Querying Tree-Structured Data Using Dimension Graphs

Dimitri Theodoratos[1] and Theodore Dalamagas[2]

[1] Dept. of Computer Science
New Jersey Institute of Technology
Newark, NJ 07102
dth@cs.njit.edu

[2] School of Electr. and Comp. Engineering
National Techn. University of Athens,
Athens, GR 15773
dalamag@dblab.ece.ntua.gr

Abstract. Tree structures provide a popular means to organize the information on the Web. Taxonomies of thematic categories, concept hierarchies, e-commerce product catalogs are examples of such structures. Querying multiple data sources that use tree structures to organize their data is a challenging issue due to name mismatches, structural differences and structural inconsistencies that occur in such structures, even for a single knowledge domain. In this paper, we present a method to query tree-structured data. We introduce dimensions which are sets of semantically related nodes in tree structures. Based on dimensions, we suggest dimension graphs. Dimension graphs can be automatically extracted from trees and abstract their structural information. They are semantically rich constructs that provide query guidance to pose and evaluate queries on trees. We design a query language to query tree-structured data. A key feature of this language is that queries are not restricted by the structure of the trees. We present a technique for evaluating queries and we provide necessary and sufficient conditions for checking query unsatisfiability. We also show how dimension graphs can be used to query multiple trees in the presence of structural differences and inconsistencies.

1 Introduction

Tree structures provide a popular means to organize the information on the Web. Taxonomies of thematic categories, concept hierarchies, e-commerce product catalogs are examples of such structures. Since the XML language [3] has become the standard data exchange format on the Web, organizing data in tree structures has been vastly established. Even if data is not stored natively in tree structures, export mechanisms make data publicly available in tree structures to enable its automatic processing by programs, scripts, and agents on the Web [11]. Querying capabilities on these structures are provided through path expression queries. Such queries are formed using some of the query languages proposed in the literature like XPath [4] and XQuery [5].

Querying multiple data sources that use tree structures to organize their data is a challenging issue due to name mismatches, structural differences and structural inconsistencies that occur in such structures, even for a single knowledge domain. Name mismatches appear because tree structures lack semantic information. For example, laptop computers might be referred to as notebooks in one product catalog but as portables in another catalog. In this paper, we do not focus on this issue and we assume that it is resolved using well-known schema matching techniques [20]. Structural differences and, far more important, structural inconsistencies appear because of the different possible ways of organizing the same data in tree structures. For example, a structural difference exists when a category appears in a product catalog but does not appear in another. A structural inconsistency appears when a product catalog for notebooks classifies new, SONY notebooks with 10in display in the path /notebooks/new/SONY/10in, while another catalog classifies the same products in the path /SONY/notebooks/10in/new.

A naive approach to cope with structural differences and inconsistencies when querying multiple tree structures is to generate different versions of the initial query, considering different subsets of nodes involved in its path expressions and their different orderings. Clearly this is not efficient due to the large number of queries that need to be generated. Another approach is to set up a global structure and define mapping rules between this structure and the local structures [13]. Such approaches require extensive manual effort, since the global schema is difficult to construct and the rules should be hard-coded in the application.

In this paper, we suggest a novel approach to query tree structured data. Our approach exploits semantic information for nodes of the trees which are called here value trees. We introduce the concept of a dimension that groups together semantically related values (nodes). The different dimensions of a value tree are related through precedence relationships incurred by the parent-child and ancestor-descendant relationships of their nodes. We capture these precedence relationships between dimensions of a value tree into the concept of a dimension graph for a value tree. Besides abstracting structural information of value trees, dimension graphs provide also semantic guidance in posing and evaluating queries. Query conditions involve dimensions, and thus query formulation is not dependent on the structure of value trees. The system uses the dimension graph of the value tree to identify orderings of the values that can possibly exist in the value tree. Only these value orderings will be used to compute the answer of the query on the value tree. This step of the computation of the query answer is performed before the query evaluation reaches the value tree which is, in general, much larger than its dimension graph.

Contribution. The main contributions of the paper are the following:

- We introduce dimensions to record semantic information for the nodes of value trees and dimension graphs to capture structural information on value trees. Dimension graphs can be automatically extracted from value trees.

- We design a query language to query value trees. Queries are not cast on the structure of a specific value tree, since they are issued on their dimensions. The user can optionally specify parent-child and/or ancestor-descendent relationships between dimensions in a query.
- We show how queries can be evaluated on value trees, making use of dimension graphs to determine orderings of dimensions that can possibly generate non-empty answers. These dimension orderings are then used for generating path expressions that are evaluated on value trees.
- We introduce the concept of query unsatisfiability which identifies a query on a dimension graph that has an empty answer on any value tree underlying this dimension graph. We provide necessary and sufficient conditions for a query to be unsatisfiable.
- Finally, we show how dimension graphs can be used to query multiple value trees in the presence of structural differences and inconsistencies.

Outline. The rest of the paper is organized as follows. The next section discusses related work. In Section 3, we introduce dimensions and we define dimension graphs for value trees. Section 4 presents the query language used to pose queries on dimension graphs. It also shows how queries can be checked for unsatisfiability and how they are evaluated on the underlying value trees. Finally, Section 5 concludes the paper and presents further work. Due to lack of space, proofs of propositions are omitted. They can be found in the full version of the paper.

2 Related Work

Many systems support query evaluation of multiple data sources, using a predefined global structure and defining mapping rules between this structure and the local structures used in the sources. The Xyleme system[13] copes with the problem of integrating XML data sources using XML views. In the Agora system[18], XQuery expressions over a given global XML schema are translated to SQL queries on local data sources. In [7], query evaluation is based on mapping rules from global to local schemas in the form of path-to-path correspondences. In [12], YAT queries are posed on a global schema and evaluated in the data sources using YAT mapping rules. In [19], Xpath queries are formed and reformulated to queries on the local catalogs, given a pre-defined DTD. Our approach differs than the aforementioned techniques in that it does not require the manual definition of hard-coded mapping rules between the virtual tree structure and the local structures.

Relevant to our work are also techniques where schema descriptions are automatically extracted from local data sources. XClust [17] generates DTDs to act as global schemas, applying clustering methods to detect similar DTDs prior to their integration. Techniques that extract DTDs from collections of XML documents are also presented in [14]. In [8], a grammar-based model is used to integrate DTDs. Contrary to our approach, these papers do not deal with query evaluation.

Schema-based descriptions for data with little or no apparent structure have also been suggested for semistructured databases [6]. Dataguides are introduced in [15]. They are structural summaries for semistructured data, useful for formulating queries, storing statistics about paths and nodes, and enabling query optimization. In [10], graph schemas are introduced to formulate, optimize and decompose queries for semistructured data. These approaches do not provide a direct solution to the problem of structural inconsistencies and differences in data sources that we address here. Further, they are purely syntactic. In contrast to our approach, they do not exploit semantic information.

Integrating tree-structured data is also a popular issue in e-commerce applications [9, 16]. Facet classification hierarchies [1, 2] also exploit sets of semantically related categories. In [21], the authors present faceted taxonomies for Web catalogs. None of these works suggest query evaluation techniques.

3 Data Model

In this section we present a data model for tree-structured data. We introduce a type of trees, called value trees, to represent tree-structured data. We also discuss the notion of a dimension, based on which a partitioning can be enforced on value trees.

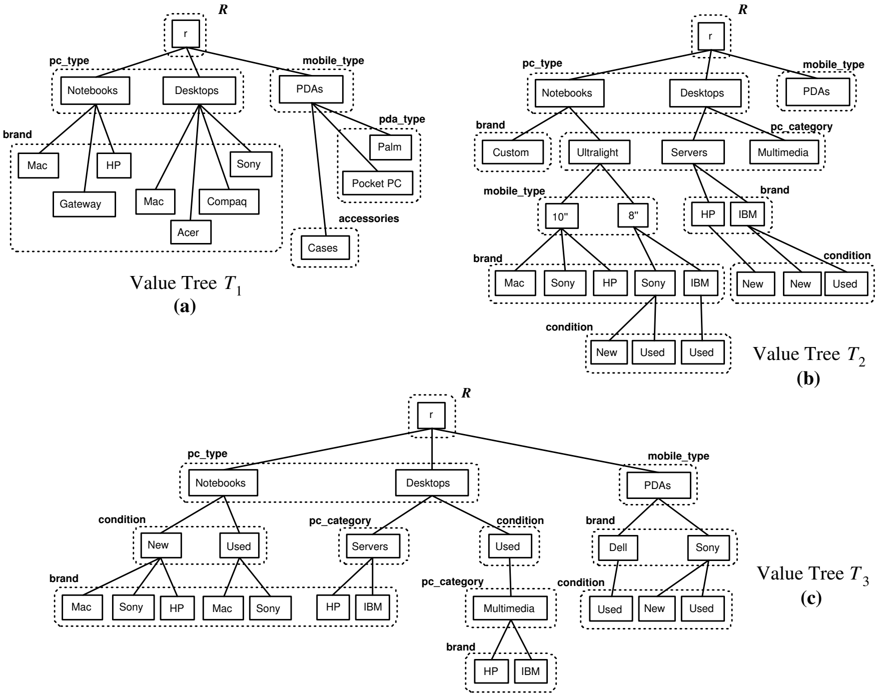
3.1 Value Trees and Dimensions

We assume a set of values V that includes a special value r. The elements of V are used to build value trees.

Definition 1. A *value tree* is a rooted node-labeled tree T, such that:
(a) Each node label in T belongs to V.
(b) Value r labels only the root of T.
(c) There are no sibling nodes in T labeled by the same value. □

Figure 1 shows examples of value trees T_1, T_2 and T_3 (for the moment, the dotted labeled rectangles that group the nodes should be ignored). These value trees are parts of taxonomies used to categorize products related to computer equipment. The same value may label multiple nodes in a value tree. For example, value HP labels two nodes in T_2. Notice that there are structural differences and inconsistencies between value trees T_1, T_2 and T_3, although they refer to the same knowledge domain. For example, there are nodes labeled Multimedia or Servers in T_2 and T_3, even though no such nodes appear in T_1. Also, a node labeled Used is a child of a node labeled Sony in T_2, although the opposite holds in T_3. Note that we assume that naming mismatches have been resolved. For instance, nodes labeled by the same value in different trees refer to the same real world concept.

Values in set V can be grouped to form dimensions. Intuitively, a dimension is a set of semantically related values. For instance, values Mac, Acer and Compaq can be interpreted as values of a dimension brand. A semantic interpretation of



Value Tree T_1
(a)

Value Tree T_2
(b)

Value Tree T_3
(c)

Fig. 1. Value trees T_1, T_2 and T_3

values is imposed by a user. A dimension can also be seen as a property with values.

Definition 2. Let V be a set of values that includes a specific value r. A *dimension set* over V is a partition $\mathcal{D}$ of V that includes a set R whose single element is value r. Each element of $\mathcal{D}$ is called *dimension*. □

Figure 2 shows a dimension set $\mathcal{D}$ and the names of its dimensions. We use these dimensions and the value trees of Figure 1 as a running example in this paper.

A dimension set also partitions the nodes of a value tree. We are interested in value trees where every path from the root to a leaf involves values from distinct dimensions. To describe this type of value trees we introduce the concept of *tree conformity* with respect to a dimension set.

Definition 3. Let $\mathcal{D}$ be a dimension set over a value set V. A value tree T *conforms* to $\mathcal{D}$ if and only if there are no two nodes on a path in T labeled by values that belong to the same dimension in $\mathcal{D}$. □

Consider for example the value trees T_1, T_2 and T_3 of Figure 1. Dotted rectangles labeled by dimensions are used to show the partitioning of nodes into dimensions. The same dimension might label different rectangles in a value tree.

Dimension Set D = { R, pc_type, brand, mobile_type, pda_type, accessories, pc_category, condition }

Dimensions:
pc_type = { Notebooks, Desktops }
brand = { Mac, Sony, HP, IBM, Gateway, Acer, Compaq }
mobile_type = { PDAs, 10", 8" }
pda_type = { Palm, Pocket_PC }
accessories = { Cases }
pc_category = { Ultralight, Multimedia, Server }
condition = { New, Used }

Fig. 2. A dimension set and its dimensions

In this case, this dimension comprises the nodes confined by all these rectangles. Dimension pc_type in T_1 refers to types of personal computers and includes nodes labeled by values Desktops and Notebooks. Dimension brand in T_3 refers to brand names and includes nodes labeled Mac, Sony, HP, IBM and Dell. All trees T_1, T_2, and T_3 conform to the dimension set $\mathcal{D}$ shown in Figure 2.

Nodes labeled by values of the same dimension need not be in the same level of a value tree. For example, in T_2, the nodes labeled 10" and 8" of dimension mobile_type are not in the same level as the node labeled PDAs of the same dimension. A value of a dimension may not appear in a value tree. For example, the value Ultralight of dimension pc_category does not appear in value tree T_3 nor in T_1, although it appears in T_2. Further, a dimension may have no value in a value tree. For instance, no value of pc_category appears in T_1.

In the following we assume that a dimension set $\mathcal{D}$ is given and all value trees conform to $\mathcal{D}$.

3.2 Dimension Graphs

Values of one dimension can label children or descendants of nodes labeled by values of any other dimension in a value tree. However, there are cases where values of one dimension do not label descendants of nodes labeled by values of some other dimension. For example, none of the values Pocket_PC and Palm of dimension pda_type labels a descendant of the nodes labeled by the value Desktops or Notebooks of dimension pc_type in the value tree T_1 of Figure 1. To capture this type of relationship between dimensions in a value tree, we introduce the concept of a dimension graph. Dimension graphs can be automatically extracted from value trees and abstract their structural information. Moreover, they provide semantic query guidance to pose and evaluate queries on value trees (see subsequent sections). Before we give the formal definition of a dimension graph with respect to a value tree, we define dimension graphs as general structures and we present the notion of a dimension precedence.

Definition 4. A *dimension graph* over dimension set $\mathcal{D}$ is a directed graph whose nodes are dimensions in $\mathcal{D}$. □

A *path* in a dimension graph is a sequence $D_1, \ldots, D_k$ of distinct nodes such that there is a directed edge from D_i to D_{i+1}, where $1 \leq i \leq k-1$.

Definition 5. Let T be a value tree over dimension set $\mathcal{D}$. A dimension $D_i \in \mathcal{D}$ *precedes* a dimension $D_j \in \mathcal{D}$ in T if and only if there are nodes n_i and n_j in T labeled by values $v_i \in D_i$ and $v_j \in D_j$, respectively, such that n_j is a child node of n_i in T. □

For example, dimension pc_type precedes dimension brand in T_1 of Figure 1, since a node labeled Mac (a value of brand) is a child of a node labeled Desktops (a value of pc_type).

Based on the definitions of dimension graphs as general structures and the notion of dimension precedence, we proceed to define formally dimension graphs with respect to a value tree.

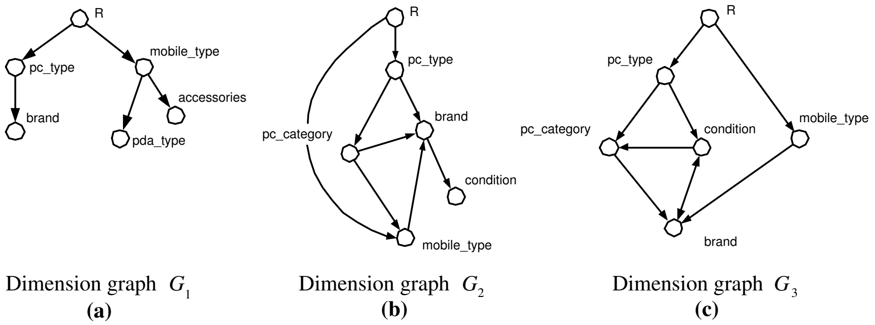
Definition 6. Let T be a value tree over dimension set $\mathcal{D}$. A *dimension graph* of T is a dimension graph (N, E), where N is a set of nodes and E is a set of edges defined as follows:

(a) There is a node D in N if and only if there is a value in T that belongs to dimension D.

(b) There is a directed edge in E from node D_i to node D_j if and only if dimension D_i precedes dimension D_j in T.

If $\mathcal{G}$ is a dimension graph of a value tree T, we say that T *underlies* $\mathcal{G}$. □

Consider for example the value trees T_1, T_2 and T_3 of Figure 1. Figure 3 shows the dimension graphs $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_3$ of T_1, T_2 and T_3, respectively. There is an edge from dimension mobile_type to dimension pda_type in $\mathcal{G}_1$, since a node labeled Palm (a value of pda_type) is a child of a node labeled PDAs (a value of mobile_type) in value tree T_1. Looking at the lower left part of value tree T_3, we note that a node labeled Mac (a value of brand) is a child of a node labeled New



Dimension graph G_1
(a)

Dimension graph G_2
(b)

Dimension graph G_3
(c)

Fig. 3. Dimension Graphs

(a value of condition). However, looking at the lower right part of T_3, a node labeled New is a child of a node labeled Dell (another value of dimension brand). Thus, dimension brand precedes dimension condition and vice versa. As a result, there is an edge from dimension condition to dimension brand and an edge from brand to condition in $\mathcal{G}_3$ which are compactly shown in the figures by a double headed edge.

The dimension graph of a value tree has a particular form. The following proposition describes some of its properties.

Proposition 1. Consider a dimension graph $\mathcal{G}$ of a value tree T over a dimension set $\mathcal{D}$. Let $v_1, \ldots, v_k$ be values from the distinct dimensions $D_1, \ldots, D_k \in \mathcal{D}$, respectively. If $v_1, \ldots, v_k$ label, in that order, nodes on a path in T, then $D_1, \ldots, D_k$ appear in that order on a path from the root in $\mathcal{G}$. □

4 Queries

We present in this section a simple query language and we outline how queries can be evaluated. Our intention is not to provide a full-fledged language. For instance, it does not include selection predicates. Our goal is to show how dimensions can be used to query value trees. Queries in this language are defined on dimension graphs. Roughly speaking, a user poses a query by annotating some dimensions in a dimension graph with permissible sets of values. The answer comprises root-to-leaf paths on the underlying value tree that involve one value from each of these value sets. An interesting feature of the language is that the user has the choice of not specifying or partially specifying parent-child and ancestor-descendant relationships between the annotated dimensions in a query. The system can identify possible orderings of dimensions in the paths of the answer based on the dimension graph only. These orderings are used as patterns for constructing the path expressions that compute the answer of the query on the underlying value tree. All the other orderings of dimensions are excluded from consideration before the computation of the query answer reaches the value tree.

4.1 Syntax

A query on a dimension graph comprises annotations of the graph dimensions with sets of values and specifications of precedence relationships between the graph dimensions.

Definition 7. Let $\mathcal{G}$ be a dimension graph over a dimension set $\mathcal{D}$. A *query* Q on $\mathcal{G}$ is a pair $(\mathcal{A}, \mathcal{P})$, where:

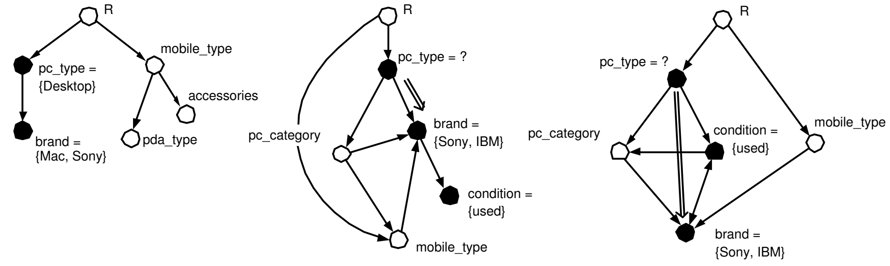
(a) $\mathcal{A}$ is a set of expressions of the from $D_i = A_i$, where D_i is a dimension in $\mathcal{G}$ different than R, and A_i is a set of values of dimension D_i or a question mark ("?"). If $D_i = A_i$ belongs to $\mathcal{A}$ we say that D_i is *annotated* in Q and A_i is called *annotation* of D_i in Q. Even if not present in $\mathcal{A}$, dimension R is assumed to be an annotated dimension, annotated with the singleton $\{r\}$. A dimension can be annotated only once in a query.

(b) $\mathcal{P}$ is a set of *precedence relationships* which are expressions of the form $D_i \rightarrow D_j$ or $D_i \Rightarrow D_j$, where D_i and D_j are annotated dimensions of Q.

Sets $\mathcal{A}$ and $\mathcal{P}$ can be empty. □

We graphically represent a query $Q = (\mathcal{A}, \mathcal{P})$ on a dimension graph $\mathcal{G}$ by labeling its nodes by their annotations in $\mathcal{A}$ and by adding to it a single (resp. double) arrow from node D_i to node D_j for every precedence relationship $D_i \rightarrow D_j$ (resp. $D_i \Rightarrow D_j$) in $\mathcal{P}$. Note that arrows are different than directed edges. The unqualified word "arrow" refers indiscreetly to a single or double arrow.

Consider for instance the dimension graphs $\mathcal{G}_1, \mathcal{G}_2$, and $\mathcal{G}_3$ of Figure 3. Figure 4 shows the graphical representation of different queries on these dimension graphs. Annotated nodes are shown in the figures with black circles. Precedence relationships are shown with single or double arrows from one node to another.



Query Q_1 on G_1
(a)

Query Q_2 on G_2
(b)

Query Q_3 on G_3
(c)

Fig. 4. Graphical Representation of Queries

Figure 4(a) represents query $Q_1 = (\mathcal{A}_1, \mathcal{P}_1)$ on dimension graph $\mathcal{G}_1$, where $\mathcal{A}_1 = \{\textsf{brand} = \{\textsf{Mac, Sony}\}, \textsf{pc\_type} = \{\textsf{Desktops}\}\}$ and $\mathcal{P}_1 = \emptyset$. In Q_1 we do not specify any precedence relationships between the annotated notes.

In the following we often identify a query with its graphical representation. Figures 4(b) and (c) represent queries Q_2 and Q_3. A double arrow from node pc_type to brand denotes the precedence relationship $\{\textsf{pc\_type} \Rightarrow \textsf{brand}\}$ in Q_2.

4.2 Semantics

The answer of a query on a value tree T is a set of root-to-leaf paths in T compactly represented as a subtree of T.

Definition 8. Let $\mathcal{G}$ be a dimension graph of a value tree T over a dimension set $\mathcal{D}$, and Q be a query on $\mathcal{G}$. The *answer* of Q on T is the maximal[3] subtree T' of T such that:

(a) T' and T have the same root r.

[3] Maximality is meant with respect to the number of nodes or edges.

(b) Every leaf node of T' is a leaf node of T.

(c) Every path from the root to a leaf node in T' includes one value from every value set annotating a node in Q.

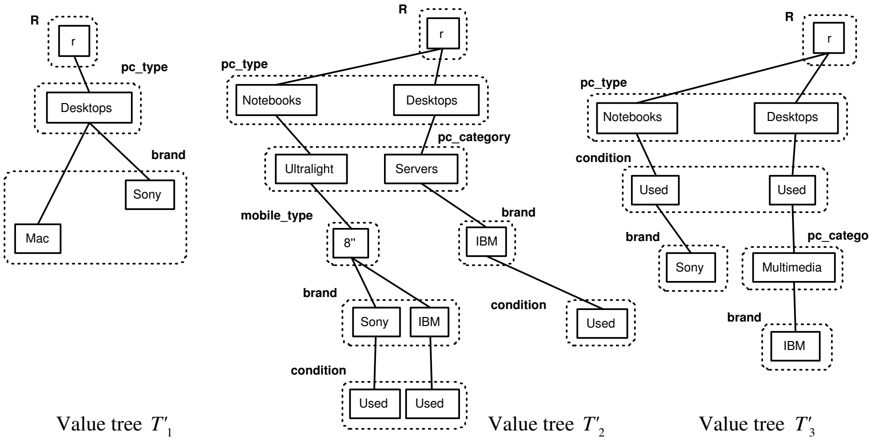
(d) Every path from the root to a leaf node in T' includes one value from every dimension annotated with a question mark in Q.
Therefore, for every annotated node (with a value set or a question mark) in Q, there is one value for the corresponding dimension appearing in every path from the root to a leaf node in T'.

(e) For every path p from the root to a leaf node in T', and for every precedence relationship $D_i \rightarrow D_j$ (resp. $D_i \Rightarrow D_j$) in Q, the value for D_j is a child (resp. descendent) of the value for D_i in p.

If there is no such a subtree T', we say that the answer of Q on T is *empty*. Symbol ϵ denotes an empty answer. □

Annotating a node with a "?" in a query is different than not annotating this node at all. In contrast to a non-annotated node, a node that is annotated with a "?" places a value of the corresponding dimension in every root-to-leaf path in the answer of the query.

Consider the queries Q_1, Q_2 and Q_3 on the dimension graphs $\mathcal{G}_1, \mathcal{G}_2$, and $\mathcal{G}_3$, respectively, graphically shown in Figure 4. Consider also the value trees T_1, T_2 and T_3 of Figure 1. Figure 5 shows the answers T'_1, T'_2 and T'_3 of Q_1, Q_2 and Q_3 on T_1, T_2 and T_3, respectively.



Value tree T'_1
(a)

Value tree T'_2
(b)

Value tree T'_3
(c)

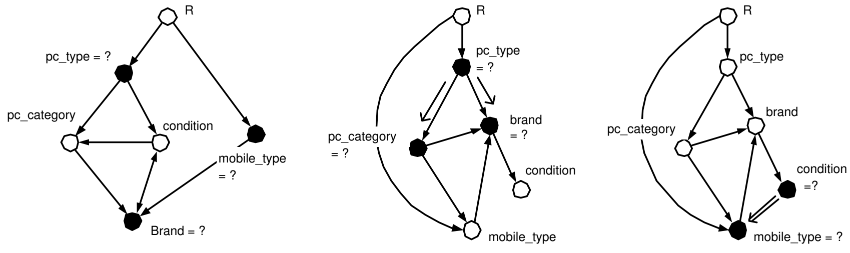
Fig. 5. Query Answers

Further, consider the query $Q_4 = (\mathcal{A}_4, \mathcal{P}_4)$, where: $\mathcal{A}_4 = \{\textsf{pc\_type} = \{\textsf{Desktops}\}, \textsf{brand} = \{\textsf{HP, Gateway}\}\}$, and $\mathcal{P}_3 = \{\textsf{pc\_type} \rightarrow \textsf{brand}\}$ on the dimension graph $\mathcal{G}_1$ shown in Figure 3. In the value tree T_1 shown in Figure 1(a) there are values of dimension brand that are children of values of dimension pc_type. However, there is no root-to-leaf path that involves values Desktops and HP, or Desktops and Gateway. Therefore, the answer of Q_4 on T_1 is empty.

4.3 Unsatisfiable Queries

A query on a dimension graph $\mathcal{G}$ is called *unsatisfiable* if its answer is empty on every value tree underlying $\mathcal{G}$. Otherwise, it is called *satisfiable.* Detecting the unsatisfiability of a query saves its evaluation on a value tree (which, in any case, produces an empty answer.) In general, this value tree is much larger than its dimension graph which might be needed for detecting the unsatisfiability of the query. The graphical representation of a query provides some intuition on unsatisfiable queries.

Consider the dimension graphs $\mathcal{G}_2$ and $\mathcal{G}_3$ of Figure 3, and the queries Q_5 on $\mathcal{G}_3$, and Q_6 and Q_7 on $\mathcal{G}_2$ graphically represented in Figure 6. These queries are unsatisfiable.



Query Q_5 on G_3 **(a)**

Query Q_6 on G_2 **(b)**

Query Q_7 on G_2 **(c)**

Fig. 6. Unsatisfiable Queries

In query Q_5 of Figure 6(a), there is no path from the root of $\mathcal{G}_3$ that involves all the annotated nodes. By Proposition 1 there is no root-to-leaf path in a value tree underlying $\mathcal{G}_3$ that involves values for the annotated dimensions in Q_5.

In query Q_6 of Figure 6(b), there is a path from the root of $\mathcal{G}_2$ through all the annotated nodes (e.g. the path (R, pc_type, pc_category, brand)). However, there are two outgoing single arrows from the same node (node pc_type). Clearly, no two values can be children of the same node in a root-to-leaf path of a value tree underlying $\mathcal{G}_2$.

In query Q_7 of Figure 6(c), there is also a path from the root of $\mathcal{G}_2$ through all the annotated nodes (e.g. the path (R, mobile_type, brand, condition)). However, there is a double arrow from node condition to node mobile_type in Q_3 and no path from node condition to node mobile_type in $\mathcal{G}_2$. By Proposition 1 there is no root-to-leaf path in a value tree underlying $\mathcal{G}_2$ that involves a value for dimension condition preceding a value for dimension mobile_type.

More generally, we can show the following result that provides sufficient conditions for a query to be unsatisfiable.

Proposition 2. A query Q on a dimension graph $\mathcal{G}$ is *unsatisfiable* if one of the following conditions holds:

(a) Arrows in Q form a directed cycle.
(b) There are precedence relationships $D \rightarrow D_i$ and $D \rightarrow D_j$ or precedence relationships $D_i \rightarrow D$ and $D_j \rightarrow D$ in Q ($D_i \neq D_j$).
(c) There is a precedence relationship $D_i \rightarrow D_j$ in Q but no edge from node D_i to node D_j in $\mathcal{G}$.
(d) There is a precedence relationship $D_i \Rightarrow D_j$ in Q but no edge from node D_i to node D_j in the *transitive closure* of $\mathcal{G}$ (in other words, no path from node D_i to node D_j in $\mathcal{G}$).
(e) The annotated nodes in Q are not on a path from the root of $\mathcal{G}$. □

In order to provide necessary conditions for query unsatisfiability, we introduce the concept of an answer path of a query.

Definition 9. Let Q be a query on a dimension graph $\mathcal{G}$. An *answer path* of Q in $\mathcal{G}$ is a path p in $\mathcal{G}$ from the root of $\mathcal{G}$ such that:

(a) All the annotated dimensions in Q are on p, and p ends on an annotated dimension of Q.
(b) If there is a precedence relationship $D_i \rightarrow D_j$ (resp. $D_i \Rightarrow D_j$) in Q, then D_j is a child (resp. descendent) of D_i in p. □

Consider for instance the query Q_2 on dimension graph $\mathcal{G}_2$ and the query Q_3 on dimension graph $\mathcal{G}_3$, which are shown in Figures 4(b) and 4(c), respectively. One can identify the following answer paths for query Q_2 in $\mathcal{G}_2$:

R, pc_type, brand, condition
R, pc_type, pc_category, brand, condition
R, pc_type, pc_category, mobile_type, brand, condition

The answer paths for query Q_3 in $\mathcal{G}_3$ are:

R, pc_type, condition, brand
R, pc_type, condition, pc_category, brand
R, pc_type, pc_category, brand, condition

The following proposition provides necessary and sufficient conditions for a query to be unsatisfiable.

Proposition 3. A query Q on a dimension graph $\mathcal{G}$ is *unsatisfiable* if and only if there is no answer path of Q in $\mathcal{G}$. □

4.4 Query evaluation

When evaluating a query, we first check it for satisfiability. If a query is satisfiable, we proceed to compute its answer on a value tree in three steps. In the first step, we compute all the answer paths of the query. In the second step, we generate path expressions based on the answer paths. In the third step we evaluate the path expressions on the value tree and compose the answer of the query.

To represent path expressions, we use a notation similar to that of XPath [4]. The fragment of XPath we use involves node names (v_i), child axis (/), descendant axis (//), wildcards ($*$), unions ($|$). The expression $(v_1| \dots |v_m)$ represents any node name in the set $\{v_1, \dots, v_m\}$. For a dimension D, we use the expression $*_D$ as an abbreviation for the expression $(v_1| \dots |v_n)$, where $\{v_1, \dots, v_n\} = D$.

Given an answer path, we construct a corresponding path expression as follows. Let $R, D_1, \dots, D_k$ be an answer path of a query Q. The corresponding path expression has the form $r/\theta_1/ \dots /\theta_k$, where, for $i = 1, \dots, k$,

$$\theta_i = \begin{cases} (v_1| \dots |v_m) & \text{if } D_i \text{ is annotated with the value set } \{v_1, \dots, v_m\} \\ *_{D_i} & \text{if } D_i \text{ is annotated with a "?" or if } D_i \text{ is not annotated} \end{cases}$$

Notice that even though nodes annotated with a "?" are treated the same way as non-annotated ones in the construction of path expressions for a query, they affect differently the answer of a query since they are taken into account in the identification of answer paths for that query.

Before showing what the result of a path expression on a value tree is, we introduce the concept of a merge of a set of value trees (or paths). Let $T_1, \dots, T_k$ be a set of value trees having the same root r. The *merge* of $T_1, \dots, T_k$, denoted $T_1 \cup \dots \cup T_k$, is a minimal[4] value tree which has $T_1, \dots, T_k$ as subtrees. It is not difficult to see that this value tree is unique.

We show now what is the result of a path expression on a value tree. Let e be a path expression and T be a value tree. Let also P be the set of paths from the root of T to the leafs of T that satisfy e. The result $res(e, T)$ of a path expression e on a value tree T is the value tree $\bigcup_{p \in P} p$. Note that the result of a path expression is different than the result of the same XPath expression. The result of a path expression is a value tree while the result of the same XPath expression is a set of nodes [4]. We can use XQuery [5] to compute the result of a path expression as it is defined here.

The answer of a query on a value tree can be computed by merging the results of its path expressions on the value tree. Let $E = \{e_1, \dots, e_n\}$ be the set of path expressions constructed from all the answer paths of a query Q. The answer of Q on a value tree T is the value tree $\bigcup_{i \in [1,n]} res(e_i, T)$.

As an example consider the query Q_2 on dimension graph $\mathcal{G}_2$, which is shown in Figure 4(b). The answer paths for Q_2 in $\mathcal{G}_2$ are shown in Section 4.3. These answer paths generate the following path expressions:

$r/*_{\mathsf{pc\_type}}/(\mathsf{Sony}|\mathsf{IBM})/\mathsf{Used}$
$r/*_{\mathsf{pc\_type}}/*_{\mathsf{pc\_category}}/(\mathsf{Sony}|\mathsf{IBM})/\mathsf{Used}$
$r/*_{\mathsf{pc\_type}}/*_{\mathsf{pc\_category}}/*_{\mathsf{mobile\_type}}/(\mathsf{Sony}|\mathsf{IBM})/\mathsf{Used}$

Evaluating these path expressions on the value tree T_2 of Figure 1(b), one can see that the result of the first path expression is an empty value tree. In

[4] Minimality is meant with respect to the number of nodes or edges.

contrast, the second path expression contributes one path, while the third one contributes two paths to the answer of Q_2 on T_2 (Figure 5(b)).

Consider also the query Q_3 on dimension graph $\mathcal{G}_3$, which is shown in Figure 4(c). The answer paths for Q_3 in $\mathcal{G}_3$ are shown in Section 4.3. They generate the following path expressions:

$r/*_{\mathsf{pc\_type}}$ /Used/(Sony|IBM)
$r/*_{\mathsf{pc\_type}}$ /Used/ $*_{\mathsf{pc\_category}}$ /(Sony|IBM)
$r/*_{\mathsf{pc\_type}}$ / $*_{\mathsf{pc\_category}}$ /(Sony|IBM)/Used

Of those path expressions, evaluating the third one on the value tree T_3 of Figure 1(c) results in an empty value tree. Only the first two contribute paths to the answer of Q_3 on T_3 (Figure 5(c)).

4.5 Querying multiple value trees

Consider the value trees $T_1, \ldots, T_n$ over a dimension set $\mathcal{D}$ and let $\mathcal{G}_1, \ldots, \mathcal{G}_n$ be their dimension graphs respectively. The dimension graphs $\mathcal{G}_1, \ldots, \mathcal{G}_n$ are not necessarily the same. In order to query the value trees $T_1, \ldots, T_n$ together, we need a "global" dimension graph. Such a graph $\mathcal{G}$ can be constructed by merging the dimension graphs $\mathcal{G}_1, \ldots, \mathcal{G}_n$. A query Q on $\mathcal{G}$ is defined on a dimension graph $\mathcal{G}_i$ if it does not involve dimensions that occur in $\mathcal{G}$ but not in $\mathcal{G}_i$. Otherwise, it is not defined on $\mathcal{G}_i$ and it returns no answers. If Q is defined on $\mathcal{G}_i$, it can be checked for consistency and evaluated as described in the previous sections. Notice that a query on the global dimension graph $\mathcal{G}$ can be applied to any of the $\mathcal{G}_i$s without the use of mapping rules.

5 Conclusion

We presented a method for querying tree structures, called value trees. Our approach exploits semantic information for the nodes of value trees. A semantic relationship between nodes in value trees was captured by the concept of a dimension. Dimension graphs were defined to capture structural information on the dimensions of a value tree. However, dimension graphs are not plain structural summaries of value trees, but rather semantically richer constructs that assist query evaluation. We designed a query language to query value trees. Queries are specified on the dimensions of the value tree and can optionally involve parent-child and ancestor-descendant relationships between these dimensions. A query is not restricted by the structure of a specific value tree. We provided necessary and sufficient conditions for query unsatisfiability and we presented a technique for evaluating satisfiable queries. Finally, we showed how dimension graphs can be used to query multiple value trees in the presence of structural differences and inconsistencies.

We are currently working towards two directions. We are first elaborating on how our framework can be used for integrating tree structured data. In particular, we examine how to apply our techniques to query and integrate XML data sources that conform to different DTDs. The second research direction involves extending our query language with additional features, e.g. branching path expressions and disjunctions.

References

1. Exchangeable Faceted Metadata Language, (XFML), 2003, http://www.xfml.org/.
2. XML Topic Maps (XTM), 2001, http://www.topicmaps.org.
3. World Wide Web Consortium site (W3C), http://www.w3c.org.
4. XML Path Language (XPath). World Wide Web Consortium site. W3C, 2003-2005, http://www.w3c.org/TR/xpath20/.
5. XML Query (XQuery). World Wide Web Consortium site, The Architecture Domain. W3C, 2003-2005, http://www.w3.org/XML/Query.
6. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: from Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, 2000.
7. B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Ontology-based Integration of XML Web Resources. In *Proc. of the ICSW'02 Conference, Sardinia, Italy*, 2002.
8. R. Behrens. A Grammar-based Model for XML Schema Integration. In *Proc. of the BNCOD'00 Conference, Exeter, UK*, 2000.
9. S. Bergamaschi, F. Guerra, and M. Vincini. A Data Integration Framework for E-commerce Product Classification. In *Proc. of the ICSW'02 Conference, Sardinia, Italy*, 2002.
10. P. Buneman, S. B. Davidson, M. F. Fernandez, and D. Suciu. Adding Structure to Unstructured Data. In *Proc. of the ICDT'97 Conference*, Delphi, Greece, 1997.
11. A. B. Chaudhri, A. Rashid, and R. Zicari. *XML Data Management*. Addison Wesley, 2003.
12. V. Christophides, S. Cluet, and J. Simeon. On Wrapping Query Languages and Efficient XML Integration. In *Proc. of the ACM SIGMOD'00 Conference*, USA, 2000.
13. S. Cluet, P. Veltri, and D. Vodislav. Views in a Large Scale XML Repository. In *Proc. of the VLDB'01 Conference, Rome, Italy*, 2001.
14. M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: A System for Extracting Document Type Descriptors from XML Documents. In *Proc. of the ACM SIGMOD'00 Conference, Dallas, Texas, USA*, 2000.
15. R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proc. of the VLDB'97 Conference, Athens, Greece*, 1997.
16. D. Kim, J. Kim, and S.-G. Lee. Catalog Integration for Electronic Commerce through Category-hierarchy Merging Technique. In *Proc. of the RIDE'02 Workshop, San Jose, USA*, 2002.
17. M. L. Lee, L. H. Yang, W. Hsu, and X. Yang. Xclust: Clustering XML Schemas for Effective Integration. In *Proc. of the CIKM'02 Conference, Virginia, USA*, 2002.
18. I. Manolescu, D. Florescu, and D. Kossmann. Answering XML Queries over Heterogeneous Data Sources. In *Proc. of the VLDB'01 Conference, Rome, Italy*, 2001.
19. P. J. Marron, G. Lausen, and M. Weber. Catalog Integration Made Easy. In *Proc. of the ICDE'03 Conference, Bangalore, India (poster)*, 2003.
20. E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4), 2001.
21. Y. Tzitzikas, N. Spyratos, P. Constantopoulos, and A. Analyti. Extended Faceted Taxonomies for Web Catalogs. In *Proc. of the WISE'02 Conference*, Singapore, Dec 2002.