
50	1499–1915	88.1–112.6	1400–1667	87.5–104.2
100	2581–3194	151.8–187.8	2461–2792	153.8–174.5
200	4519–4992	265.8–293.6	4278–4831	267.4–301.9
300	6181–6571	363.5–386.5	6040–6294	377.5–393.4
400	7700–8411	452.9–494.7	7437–8126	464.8–507.9
500	8940–9653	525.8–567.8	8696–9246	543.5–577.9
1000	15278–16083	898.7–946	14615–15443	913.4–965.2
1500	20933–21393	1231.3–1258.4	19357–20171	1209.8–1260.7
2000	25114–25974	1477.29–1527.9	27599–27916	1724.9–1744.7
2500	29537–30251	1737.4–1779.4	28562–29331	1785.1–1822.2
3000	33228–34199	1954.6–2011.7	30545–31452	1909–1965.7

completely disjoint and (2) *preserve the ranking* of the corresponding true edit distances $d(T, S_i)$—this, of course, implies that our L_1 estimates correctly rank *all* the S_i input trees in most of our test cases. The only situation where our observed L_1-estimate ranges show some (typically small) overlap is for very small differences in tree-edit distance, that is, when $|d(T, S_i) - d(T, S_j)| = 10$ in Table I. Thus, for such small edit-distance separations (remember that we are dealing with 50,000-node trees), it is possible for our L_1 estimates to misclassify certain input documents; still, a closer examination of our results shows that, even in these cases, the percentage of misclassifications is always below 17.5% (i.e., at most 7 out of 40 documents).

It is worth noting that our approximate document-ranking setup is, in fact, closely related to a simple version of the approximate similarity-join scenarios discussed in Section 6. In a sense, our goal here is to correctly identify the "closest" approximate duplicates of a target document T in a collection of input documents S_i that are within different tree-edit distances of T—these closest duplicates essentially represent the subset of S_i documents that would *join* with T (for an appropriate setting of the similarity threshold to account for the distance distortion; see Theorem 6.1). Thus, assuming that the L_1/AMS sketching techniques developed in Section 6 correctly preserve the L_1-distance ranges of the underlying image vectors, our ranking results provide an indication of the percentages of false positives/negatives in the approximate similarity-join operation (based on the overlap between different distance ranges), and the required "distance separation" between document clusters in the joined XML streams to suppress such estimation errors.

7.2.5 *Running Time and Space Requirements.* Table II depicts the observed running-times and memory footprints for our TREEEMBED embedding

Table II. TREEEMBED Running Times and Memory Footprints: XMark Data Trees

Tree Size	Document Size	TREEEMBED Running Time	TREEEMBED Memory Footprint
20K nodes	1.1 MB	2.5 s	3.0 MB
50K nodes	2.9 MB	6.3 s	7.9 MB
100K nodes	5.7 MB	12.7 s	16.7 MB
150K nodes	8.9 MB	19.7 s	25.0 MB
200K nodes	11.7 MB	26.5 s	33.3 MB
250K nodes	14.5 MB	34.4 s	41.9 MB
300K nodes	17.4 MB	41.2 s	49.1 MB
350K nodes	20.5 MB	49.9 s	57.9 MB
400K nodes	23.5 MB	58.2 s	66.4 MB

algorithm over XMark data trees of various sizes (the results for SwissProt are very similar and are omitted). We should, of course, note here that our current TREEEMBED implementation does not employ the small-space optimizations discussed in Section 5 (that is, we always build the full XML tree in memory); still, as our numbers show, the memory requirements of our scheme grow only linearly (with a small constant factor ≤ 3) in the size of the input document. Furthermore, our embedding algorithm gives very fast running times; for instance, our TREEEMBED code takes less than 1 min to build the L_1 vector image of a 400,000-node XML tree. Thus, once again, compared to computationally expensive, exact tree-edit distance calculations, our techniques can provide a very efficient, approximate alternative.

8. CONCLUSIONS

In this article, 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 pseudorandom sketching techniques to obtain novel, small-space algorithms for building concise sketch synopses and approximating similarity joins over streaming XML data. An empirical study with synthetic and real-life data sets has validated our approach, demonstrating that the behavior of our embedding scheme over realistic XML trees is much better than what would be predicted based on our worst-case distortion bounds, and revealing several interesting properties of our algorithms in practice. Our embedding result also has other important algorithmic applications, for example, as a tool for very fast, approximate tree-edit distance computations.

APPENDIX: ANCILLARY LEMMAS FOR THE UPPER-BOUND PROOF

In this section, we complete the upper bound proof for the distortion of our embedding algorithm. Recall the terminology of Section 4.4. We have defined sets M^i and P^i as subsets of T^i and $(T_1 + T_2)^i$, respectively. We have assumed by induction that for every $x \in T^i - M^i$ there exists a unique $f(x) \in (T_1 + T^2)^i$

such that the trees $x(T)$ and $f(x)(T_1 + T_2)$ look identical. In other words, if we forget about the regions M^i and P^i, the remaining forests are basically the same. Now, we want to prove this fact for the next stage of parsing. In order to do this, we defined another region N^i which encloses M^i and in some way represents the region which can get influenced by M^i in the next stage of parsing.

So we pick a node $w \in T^i - N^i$ and w' is the corresponding node in $(T_1 + T_2)^i$. So we know that $w(T)$ and $w'(T_1 + T_2)$ are the same trees. Now, w gets absorbed in a node q in T^{i+1} and w' in q' in $(T_1 + T_2)^{i+1}$. Our goal now is to show that $q(T)$ and $q'(T_1 + T_2)$ are identical trees. We will do this by following the a natural procedure—we will show that w and w' get parsed in exactly the same manner, that is, if w gets merged with a leaf, then the same thing happens to w'. But note that it is not enough to just show this fact. For example, if w and w' get merged with leaves l and l', we have to show that $l(T)$ and $l'(T_1 + T_2)$ are also identical subtrees.

Thus, we first need to go through a set of technical lemmas, which show that the mapping f preserves the neighborhood of w as well. For example, we need to show facts like parent of w and parent of w' get associated by f as well. So, we need to explore the properties of f and first show that it preserves sibling relations and parent-child relations. Once we are armed with these lemmas, we just need to prove the following facts:

—If w is a leaf and if merged with its parent, then the same happens to w'.
—If w is a leaf and is merged with some of its leaf siblings, then w' is also a leaf and gets merged with the corresponding siblings.
—If w has a leaf child which gets merged into it, then w' also has a corresponding leaf child which gets merged into it.
—If w is a degree-2 node in a chain, and gets merged with some other such nodes, then the same fact applies to w'.

Clearly, the above facts will be enough to prove the result we want. As we mentioned before, we need to analyze some properties of the parsing algorithm and the function f, so that we can set up a correspondence between neighborhoods of w and w'. We proceed to do this first.

We first show the connection between a node w and the associated tree $w(T)$. The following fact is easy to see.

CLAIM A.1. *Let x and y be two distinct nodes in T^i. x is a parent of y iff $x(T)$ contains a node which is the parent of a node in $y(T)$.*

PROOF. Proof is by induction on i. Let us say the fact is true for $i-1$. Suppose x is a parent of y in T^i. Let X' be the set of nodes in T^{i-1} which got merged to form x. Define Y' similarly. Then there must be a node in X' which is the parent of a node in Y'. The rest follows by induction on these two nodes. The reverse direction is similar. □

LEMMA A.2. *Suppose $x \in T^i$ is a leaf. Then $x(T)$ has the following property—if $y \in x(T)$, then all descendants of y are in $x(T)$.*

PROOF. Suppose not. Then there exist nodes $a, b \in T$ such that a is the parent of b, and yet $a \in x(T), b \notin x(T)$. Suppose b is in a node y' in T^i. But then the claim above implies that x is not a leaf, a contradiction. □

Now, we go on to consider the case when x has at least two children. Consider the nodes in T^{i-1} which formed x. Only two cases can happen—either x was present in T^{i-1} or T^{i-1} contains a node u with at least two children and x is obtained by collapsing a leaf child into u. In either case, x corresponds to a unique node with at least two children in T^{i-1}. Carrying this argument back all the way to T^0, we get the following fact.

CLAIM A.3. *Let x be a node with at least two children in T^i. Then $x(T)$ is a subtree of T which looks as follows: there is a unique node with at least two children, call it x_0, such that all nodes in $x(T)$ are descendants of x_0. Further, if $y \in x(T)$, $y \neq x_0$, then all descendants of y (in T) are also in $x(T)$.*

The proof of the fact above is again by induction and using the previous two claims.

CLAIM A.4. *Let x and y be two nodes in T^i. Suppose x is a sibling of y. Then, $x(T)$ contains a node which is a sibling of a node in $y(T)$. Conversely, if $x(T)$ contains a node which is a sibling of a node in $y(T)$, then x and y are either siblings or one of them is the parent of the other.*

PROOF. Suppose x is a sibling of y. Let w be their common parent. w has at least two children. By Claim A.3, $w(T)$ contains a node w_0 such that if $z \in w(T), z \neq w_0$, then $w(T)$ contains all descendants of z.

Claim A.1 implies that there is a node $a \in x(T)$ and a node $b \in w(T)$ such that a is a child of b. We claim that $b = w_0$. Indeed, otherwise all descendants of b, in particular, a, should have been in $w(T)$. Similarly, there is a node c in $y(T)$ whose parent is w_0. But then a and c are siblings in T.

Conversely, suppose there is a node in $x(T)$ which is a sibling of a node in $y(T)$. Let the common parent of these two nodes in T be w. Let w' be the node in T^i containing w. If w' is x, then x is the parent of y. So, assume w' is not x or y. It follows from Claim A.1 that w' is the parent of x and y. So, x and y are siblings. □

Recall that we associate a set P^i with M^i. We already know that M^i is a connected subtree. Of course, we can not say the same for P^i because $T_1 + T_2$ itself is not connected. But we can prove the following fact.

LEMMA A.5. *P^i restricted to T_1^i or T_2^i is a connected set.*

PROOF. Suppose P^i is not connected. Then there exist two nodes x, y in the same component of $(T_1 + T_2)^i$, such that $x, y \in P^i$ but at least one internal node in the path between x and y is not in P^i. We can in fact assume that all internal nodes in this path are not in P^i (otherwise, we can replace x and y by two nodes on this path which are in P^i but none of the nodes between them are not in P^i). Let this path be $x, a_1, \ldots, a_n, y$. Let $b_i \in T^i - M^i$ be such that $f(b_i) = a_i$.

First observe that b_i is adjacent with b_{i-1}. Indeed, suppose a_i is the parent of a_{i-1} (the other case will be a_i is a child of a_{i-1}, which is similar). By Claim A.1, there is a node in $a_i(T_1+T_2)$ which is the parent of a node in $a_{i-1}(T_1+T_2)$. Since $a_i(T_1+T_2) = b_i(T)$ and $a_{i-1}(T_1+T_2) = b_{i-1}(T)$, we see that there is a node in $b_i(T)$ which is the parent of a node in $b_{i-1}(T)$. Applying Claim A.1, we see that b_i is adjacent with b_{i-1}. Thus, $b_1, \ldots, b_n$ is a path.

Since x is adjacent with a_1, there is a node x_0 in $x(T_1+T_2)$ which is adjacent with a node c_1 in $a_1(T_1+T_2)$ (again, using Claim A.1). If x_0 is not the root of T_1, then x_0 is also a node in T. So, there is a node $x' \in T^i$, such that $x_0 \in x'(T)$. Clearly $x' \in M^i$; otherwise, $f(x')$ must be x (since $f(x')(T_1+T_2)$ and $x(T_1+T_2)$ will share the node x_0 and so must be the same). Now, x' must be adjacent with b_1 (because x_0 is adjacent with c_1 in T—note that c_1 is a node in T as well). The other case arises when x_0 is the root of T_1. So, x_0 is the parent of c_1. But then v is the parent of c_1 in T. Let x' be the node containing v in the tree T^i. So $x' \in M^i$ and is adjacent with b_1.

Thus, we get a node $x' \in M^i$, such that x' is adjacent with b_1. In fact, if x is the parent (child) of a_1, then the same applies to x and b_1 (and vice versa). Similarly, there is a node $y' \in M^i$ adjacent with b_n. So if x' and y' are different nodes in M^i, then this contradicts the fact that M^i is connected. So $x' = y'$. To avoid any cycles in T^i, it must be the case that $b_1 = \cdots = b_n$. First observe that, in this case, x and y are children of a_1. The only other possibility is that x is the parent of a_1 and y a child of a_1—but then x' is the parent of b_1 and y' a child of b_1 and so x', y' cannot be the same nodes.

Thus, we have that x, y are children of a_1 and x' is a child of b_1. By Claim A.3, $a_1(T_1+T_2)$ has a node a' which is the parent of a node $x_0 \in x(T_1+T_2)$ and a node $y_0 \in y(T_1+T_2)$. By definition of x' (and y'), $x_0, y_0 \in x'(T)$. Further, $a' \in b_1(T)$. Consider the largest integer i' such that the nodes containing x_0 and y_0 in the tree $T^{i'}$ were different—call these nodes x'' and y''. Let b'' be the node containing a'. So x'' and y'' are children of b''. Also $i' < i$.

When we parse $T^{i'}$, we merge x'' and y'' into a single node, $x''' \in T^{i'+1}$. However, in $(T_1+T_2)^{i'+1}$, the nodes containing x_0 and y_0 are different. So, $x''' \in M^{i'+1}$. So one of the nodes of $T^{i'}$ which merged into x'' must have been in $N^{i'}$. Since x'' and y'' are siblings and we merge them, it must be the case that they are leaves. Thus, the only nodes in $T^{i'}$ which get merged to form x''' are leaf children of b''. So one of these leaf children is in $N^{i'}$. Since $N^{i'}$ is a connected set and has size at least 2, $b'' \in N^{i'}$. But then $b_1 \in M^i$, a contradiction. □

We now show the fact that f preserves parent-child and sibling relations.

LEMMA A.6. *Suppose x and y are two nodes in $T^i - M^i$. If x is the parent of y, then $f(x)$ is the parent of $f(y)$. If x is a sibling of y, and $f(x)$, $f(y)$ are in the same component of $(T_1+T_2)^i$, then $f(x)$ is a sibling of $f(y)$. The converse of these facts is also true.*

PROOF. Suppose x is the parent of y. By Claim A.1, there is a node $x' \in x(T)$ and a node $y' \in y(T)$ such that x' is the parent of y' in T. x' and y' are nodes in $T_1 \cup T_2$ as well. Unless $x' = v$, x' is the parent of y' in $T_1 \cup T_2$ as well. If $x' = v$, x will be in M^i, which is not the case.

Thus, x' is the parent of y' in $(T_1 \cup T_2)$. Since $f(x)(T_1 + T_2) = x(T)$, $f(y)(T_1 + T_2) = y(T)$, another application of Claim A.1 to $T_1 \cup T_2$ implies that $f(x)$ is a parent of $f(y)$.

The other fact can be proved similarly using Claim A.4. The converse can be shown similarly. □

LEMMA A.7. *Suppose x is a leaf node in $T^i - M^i$. Then $f(x)$ is a leaf in $(T_1 + T_2)^i - P^i$. The converse is also true.*

PROOF. Suppose $f(x)$ has a child z' in $(T_1 + T_2)^i$. So there are nodes $a \in w'(T_1 + T_2)$ and $b \in z'(T_1 + T_2)$ such that a is the parent of b in $T_1 \cup T_2$. Let z be the node in T^i which contains b. Since $f(x)(T_1 + T_2) = x(T)$, x should be the parent of z, which is a contradiction. So $f(x)$ is a leaf as well. The converse can be shown similarly. □

LEMMA A.8. *Suppose a node u in M^i has at least two children. Let x be a child of u. If there is a node in $x(T)$ which is an immediate sibling of a node in $u(T)$, then x is a corner node.*

PROOF. By Claim A.3, $u(T)$ has a node u_0 such that any other node in $u(T)$ has all its descendants in $u(T)$. Now, $x(T)$ has a node x_0 and $u(T)$ has a node u_1 such that x_0 and u_1 have common parent. So this common parent must be u_0. Consider the highest i for which the node in $T^{i'}$ containing u_0 was distinct from the node in $T^{i'}$ containing u_1—call these y and z, respectively. Clearly, $i' < i$. While parsing $T^{i'}$, we moved z up to its parent y. But then we must have marked all immediate siblings of z as corner nodes. In particular, the node containing x_0 must have been a corner node. This implies that x must be a corner node. □

We now state a useful property of the set M^i.

LEMMA A.9. *Suppose x is a node in M^i which has at least two children and at least one child of x is not in M^i. Then x is either c^i or v^i. Similarly, if x is a node in N^i which has at least two children such that at least one of them is not in N^i, then x is either v^i or the center node in N^i.*

PROOF. The proof follows easily by induction on i. When $i = 1$, M^i is simply v^i, so there is nothing to prove. So assume the induction hypothesis is true for some value of i. The only case when $N^i - M^i$ will have a with more than two children is when the center z of N^i is different from c^i. If $c^i = v^i$, we have nothing to prove because the new center of M^{i+1} will be the node containing z. So assume that c^i is not same as v^i. But then, all children of c^i must be in M^i (only then we shall move the center for N^i to a new node). So the node containing c^i in M^{i+1} will have all its children in M^{i+1}. This proves the lemma. □

Recall that w is defined to be a node in $T^i - N^i$ and $w' = f(w) \in (T_1 + T_2)^i$. We want to show that w and w' are parsed in the same manner, that is, $q(T) = q'(T_1 + T_2)$ (using the notation in Section 4.4). We now show that the two nodes will be parsed identically.

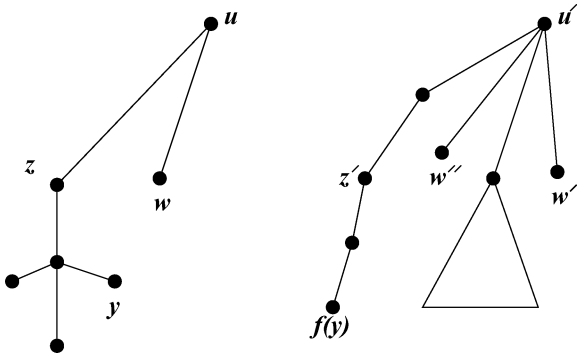


Fig. 17. Proof of lemma A.11 when $w'' \in P^i$.

LEMMA A.10. *Suppose w is a leaf (so w' is a leaf as well). If w is a lone leaf child of its parent, then so is w'. The converse is also true.*

PROOF. Suppose w is a lone leaf child of its parent, call it u. Let u' be the parent of w' in $(T_1 + T_2)^i$. Suppose, for the sake of contradiction, that w' is not a lone child of u'. So it has an immediate, say left, sibling w'', which is also a leaf.

We first argue that $w'' \in P^i$. Suppose not. Now, w'' corresponds to a node x in $T^i - M^i$, that is, $w'' = f(x)$. So x is also a leaf and a left sibling of w. But w is a lone child of u. So it must happen that there is a nonleaf child of u, call it y, between x and w (because x is a lone child). Observe that $y \in M^i$; otherwise $f(y)$ will lie between w' and w''. Since this holds for all nodes y between w and x, we should have added w to N^i (according to Rule (i)), a contradiction.

So it follows that $w'' \in P^i$. Let x be the immediate left sibling of w (if any), $x \notin M^i$; otherwise we would have added w to N^i. So $f(x)$ is a left sibling of w' (if they are in the same component). So w'' lies between $f(x)$ and w'. Since there is no node between x and w in T^i, all nodes in $w''(T_1+T_2)$ (we can think of this as nodes of T) must be part of $u(T)$. But then, by Lemma A.8, w should have been a corner node and should have been added to N^i. The converse can be shown similarly. □

LEMMA A.11. *Suppose w is the leftmost lone leaf child of its parent u. Then $u \notin M^i$. Further, w' is the leftmost lone leaf child of its parent u'.*

PROOF. Suppose $u \in M^i$. Then we would have added w to N^i. Define $u' = f(u)$. Then u' is the parent of w' (using Claim A.1). We already know that w' is a lone leaf child of u' (using the lemma above). Suppose it is not the leftmost such child. Let w'' be a lone leaf child of u' which is to the left of w'. First we argue that $w'' \notin P^i$.

Suppose $w'' \in P^i$ (see Figure 17). Consider the nodes in $w''(T_1 + T_2)$. One of these nodes must be a child of a node in $u'(T_1 + T_2)$. Let this node be z_0. z_0 is also a node in T. Further, z_0 is a child of a node in $u(T)$ because $u(T) = u'(T_1 + T_2)$. Let z be the node in T^i containing z_0. We claim that $z \in M^i$. Otherwise, $f(z)(T_1 + T_2) = z(T)$. So $z_0 \in f(z)(T_1 + T_2)$. But $z_0 \in w''(T_1 + T_2)$. Then, it must be the case that $w'' = f(z)$. But then $w'' \notin P^i$, a contradiction. So $z \in M^i$. Note that z is a child of u.

Let y be a descendant of z in T^i. Suppose $y \notin M^i$. Then $f(y)$ is a node in $(T_1 + T_2)^i$ such that $y(T) = f(y)(T_1 + T_2)$. Since y is a descendant of u, $f(y)$ is also a descendant of u'. If all nodes between $f(y)$ and u' were not in P^i, then we will get a path between y and u in T^i such that no internal node is in M^i. But this is not true (since $z \in M^i$). Thus, there is an internal node in this path which is in P^i—call this node z'.

So we have the situation that u' has a leaf child w'' and another descendant z' which are in P^i. But $u' \notin P^i$. This violates the fact that P^i is connected (Lemma A.5). So all descendants of z are in M^i. But then we would have added u to N^i and, consequently, w to N^i.

Thus, $w'' \notin P^i$. So there is a node $a \in T^i - M^i$ such that $f(a) = w''$. So a is a leaf as well. Suppose $a \in N^i$. Since N^i has at least two nodes and N^i is a connected set, it must be the case that $u \in N^i$. Note that we never add a node with at least two children to N^i using Rules (i)—(iv). So it must be the case that u is the center of N^i. But then we would have added w to N^i, a contradiction.

Thus, it follows that $a \notin N^i$. But then, by Lemma A.10, a is a lone leaf child as well. This contradicts the fact that w has this property. □

The lemma above shows that if w is merged with its parent, then so is w' and $q(T) = q'(T)$.

LEMMA A.12. *Suppose w is a leaf node. Let its immediate siblings on the right (left) be $w_0 = w, w_1, \ldots, w_k$, where all nodes except perhaps w_k are leaves. Further, suppose $k < \Delta$. Then $w_0, \ldots, w_k$ are not in M^i. Moreover, the immediate right (left) siblings of $f(w)$ in $(T_1 + T_2)^i$ are $f(w_0), f(w_1), \ldots, f(w_k)$.*

PROOF. Let u be the parent of w. If $w_j \in M^i$, then w will be added to N^i. So none of the nodes in $w_0, \ldots, w_k$ are in M^i. All we have to show now is that $f(w_j)$ is an immediate left sibling of $f(w_{j+1})$. Suppose not. Let x' be a sibling between $f(w_j)$ and $f(w_{j+1})$. All nodes in $x'(T_1 + T_2)$ must be in $u(T)$. Lemma A.8 now implies that w_j must be a corner node. But then w should be in N^i, a contradiction again. □

The lemma above shows that, if w was merged with a set of siblings, then w' will be merged with the same set of siblings.

LEMMA A.13. *Suppose w is a node with at least two children. Then $f(w)$ also has at least 2 children. Let y be the leftmost lone leaf child of w. Then $y \notin M^i$ and $f(y)$ is the leftmost lone child of $f(w)$.*

PROOF. Suppose w has a child u such that all descendants of u are in M^i. Then w will be added to N^i.

So not all descendants of of u_1 or u_2 are in M^i. Since M^i is a connected set, it can contain at most one of u_1 and u_2. So suppose $u_1 \notin M^i$. Then $f(u_1)$ is a child of $f(w)$. Further, let x be the descendant of u_2 which is not in M^i. Then $f(x)$ is a descendant of $f(w)$, but not a descendant of $f(u_1)$.

So, $f(w)$ has at least two children. We claim that $y \notin N^i$. Indeed, if $y \in N^i$, and the fact that N^i is a connected set, implies that $N^i = \{y\}$. So $M^i = \{y\}$. But then $w \in N^i$, a contradiction. So $y \notin N^i$. But then Lemma A.11 implies that $f(y)$ is a leftmost lone leaf child of w' as well. □

The lemma above implies that if w is merged with a leaf child, then w' is also merged with the same leaf child.

LEMMA A.14. *Suppose w is a degree-2 node. Let $(w_0 = w, w_1, \ldots, w_k)$ be an ancestor to descendant path of length at most $\Delta - 1$, such that all nodes except perhaps w_k are of degree-2. Then $w_0, \ldots, w_{k-1}$ are in $T^i - M^i$. Further, $f(w_0), \ldots, f(w_{k-1})$ forms a path of degree-2 nodes in $(T_1 + T_2)^i$.*

If w_k is a degree-2 node, then $w_k \notin M^i$ and $f(w_k)$ is adjacent with $f(w_{k-1})$. If w_k has degree at least 3, then the neighbor of $f(w_{k-1})$ other than $f(w_{k-2})$ is of degree at least 3 as well.

PROOF. First consider the case when $w_0, \ldots, w_k$ are nodes of degree 2. None of them can be in M^i; otherwise $w \in N^i$. Thus, $f(w_0), \ldots, f(w_k)$ is also a chain in $(T_1 + T_2)^i$. Now we need to argue that these nodes have degree 2 as well. But this is true from the fact that w_i and $f(w_i)$ represent the same tree in T.

So suppose w_k has at least two children. If $w_k \notin M^i$, then $f(w_k)$ also has at least two children. Hence assume that $w_k \in M^i$. If w is an ancestor of w_k, then w will be added to N^i. So assume w is a descendant of w_k. So w is a descendant of w_k. Further, if all descendants of w_k which are not descendants of w_{k-1} are in M^i, then $w \in N^i$. So w_k has a descendant x such that $x \notin M^i$ and x is not a descendant of w_{k-1}.

Now consider the tree $w_k(T)$—since w_k has at least two children, $w_k(T)$ has a unique highest node, z, such that all nodes in $w_k(T)$ are descendants of z. There is a node in $w_{k-1}(T)$ which is a child of z. Let w'_k be the node in $(T_1 + T_2)^i$ containing z. Then w'_k is the parent of $f(w_{k-1})$. Further, $f(x)$ is a descendant of w'_k, but not a descendant of $f(w_{k-1})$. So w'_k has degree at least 3. □

The lemma above shows that if w is a degree-2 node which is merged with some other degree-2 nodes, then w' will be merged with the same nodes. One final case remains.

LEMMA A.15. *Suppose w is not merged with any node in T^i. Then w' is also not merged with any node in $(T_1 + T_2)^i$.*

PROOF. First consider the case when w is a leaf. Let the parent of w be u. We know that w' is also a leaf. Let u' be its parent. If w' has siblings which are leaves, then Lemma A.10 implies that w is also not a lone leaf. But then w will be merged with a sibling, which is a contradiction. So w' must be a lone child of u'.

Suppose w' is the leftmost lone leaf child of u'. Let x be the leftmost lone leaf child of u. We know that $x \neq w$; otherwise w will merge with its parent. If $x \notin N^i$, then Lemma A.11 implies that $f(x)$ is the leftmost lone leaf child of u', which is not true because $f(w) = w'$ and f is 1-1. So $x \in N^i$.

Now we claim that $u \in N^i$. Indeed, if not, the fact that N^i is a connected set implies that $N^i = \{x\}$. But M^i is a subset of N^i, and so M^i must also be $\{x\}$. But then u will be added to N^i.

So we can assume $u \in N^i$. Now, Lemma A.9 implies that u is either v^i or the center of N^i. Suppose u is the center z of N^i. Let y be the leftmost lone leaf child of u which is not in M^i—y cannot be same as w, because y gets added to N^i. But then, $f(y)$ is a lone leaf child to the left of w'—a contradiction.

So now assume u is same as v^i. Two cases can happen—either w is a removed descendant of v^i or not—and in either cases we can argue by using Rule (ii) that it should be in N^i. Thus, we have shown that if w is a leaf node, then w' also does not get merged with any other node.

Now suppose w' has a lone leaf child x'. If $x' \notin P^i$, then there is a leaf node $x \notin N^i$ such that $f(x) = x'$. But then, x is also a lone leaf child of w (Lemma A.10). So w will also merge with one its leaf children, a contradiction.

Finally, suppose w' is a degree-2 node, and either the parent or the child of w' is of degree 2. First observe that w must also be of degree 2—otherwise it has at least two children. Since $f(w) = w'$, and w' has only one child, both these children must be in M^i—but then due to connectedness of M^i, w is also in M^i, a contradiction.

So w also has only one child, call it x. Let the child of w' be x'. Suppose x' has only one child. x must have at least two children, otherwise w will also be merged with a node. Now all but at most one child of x will be in M^i. If x has more than two children, then the fact that M^i is connected implies that x is in M^i as well. But then w will be added to N^i. So x has exactly two children—one of these is in M^i, the other not in M^i. Now $x \in N^i$; otherwise x' will also have at least two children. So x is the new center node of N^i. But then w will be added to the set N^i, a contradiction.

Now let the parent of w be y and that of w' be y'. Suppose y' has only one child. Then, Lemma A.10 implies that y also has only one child. But then, w will be merged with one of the nodes—a contradiction. Thus, w' is not merged with any node as well. □

Thus, we have demonstrated the invariant for T^{i+1} and $(T_1 + T_2)^{i+1}$.

ACKNOWLEDGMENTS

Most of this work was done while the second author was with Bell Labs. The authors thank the anonymous referees for insightful comments on the article, and Graham Cormode for helpful discussions related to this work.

REFERENCES

ACHARYA, S., GIBBONS, P. B., POOSALA, V., AND RAMASWAMY, S. 1999. Join synopses for approximate query answering. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data* (Philadelphia, PA). 275–286.

ALON, N., GIBBONS, P. B., MATIAS, Y., AND SZEGEDY, M. 1999. 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, PA).

ALON, N., MATIAS, Y., AND SZEGEDY, M. 1996. The space complexity of approximating the frequency moments. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing* (Philadelphia, PA). 20–29.

ALTINEL, M. AND FRANKLIN, M. J. 2000. Efficient filtering of XML documents for selective dissemination of information. In *Proceedings of the 26th International Conference on Very Large Data Bases* (Cairo, Egypt). 53–64.

APOSTOLICO, A. AND GALIL, Z., Eds. 1997. *Pattern Matching Algorithms.* Oxford University Press, Oxford, U.K.

ARASU, A., BABCOCK, B., BABU, S., MCALISTER, J., AND WIDOM, J. 2002. Characterizing memory requirements for queries over continuous data streams. In *Proceedings of the Twenty-first ACM*

SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (Madison, WI). 221–232.

BABCOCK, B., BABU, S., DATAR, M., MOTWANI, R., AND WIDOM, J. 2002. Models and issues in data stream systems. In *Proceedings of the Twenty-First ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Madison, WI). 1–16.

BAR-YOSSEF, Z., JAYRAM, T., KUMAR, R., SIVAKUMAR, D., AND TREVISAN, L. 2002. 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, MA).

CHAKRABARTI, K., GAROFALAKIS, M., RASTOGI, R., AND SHIM, K. 2000. Approximate query processing using wavelets. In *Proceedings of the 26th International Conference on Very Large Data Bases* (Cairo, Egypt). 111–122.

CHAN, C.-Y., FELBER, P., GAROFALAKIS, M., AND RASTOGI, R. 2002. Efficient filtering of XML documents with XPath expressions. In *Proceedings of the Eighteenth International Conference on Data Engineering* (San Jose, CA).

CHARIKAR, M., CHEN, K., AND FARACH-COLTON, M. 2002. Finding frequent items in data streams. In *Proceedings of the International Colloquium on Automata, Languages, and Programming* (Malaga, Spain).

CHARIKAR, M. AND SAHAI, A. 2002. Dimension reduction in the l_1 norm. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science* (Vancouver, B.C., Canada).

CORMODE, G., DATAR, M., INDYK, P., AND MUTHUKRISHNAN, S. 2002a. 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). 335–345.

CORMODE, G., INDYK, P., KOUDAS, N., AND MUTHUKRISHNAN, S. 2002b. Fast mining of massive tabular data via approximate distance computations. In *Proceedings of the Eighteenth International Conference on Data Engineering* (San Jose, CA).

CORMODE, G. AND MUTHUKRISHNAN, S. 2002. The string edit distance matching problem with moves. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (San Francisco, CA).

DASU, T. AND JOHNSON, T. 2003. *Exploratory Data Mining and Data Cleaning*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., New York, NY.

DIAO, Y., ALTINEL, M., FRANKLIN, M. J., ZHANG, H., AND FISCHER, P. 2003. Path sharing and predicate evaluation for high-performance XML filtering. *ACM Trans. Database Syst. 28*, 4 (Dec.), 467–516.

DIAO, Y. AND FRANKLIN, M. 2003. Query processing for high-volume XML message brokering. In *Proceedings of the 29th International Conference on Very Large Data Bases* (Berlin, Germany).

DOBRA, A., GAROFALAKIS, M., GEHRKE, J., AND RASTOGI, R. 2002. Processing complex aggregate queries over data streams. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data* (Madison, WI). 61–72.

DOBRA, A., GAROFALAKIS, M., GEHRKE, J., AND RASTOGI, R. 2004. Sketch-based multi-query processing over data streams. In *Proceedings of the 9th International Conference on Extending Database Technology* (EDBT'2004, Heraklion-Crete, Greece).

FEIGENBAUM, J., KANNAN, S., STRAUSS, M., AND VISWANATHAN, M. 1999. 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, NY).

FLORESCU, D., KOLLER, D., AND LEVY, A. 1997. Using probabilistic information in data integration. In *Proceedings of the 23rd International Conference on Very Large Data Bases* (Athens, Greece).

GANGULY, S., GAROFALAKIS, M., AND RASTOGI, R. 2004. Processing data-stream join aggregates using skimmed sketches. In *Proceedings of the 9th International Conference on Extending Database Technology* (EDBT'2004, Heraklion-Crete, Greece).

GAROFALAKIS, M., GEHRKE, J., AND RASTOGI, R. 2002. Querying and mining data streams: you only get one look (Tutorial). In *Proceedings of the 28th International Conference on Very Large Data Bases* (Hong Kong, China).

GAROFALAKIS, M. AND GIBBONS, P. B. 2001. Approximate query processing: Taming the terabytes (Tutorial). In *Proceedings of the 27th International Conference on Very Large Data Bases* (Roma, Italy).

GAROFALAKIS, M. AND KUMAR, A. 2003. Correlating XML data streams using tree-edit distance embeddings. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (San Diego, CA). 143–154.

GILBERT, A. C., GUHA, S., INDYK, P., KOTIDIS, Y., MUTHUKRISHNAN, S., AND STRAUSS, M. J. 2002a. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the 34th Annual ACM Symposium on the Theory of Computing* (Montreal, P.Q., Canada).

GILBERT, A. C., KOTIDIS, Y., MUTHUKRISHNAN, S., AND STRAUSS, M. J. 2001. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of the 27th International Conference on Very Large Data Bases* (Rome, Italy).

GILBERT, A. C., KOTIDIS, Y., MUTHUKRISHNAN, S., AND STRAUSS, M. J. 2002b. How to summarize the universe: Dynamic maintenance of quantiles. In *Proceedings of the 28th International Conference on Very Large Data Bases* (Hong Kong, China). 454–465.

GRAVANO, L., IPEIROTIS, P. G., JAGADISH, H., KOUDAS, N., MUTHUKSRISHNAN, S., AND SRIVASTAVA, D. 2001. Approximate string joins in a database (almost) for free. In *Proceedings of the 27th International Conference on Very Large Data Bases* (Rome, Italy).

GREENWALD, M. AND KHANNA, S. 2001. Space-efficient online computation of quantile summaries. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data* (Santa Barbara, CA).

GUHA, S., JAGADISH, H., KOUDAS, N., SRIVASTAVA, D., AND YU, T. 2002. Approximate XML joins. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data* (Madison, WI).

GUPTA, A. AND SUCIU, D. 2003. Stream processing of XPath queries with predicates. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (San Diego, CA).

INDYK, P. 2000. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science* (Redondo Beach, CA). 189–197.

INDYK, P. 2001. Algorithmic aspects of geometric embeddings. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science* (Las Vegas, NV).

INDYK, P., KOUDAS, N., AND MUTHUKRISHNAN, S. 2000. Identifying representative trends in massive time series data sets using sketches. In *Proceedings of the 26th International Conference on Very Large Data Bases* (Cairo, Egypt). 363–372.

IOANNIDIS, Y. E. AND POOSALA, V. 1999. Histogram-based approximation of set-valued query answers. In *Proceedings of the 25th International Conference on Very Large Data Bases* (Edinburgh, Scotland).

JOHNSON, W. B. AND LINDENSTRAUSS, J. 1984. Extensions of lipschitz mappings into Hilbert space. *Contemp. Math. 26*, 189–206.

KARP, R. M. AND RABIN, M. O. 1987. Efficient randomized pattern-matching algorithms. *IBM J. Res. Devel. 31*, 2 (Mar.), 249–260.

KNUTH, D. E. 1973. *The Art of Computer Programming (Vol. 1/Fundamental Algorithms).* Addison-Wesley, Reading, MA.

LAKSHMANAN, L. V. S. AND PARTHASARATHY, S. 2002. On efficient matching of streaming XML documents and queries. In *Proceedings of the 8th International Conference on Extending Database Technology* (EDBT'2002, Prague, Czech Republic).

MANKU, G. S. AND MOTWANI, R. 2002. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases* (Hong Kong, China). 346–357.

MOTWANI, R. AND RAGHAVAN, P. 1995. *Randomized Algorithms.* Cambridge University Press, Cambridge, U.K.

NOLAN, J. P. 2004. Stable distributions: Models for heavy-tailed data. Available online at `http://academic2.american.edu/-jpnolan/stable/stable.html`.

POLYZOTIS, N. AND GAROFALAKIS, M. 2002. Statistical synopses for graph-structured XML databases. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data* (Madison, WI).

POLYZOTIS, N., GAROFALAKIS, M., AND IOANNIDIS, Y. 2004. Approximate XML query answers. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (Paris, France).

SCHMIDT, A., WAAS, F., KERSTEN, M., CAREY, M. J., MANOLESCU, I., AND BUSSE, R. 2002. XMark: A benchmark for XML data management. In *Proceedings of the 28th International Conference on Very Large Data Bases* (Hong Kong, China).

SHAPIRA, D. AND STORER, J. A. 2002. Edit distance with move operations. In *Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching* (CPM'2002), Fukuoka, Japan). 85–98.

THAPER, N., GUHA, S., INDYK, P., AND KOUDAS, N. 2002. Dynamic multidimensional histograms. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data* (Madison, WI). 428–439.

UCHAIKIN, V. V. AND ZOLOTAREV, V. M. 1999. *Chance and Stability : Stable Distributions and their Applications.* VSP, Utrecht, The Netherland.

UKKONEN, E. 1992. Approximate string matching with q-grams and maximal matches. *Theoret. Comput. Sci. 92*, 191–211.

VITTER, J. S. AND WANG, M. 1999. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data* (Philadelphia, PA).

ZHANG, K. AND SHASHA, D. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput. 18*, 6 (Dec.), 1245–1262.

Received November 2003; revised July 2004; accepted November 2004