Performance Oriented Schema Matching

Khalid Saleem[1], Zohra Bellahsene[1], and Ela Hunt[2]

[1] LIRMM - UMR 5506 CNRS University Montpellier 2,
161 Rue Ada, F-34392 Montpellier
{saleem, bella}@lirmm.fr
[2] Department of Computer Science, ETH Zurich, CH-8092
hunt@inf.ethz.ch

Abstract. Semantic matching of schemas in heterogeneous data sharing systems is time consuming and error prone. Existing mapping tools employ semi-automatic techniques for mapping two schemas at a time. In a large-scale scenario, where data sharing involves a large number of data sources, such techniques are not suitable. We present a new robust mapping method which creates a mediated schema tree from a large set of input XML schema trees and defines mappings from the contributing schema to the mediated schema. The result is an almost automatic technique giving good performance with approximate semantic match quality. Our method uses node ranks calculated by pre-order traversal. It combines tree mining with semantic label clustering which minimizes the target search space and improves performance, thus making the algorithm suitable for large scale data sharing. We report on experiments with up to 80 schemas containing 83,770 nodes, with our prototype implementation taking 587 seconds to match and merge them to create a mediated schema and to return mappings from input schemas to the mediated schema.

1 Introduction

Schema matching relies on discovering correspondences between similar elements of two schemas. Several different types of schema matching tools [8, 9] have been studied, demonstrating their benefit in different scenarios. In data integration schema matching is of central importance [1]. The need for information integration arises in data warehousing, OLAP, data mashups [6], and work flows. Omnipresence of XML as a data exchange format on the web and the presence of metadata available in that format force us to focus on schema matching, and on matching for XML schemas in particular.

We consider schemas to be rooted, labeled trees. This supports the computation of contextual semantics in the tree hierarchy. The contextual aspect is exploited by tree-mining, making it feasible to use almost automated approximate schema matching [4] and integration in a large-scale scenario. The individual semantics of node labels have their own importance. We utilize linguistic matchers, based on tokenisation, and synonym and abbreviation tables, to extract the concepts hidden within them. The use of synonym and abbreviation tables is considered to be a form of user intervention.

Tree mining techniques extract similar sub tree patterns from a large set of trees and predict possible extensions of these patterns. A pattern starts with one node and is incrementally augmented. There are different techniques [11] which mine rooted, labeled, embedded or induced, ordered or unordered sub-trees. The function of tree mining is to find sub-tree patterns that are frequent in the given set of trees, which is similar to schema matching that tries to find similar concepts among a set of schema trees.

Our Contributions

a) Matching, merging and the creation of a mediated schema with semantically approximate mappings, in one algorithm which has good performance.
b) Use of tokenisation, abbreviation and synonym matching of label tokens, intuitively supporting the clustering of similar labels to minimize the search space.
c) Extension of a tree mining data structure [11] to schema matching, using ancestor/ descendant properties for quality contextual semantic matching.
d) Ability to produce element level [8] simple 1:1 mappings, and complex mappings, including 1:n(leaf mapped to non-leaf) and n:1(non-leaf mapped to leaf).
e) Experiments with real XML schema instances (OAGIS, XCBL[3]) showing performance appropriate for a large scale scenario.
f) Quality evaluation based on *precision*, showing that our method is reliable.

The remainder of the paper is organized as follows. Section 2 presents the related work in large-scale schema integration. Section 3 defines the concepts. In Section 4 we describe our approach, Performance Oriented Schema Matching, along with a running example. Section 5 presents the experimental evaluation. Section 6 gives a discussion, outlines future work and concludes.

2 Related Work

Nearly all schema-matching systems compare two schemas at a time and aim for quality matching but require significant human intervention. Several surveys [3, 8, 9] shed light on this aspect and show that extending the matching to data integration is time consuming and limited in scope. Large scale schema matching has been investigated in the web interface schema integration [5, 10] using data mining. Matching of two large bio-medical taxonomies [3, 7] was demonstrated using COMA++ [3] and Protoplasm [2].

There are numerous issues in the semantic integration of a large number of schemas. For example, Semantic Web, by definition, offers a large-scale dynamic environment where individual service providers are independent. In such a situation the mappings can never be exact, rather they are approximate [4, 5].

Performance is an open issue in schema matching [3, 8, 9]. The complexity of the schema matching task is proportional to the size of participating schemas

[3] OAGIS : http://www.openapplications.org/, XCBL : http://www.xcbl.org/

and the number of match algorithms employed, i.e. $O(nma)$, where n and m are node counts in the source and target schema and a is the number of algorithms applied. The quality of mappings depends on the type and number of matching algorithms and their combination strategy [2, 3].

One of the most recent matching and merging tool is Coma++ [3] which produces quality matches. Coma is a composite matcher which can reuse previous mappings. It uses user defined synonym and abbreviation tables, along with other matchers. Coma can map large schemas with the help of user input. The user can identify fragments of the schema to be mapped. This option is intended to manage the namespace/ include characteristics of XML schemas. However, human intervention in the schema mapping and merging process is needed. Systems like Coma, which produce mappings and no integrated schema, do not support automated data integration suitable for application environments with hundreds of schemas.

Semantically, a match between two nodes can be either an equivalence or a partial equivalence. In a partial match, the similarity is partial, e.g., an element Name = 'John M. Brown' in source schema is partially matched to LastName='Brown' and FirstName ='John' in the target, because Name also contains the MiddleInitial='M'. If there are several possible matchings of the source element to the mediated schema, the best/most correct match can be selected (manual in [2, 3]). The choice can depend upon match quality confidence computed at run time [2, 3, 8, 9]. We use a hybrid approach which automatically selects the best match and performs the binary integration of schemas using the ladder technique [1]. Our method caters both for the quality as well as performance in large scale scenarios, using domain specific linguistic matching (synonym and abbreviation oracles), clustering, and tree mining.

3 Preliminaries

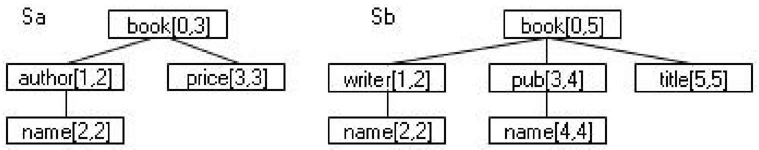


Fig. 1. Input Schema Trees S_a and S_b.

Semantic matching requires the comparison of concepts which are structured as schema elements. Node labels of schema elements are considered to be concepts and each element's contextual placement in the schema enhances the semantics of the concept. For example, in Fig. 1 Sb, the elements writer/**name** and publisher/**name** have similar labels but their contexts are different, which makes them conceptually disjoint. In an XML tree, the combination of the element label and the structural placement of the element produces the concept.

Def. 1 (Schema Tree) : A *schema tree* is a rooted, labeled tree [11]. A schema tree, S=(V,E), consists of V={0,1,. . . ,n}, a set of nodes, and E={(x,y) | x,y ∈ V}, a set of edges. One distinguished node r ∈ V is designated the root, and for all x ∈ V, there is a unique path from r to x. Further, *lab*:V → L is a labeling function mapping nodes to labels in L=$\{l_1, l_2, \ldots\}$, and *vt*:L → $\mathcal{P}(V)$ is a function which returns for each label l_i∈ L a set of nodes V_i ⊆V with labels similar to l_i.
Def. 2 (Node Scope) : Nodes x ∈ V are numbered according to their position in the pre-order traversal of the tree S (where the root is numbered 0, and x is numbered X). Let T(x) denote the sub-tree rooted at x, and let y be the rightmost leaf (or highest numbered descendant) under x, numbered Y. Then the scope of x is defined as scope(x)=[X,Y]. Intuitively, scope(x) is the range of nodes under x, and includes x itself (Fig. 1). The count of nodes in T(x) is Y-X+1.
Def. 3 (Label Semantics) : Label semantics corresponds to the conceptual meaning of the label (irrespective of context). It is a composition of concepts attached to the tokens making up the label. $C_l : l \to (C(t_1), \ldots, C(t_m))$ where m is the number of tokens making up the label.
Def. 4 (Node Semantics): Node semantics for x ∈ V, C_x, combines the semantics of the node label C$_{l_x}$ with its contextual placement in the tree, TreeContext(x), as follows [11]: $C_x : x \to (C_{l_x}, TreeContext(x))$. TreeContext of a node x is its scope (Def. 2).

3.1 Scope Properties

Scope properties describe the contextual placement of a node [11]. Property testing involves simple integer comparisons.
Unary Properties, given a node x with scope [X,Y]
Pr. 1: Leaf Node(x): X=Y, **Pr. 2**: Non-Leaf Node(x): X < Y.
Binary Properties, given x [X,Y], xd[Xd,Yd], xa[Xa,Ya], and xr[Xr,Yr]
Pr. 3: Descendant (x,xd), xd is a descendant of x: Xd>X ∧ Yd≤Y
Pr. 4: DescendantLeaf (x,xd) combines Pr. 1 and 3:
Xd>X ∧ Yd≤Y ∧ Xd=Yd
Pr. 5: Ancestor (x,xa), complements Pr. 3, xa is ancestor of x: Xa<X ∧ Ya≥Y
Pr. 6: RightHandSideNode (x,xr) with non-overlapping scope, xr is RHSNode of x: Xr>Y.

Example 3.1 : In Fig. 1 Sa, Pr. 1 for node **price**[3,3] defines it as a leaf. Pr. 2 for **author**[1,2] states that it is a non-leaf (an inner node).
Example 3.2 : The task is to find a mapping for Sa tree node author/**name** in Sb. In Fig. 1 Sb there are two nodes called **name**: [2,2] and [4,4]. Given synonymy between words **author** and **writer**, and top down traversal, Sa **author**[1,2] is already mapped to Sb **writer** [1,2], we perform the descendant node check on

nodes [2,2] and [4,4] with respect to **writer**[1,2]. Node [2,2] is a descendant of [1,2], using Pr. 3, and node [4,4] is not a descendant of [1,2], thus author/**name** is mapped to writer/**name**.

4 Our Approach

We assume that only schema trees are available as input. Our method accepts a set of schema trees and outputs the mediated schema tree and the corresponding mappings.

Def. 5 (Semantic Mediation)
INPUT: A set of schema trees $SSet$=$\{S_1$,$S_2 \ldots S_u\}$.
OUTPUTS:
a) A mediated schema tree S_m, which is a composition of all distinct concepts in $SSet$. $S_m = P^u_{i=1}(S_i)$, $P(S_i) = \{C_1\ \rho\ C_2\ \rho \ldots C_n\}$ includes all distinct concepts in S_i (Def. 4). P is the composition function and ρ denotes the composition operator.
b) A set of mappings $M = \{M_1, M_2, \ldots M_w\}$ from the concepts of input schema trees to the concepts in the mediated schema.

The mediated schema tree S_m is a composition of all nodes representing distinct concepts in the set of schemas. During the integration process if a node is not present in S_m, a new edge e' is created in S_m and a node is added to it.

4.1 Assumptions

We make the following assumptions, valid in domain specific schema integration (extended from [10]).
a) Schemas in the same domain contain the same domain concepts, but differ in structure and concept naming.
b) We select the input schema with the highest number of nodes as the initial mediated schema. Since each node represents a concept, this covers the maximum number of concepts. This choice minimizes the addition of new concepts (nodes not present in the mediated schema) to the mediated schema and improves performance.
c) Only one type of matching between two labels is possible. For example, author is a synonym of writer.
d) In one schema, different labels for the same concept are rarely present.
e) A node from the input schema is only matched to the set (cluster) of similarly labeled nodes present in the mediated schema.

4.2 Example of Schema Integration

We developed an algorithm which works in three steps. First, we perform *pre-mapping*. Schema trees are input to the system as a stream of XML and the node number and parent for each node, **node scope**, schema size, and schema depth are calculated. A listing of nodes and of distinct labels for each tree is constructed.

Next, a *linguistic matcher* identifies semantically similar node labels. The user can set the level of similarity of labels as A) Label String Equivalence, B) Label Token Set Equivalence (using abbreviation table), or C) Label Synonym Token Set Equivalence (synonym table). The matcher derives the meaning for each individual token and combines these meanings to form a label concept. Similar labels are clustered. Since each input node corresponds to its label object, this intuitively forms **clusters of similarly labeled nodes** within the group of schemas to be merged.

Table 1. Before Node Mapping

a. List of labels, ordered alphabetically

0	1	2	3	4	5	6	7	8
author	book	name	name	price	pub	title	writer	ROOT

b. Input Schema Nodes' Matrix : Row 1 is Sa and Row 2 is Sb

1,2,0	0,3,-1	2,2,1		3,3,0				
	0,5,-1	2,2,1	4,4,3		3,4,0	5,5,0	1,2,0	

c. Initial Mediated Schema, Sm, renumbered after adding ROOT to Sb

	1,6,0	3,3,2	5,5,4		4,5,1	6,6,1	2,3,1	0,6,-1

*Column entries show node scope and parent

Example 4.1 : Consider labels "POShipDate" and "PurchaseOrderDeliverDate". In the abbreviation table PO stands for *purchase order* and in the synonym table 'deliver'='ship'. This implies that the two labels are similar.

In the *integration and mapping part*, we first select the input schema tree with the highest number of nodes and designate it as the initial mediated schema (Section 4.1). Next, we take each schema in turn and merge it with the mediated schema, following the **binary ladder technique** highlighted in [1]. This requires matching, mapping and merging. Concepts from input schemas are mapped to the mediated schema.

The algorithm traverses the input schema depth-first, mapping parents before siblings. If a new concept is found, with no match in the mediated schema, a new concept node is created and added to the mediated schema as the right most child of the node in the mediated schema, to which the parent of current node is mapped. This new node is used as the target node in the mapping (Def. 5). The algorithm combines node label similarity and contextual positioning, calculated using properties defined in Section 3.1. Our example uses the two schemas shown in Fig. 1 where S_a and S_b are shown with information calculated during *pre-mapping*. A list of labels created in this traversal is shown in Table 1a. Nodes 2 and 3, with the same label 'name' but different parents (author and publisher)

Table 2. After Node Mapping

a. Labels List

0	1	2	3	4	5	6	7	8
author	book	name	name	price	pub	title	writer	ROOT

b. Mapping Matrix : Row 1 is Sa and Row 2 is Sb

1,2,0,**7**	0,3,-1,**1**	2,2,1,**2**		3,3,0,**4**				
	0,5,-1,**1**	2,2,1,**2**	4,4,3,**3**		3,4,0,**5**	5,5,0,**6**	1,2,0,**7**	

c. Final Mediated Schema

	1,7,0, **1.0,2.0**	3,3,2, **1.2,2.2**	5,5,4, **2.4**	7,7,1, **1.3**	4,5,1, **2.3**	6,6,1, **2.5**	2,3,1, **1.1,2.1**	0,7,-1

*Column entries show node scope, parent and **mapping**

are shown to be disjoint. The last label is a new label, ROOT, created by our algorithm.

Table 1b shows a matrix of size um, where u is the number of schemas and m the total number of distinct labels in all schemas (the length of the label list). A matrix row represents an input schema tree. Each non-null entry contains the node scope and parent node number. Each node is placed in the column which holds its label.

The larger schema tree S_b, see Fig. 1, is selected as the initial mediated schema S_m. ROOT is added to S_m, and the nodes are renumbered to reflect this. A list of size m (Table 1c) holds S_m, assuming the same column order as in Table 1b.

The node mapping algorithm takes the data structures in Table 1 as input, and produces mappings shown in Table 2b, and the integrated schema in Table 2c. In the process, the input schema S_a is mapped to mediated schema S_m. The mapping is read as the column number (Table 2b **mapping**) of node in the mediated schema. Saving mappings as column number gives us the flexibility to add new nodes to S_m, without disturbing the previous mappings. Scope values of some existing nodes are affected, cf. Table 1c and Table 2c, because of addition of new nodes (identified by Pr. 5 or 6; scope values adjusted accordingly), but column numbers of all previous nodes remain the same. Thus, intuitively, none of the existing mappings are affected.

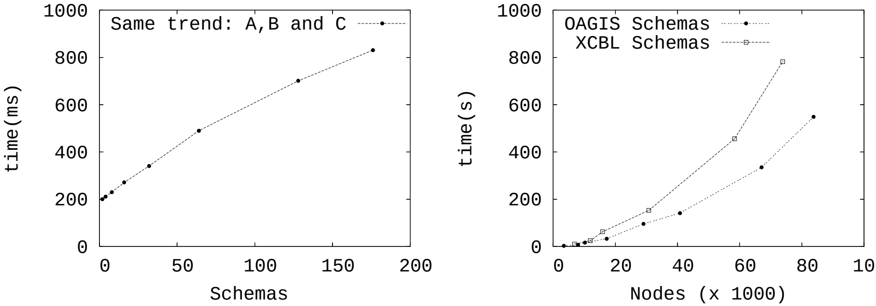
Node mapping for input schema tree S_a (Table 1b row 1) starts from the label ‘book’ with scope [0,3]. As it is a root node, with only one similar node ‘book’ [1,6] in S_m, mapping **1** is added in column 1 in S_a row, shown in bold in Table 2b, for node $S_a[0,3]$. This is now recorded in the mediated schema, see Table 2c, as **1.0** i.e., node 0 in Schema 1 i.e., S_a mapped to node 0 in S_m. Next node to map is $S_a.author[1,2]$, similar to $S_m.writer[2,3]$. Both nodes are internal and the ancestor check returns true since parent nodes of both are already mapped. The resulting mapping for label 0 is **7**. For node 2 with label ‘name’, there are two possibilities, nodes attached to label 2 (col. 2) and label 3 (col. 3). Descendant(name,writer) is true for node in column 2 and false for 3 by Pr. 3 (Example 3.2). Hence **2** is the correct map. The last node in S_a is price[3,3]. There is no match in S_m, so a new node is added to S_m, as an entry in the column with label 'price' in the mediated schema list (Table 2c). This node is created as the right most sibling of node in the mediated tree to which the parent node of current input node is mapped, i.e. 'book'. The scope and parent node link are adjusted for the new node and its ancestors, and a mapping is created from the input node to this newly created node.

Algorithm complexity: Given a set of input schemas S= $\{S_1, S_2, \ldots S_u\}$, we select as the mediated schema the schema tree with the highest number of nodes, max(N(S_i)) where N(S_i)returns the number of nodes in a schema. We match each node of each input schema to the mediated schema. The number of input schema nodes N_t is given by $N_t = \sum_{i=1}^{u} N(S_i)$. Therefore the complexity of Node Mapper algorithm is O($N_t N(S_m)$). This is quadratic in the size of schema set that is to be integrated. Our experiments confirm this complexity.

5 Experimental Evaluation

The experiments were performed on a PC, Pentium 4-M, 1.80 GHz, 768 MB RAM, running Windows XP, and Java 1.5. Three sets of schemas from different domains were used, with *ISN*, Integrated Schema Nodes, giving the schema size of the largest integrated schema.

1. Books: 176 synthetic schemas; Avg./Max/Min nodes 8/14/5, ISN 23
2. OAGIS: 80 real schemas; Avg./Max/Min nodes 1047/3519/26, ISN 70191
3. XCBL: 44 real schemas; Avg./Max/Min nodes 1678/4578/4, ISN 4803

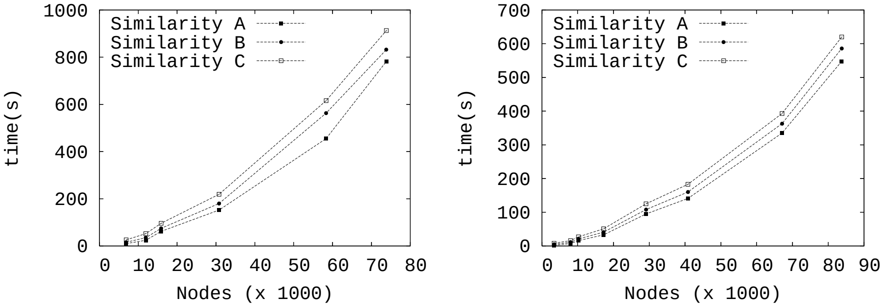


(a) Books schema integration, time against schema count

(b) OAGIS / XCBL schema integration, time against node count

Fig. 2. Small synthetic schemas are matched faster than complex real life schemas

Performance was evaluated in three label similarity scenarios: A) Label String Equivalence, B) Label Token Set Equivalence, and C) Label Synonym Token Set Equivalence. Figures 2 and 3 demonstrate the performance of our method. Our experiments show that the execution time reflects the number of



(a) Integration of XCBL Schemas (b) Integration of OAGIS Schemas

Fig. 3. Influence of matchers A, B, C on schema integration time

schemas to be integrated, and appears to be at worst quadratic in the number of nodes compared.

Fig. 2a shows a comparison of three kinds of matching: A, B, and C for sets of 2, 4, 8, 16, 32, 64, 128, 176 Books schemas. There is no difference in the performance of various matchers, which is possibly due to the fact that synthetic schemas vary little in their labels. Fig. 2b shows time in seconds for Domains 2 and 3. Fig. 3a shows the time (s) against the number of nodes processed, for the three similarity methods for XCBL. XCBL schemas are slower to match than OAGIS schemas, see Fig. 3b. This is due to the higher average number of nodes in XCLB schemas. It takes less than 600 seconds to match 80 OAGIS schemas, while 44 XCLB schemas require more than 800 seconds.

Table 3. Quality Evaluation. A, B, C are the label similarity levels, and schemas are at www.lirmm.fr/∼saleem/matching/schemas

Domain	PurchaseOrd		Books		OAGIS		XCBL	
Schema	S1	Sm	S1	Sm	S1	Sm	S1	Sm
Size	14	18	8	15	26	34	647	743
Precision A/B/C	0.29/0.36/1		0.5/0.5/1		0.77		0.96	

Since there is no established schema integration benchmark for a large scale scenario, including both schemas and mappings, it is impossible to carry out a full **quality comparison**. We evaluate the quality of our solution by looking at a random schema pair in the set and counting the number of correctly placed nodes in the integrated schema, *correctlyPlacedNodes/ allPlacedNodes*, as our algorithm will always add a node if it cannot find a match. We establish the precision measure by manual inspection of schemas, see Table 3 for results. Since our method takes the larger schema as the initial mediated schema, the smaller schema is integrated into the larger. Real domains schemas follow the same namespaces, with no abbreviations and synonym applicability, as established by manual inspection. Thus showing no variance in quality for the three label similarity cases. Synthetic domain schemas show fluctuation because of the abbreviated and synonym labels incorporated manually to study the algorithm.

6 Conclusions

We have introduced a novel technique based on tree mining, for schema matching, integrating and mapping of a large set of schemas. We have investigated its scalability with respect to time performance, in the context of approximate mapping. The experimental results demonstrate that our approach scales to hundreds of schemas and thousands of nodes. The linguistic matching of node labels uses tokenisation, abbreviations and synonyms. The matching strategy is hybrid, and optimized for schemas in tree format. Our algorithm provides an almost automated solution to the large scale mediation problem.

Our results point to significant future work. We are planning to investigate the application of our approach in P2P architectures, and enhancements to linguistic matching. Another issue for the future is a benchmark for schema mapping evaluation in a large scale schema integration scenario. To further benefit from tree mining, we are going to use it to identify co-relationships between subtrees within a forest of schema trees, which will help in identifying subsumptions and overlaps, for the discovery of n:m complex mappings.

References

1. C. Batini, M. Lenzerini, and S. B. Navathe. A comparitive analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
2. P. A. Bernstein, S. Melnik, M. Petropoulos, and C. Quix. Industrial-strength schema matching. *SIGMOD Record*, 33(4):38–43, 2004.
3. H.-H. Do and E. Rahm. Matching large schemas: Approaches and evaluation. *Information Systems*, 32(6):857–885, 2007.
4. A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Y. Halevy. Learning to match ontologies on the semantic web. *VLDB J.*, 12(4):303–319, 2003.
5. B. He, K. C.-C. Chang, and J. Han. Discovering complex matchings across web query interfaces: a correlation mining approach. In *KDD*, pages 148–157, 2004.
6. A. Jhingran. Enterprise information mashups: Integrating information, simply - keynote address. In *VLDB*, 2006.
7. P. Mork and P. A. Bernstein. Adapting a generic match algorithm to align ontologies of human anatomy. In *ICDE*, 2004.
8. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.
9. P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *J. Data Semantics IV*, pages 146–171, 2005.
10. W. Su, J. Wang, and F. Lochovsky. Holistic query interface matching using parallel schema matching. In *ICDE*, 2006.
11. M. J. Zaki. Efficiently mining frequent embedded unordered trees. *Fundamenta Informaticae 65*, pages 1–20, 2005.