Cooperative Query Answering for Semistructured Data

Michael Barg **Raymond K. Wong**

School of Computer Science & Engineering
University of New South Wales
Sydney, NSW 2052, Australia
Email: {mbarg,wong}@cse.unsw.edu.au

Abstract

Semistructured data, in particular XML, has emerged as one of the primary means for information exchange and content management. The power of XML allows authors to structure a document in a way which precisely captures the semantics of the data. This, however, poses a substantial barrier to casual and non-expert users who wish to query such data, as it is the structure of the data which forms the basis of all XML query languages. Without an accurate understanding of how the data is structured, users are unable to issue meaningful queries. This problem is compounded when one realises that data adhering to different schema are likely to be contained within the same data warehouse or federated database. This paper proposes a method which enables users to meaningfully query semistructured data with no prior knowledge of its structure. We describe a mechanism for returning approximate answers to a database query when the structure of the underlying data is unknown. Our mechanism also returns useful results to the user if a specific value in the query cannot be matched. We discuss a number of novel query processing and optimisation techniques which enable us to perform our cooperative query answering in an efficient and scalable manner.

Keywords: Cooperative query processing, semistructured data, XML.

1 Introduction

The richness of the XML data format allows data to be structured in a way which precisely captures the semantics required by the author(s) [XML]. Such requirements are naturally influenced by the purpose of the author(s), as well as the context in which the document is being written. Such purposes and contexts vary widely, resulting in data with the same *semantic* content having vastly differing structure. This poses substantial barriers to casual users and non-domain experts who wish to query the data, as it is the structure of the data which forms the basis of all XML query languages [ABI97, BUN97]. Without at least some notion of the structure, a user cannot meaningfully query the data. This problem is compounded when one considers that heterogeneous data adhering to different schema are likely to be included in the same data warehouse, federated database or integrated data repository [MF+01, QR+95].

Even if the structure is known to the user, an empty result set is frequently returned to the user. Such an occurrence may be due to the user misconception about the data structure or simply that a specific value was not matched. This rigid answer does not provide any helpful information to the users, except an indication that some portion of their query could not be satisfied.

Suppose, for example, we are looking for trends in "insurance claims" related to "smoking" for "women" over "40". The information we are after may be contained in insurance company records, court transcripts, or even newspaper articles. Even if we decide we are only interested in examining court transcripts, we do not know the structural relationship between the terms of interest. We are left in the predicament of knowing exactly what we are looking for, but not knowing how to find it.

Consider another example, where we wish to find the phone number of *Bob*, the new sales manager. Unfortunately, his entry has not been created in the database yet. Instead of informing the user that no such phone number exists, it would be much more cooperative if the database could relax the query and return the phone number of the secretary and/or the reception of sales department instead.

The notion of cooperative query answering is well established for relational databases. Databases may attempt to return answers which "approximately" answer the users query, if no exact match can be found. Such approaches typically assume the *structure* of a query is correct, and employ a variety of techniques to approximate the *value* of particular criteria [BW99, CY+96, GG+98].

XML data, however, poses its own, more complex set of issues with regards to cooperative query answering. As it is the structure of the data which poses the greatest barrier to user queries, a more useful approach to cooperative query answering for semistructured data is to return an answer where the *structure* of the data approximately matches the *structure* specified by the query. To be truly useful, such a notion of "approximate" matching should incorporate some concept of *semantic* as well as *structural* similarity. Thus, two structures should be considered "similar" if they are both close in the structural sense, and convey semantically similar information.

For cooperative query processing, we consider the XML database (such as a Lore database [MH+97]) as a general graph, comprised of one or more individual data sets. Whilst raw XML can always be represented as a tree, the *interpreted* structure (ie. the structure obtained if links are materialised as edges) can be an arbitrary graph. As links generally indicate a "relatedness" between elements, we must consider this interpreted structure to maximise the effectiveness of our cooperative query answering.

With semi structured data, it is frequently appropriate to return approximate answers even if the data *does* contain an exact match to a query. Suppose, for example, we wish to find all "restaurants" in "Soho" which serve "seafood". Figure 1 shows one possible structure for XML data which represents this information. Suppose we specified a structure in our query which exactly matched the restaurant "something fishy". However, even in this small data set, there are two other restaurants which are *semantically* appropriate to return, even though they do not *structurally* match the query. For this reason, it is fre-

Copyright ©2003, Australian Computer Society, Inc. This paper appeared at Fourteenth Australasian Database Conference (ADC2003), Adelaide, Australia. Conferences in Research and Practice in Information Technology, Vol. 17. Xiaofang Zhou and Klaus-Dieter Schewe, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

quently appropriate to return approximate answers, in an appropriate ranked order, even when there is data which exactly matches the query.

As our approach seeks to determine the similarity of subgraphs, where the overall graph (i.e. the entire database) has many subgraphs, the process is potentially very expensive. Given that the most likely use of cooperative query answering is in real time interactive querying sessions, the processing speed is especially crucial.

In this paper we present a mechanism for implementing cooperative query processing for semistructured data in near linear time. We present a method for efficiently scoring topological similarity between graphical components, which takes the semantics of the structure into consideration. We extend the encoding schemes presented in [BW01] to facilitate the efficient calculation of progressive overall scores.

2 Basic Concepts

Each query, Q can be represented by a query tree, Q^T, which indicates the required topological relationship between target nodes. For example, to find the phone number of a restaurant in Soho, the query `/restaurant[.//Soho]/phone_number` (i.e. `restaurant` has a child node `phone_number` and a descendant `Soho`) yields the query tree shown in figure 2.

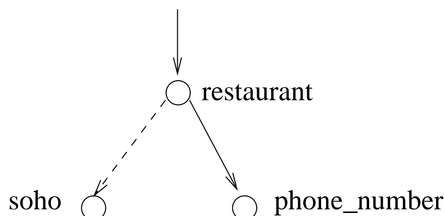

Figure 2: Query Tree for: `/restaurant[.//Soho]/phone_number`

Let $Q^T(a, b)$, represent an edge in the query tree Q^T. Note that each edge specifies the required topological relationship, $Q(a, b)$, between two nodes. For example, the edge $Q^T(\texttt{restaurant}, \texttt{Soho})$ indicates the query requirement $Q(\texttt{restaurant}, \texttt{Soho})$, that `restaurant` nodes must have a descendant `Soho`. Symmetrical definitions apply for $Q(\texttt{Soho}, \texttt{restaurant})$.

A result term, t_i, is a query term which corresponds to a node explicitly required by the query. For instance, in the above example, we are explicitly requesting all `phone_number` nodes whose location in the graph satisfies certain criteria. Thus in this instance, the result term is `phone_number`. Note that it is possible for a given query tree to contain a number of different result terms, or the same result term repeated in different segments of the query tree.

For a given edge, $Q^T(a, b)$, we say that a is the head of the edge iff len(a, t_i) < len(b, t_i), where len(a, b) is the minimum number of edges between a and b, and len(a, t_i) ≤ len(a, t_j), and len(b, t_i) ≤ len(b, t_j) $\forall t_j$. If len(a, t_i) = len(b, t_i), head$(Q^T(a, b)) = a$, where a is closer to the root of Q^T than b. Informally, the "head" of an edge is the end which is closer (ie. in terms of number of edges) to the nearest result term. (If both ends of the edge are equidistant, we choose the "head" as the end which is closer to the root of the query tree). Note that the "head" of an edge is only defined for edges of the query tree, not the actual data. As expected, tail$(Q^T(a, b))$ = NOT(head$(Q^T(a, b))$).

3 Overview

Our overall approach is to return a set of results, ranked by an overall score which indicates how well the subgraph containing the result satisfies the query criteria. If a given result node exists in more than one potential subgraph (which is usually the case), we select the subgraph which gives the lowest score.

The overall score is based on how closely the structure of the subgraph conforms to the structure required by the query tree. We progressively determine the score, using a graph algorithm to determine how well each candidate pair of nodes satisfy the criteria in a single edge of the query tree.

Implementing such an approach with an arbirary disk based graph would require at most $V \times n$ disk seeks, for a graph with V vertices and a query itree with n edges. Obviously such an approach is impractically slow!

Instead of directly examining the graph, we extend the work done in [BW01] to encode the subgraph under consideration. Individual nodes are encoded in a small space (typically less than 20 bytes), and stored in a structural index. These are combined as required into larger encodings which represent subgraphs containing these nodes. Graph based algorithms are then performed utilising these encodings, rather than considering the raw graph itself.

We extend the algorithms presented in [BW01] to achieve constant time score calculation. Firthermore, as the encodings are designed to remain as small as possible, calculations and comparisons occur in main memory. As these comparisons and calculations heavily utilise bitwise comparisons and optimisations, such operations are performed very quickly.

Due to the exponential number of possible subgraphs, we employ a greedy algorithm to aggressively prune the search tree. For each query tree edge, each node which matches the head is "labeled" with the progressive score. This is comprised of a score which indicates the best match for the given node to the criteria specified by the query tree edge, and the previous progressive score for the relevant node which matches the tail. In this way the final score is progressively calculated with minimal additional overhead.

4 Similarity Considerations

The notion of "similarity" incorporates both topological deviation and the semantics such deviation implies. For example, consider the query `/restaurant/Soho`. Figure 3(a) shows the structure which exactly matches the query. If this structure is to be approximated, the new topology should be the closest structure in the *semantic* sense. Informally, this can be defined as "a different structure which means the same thing".

Figure 3(b)-(f) shows alternative structures which do not match the *structural* requirements of the query, but do match the *semantic* requirements. As can be seen, if the structure implied by the query requires that `a` be a child of `b`, it is possible that *any*other topological relationship between `a` and `b` may also satisfy the *semantic* requirements of the query. Determining this computationally, however, has a number of potential difficulties.

Whilst many mechanisms exist for determining if two terms "mean the same thing" (such as thesauri to lexical databases (eg. word net) to ontologies), none of these mechanisms take into considerations structural relationships, (where the structure is defined arbitrarily). Furthermore, some mechanism needs to be developed to *efficiently* determine the two similarities of two structures.

Proximity (in the structural sense) has been successfully used to indicate the "relatedness" of two terms [BW01, GS+98]. We extend this notion to define deviation proximity, which gives a measure of how far one structure "deviates" from a topological criteria.

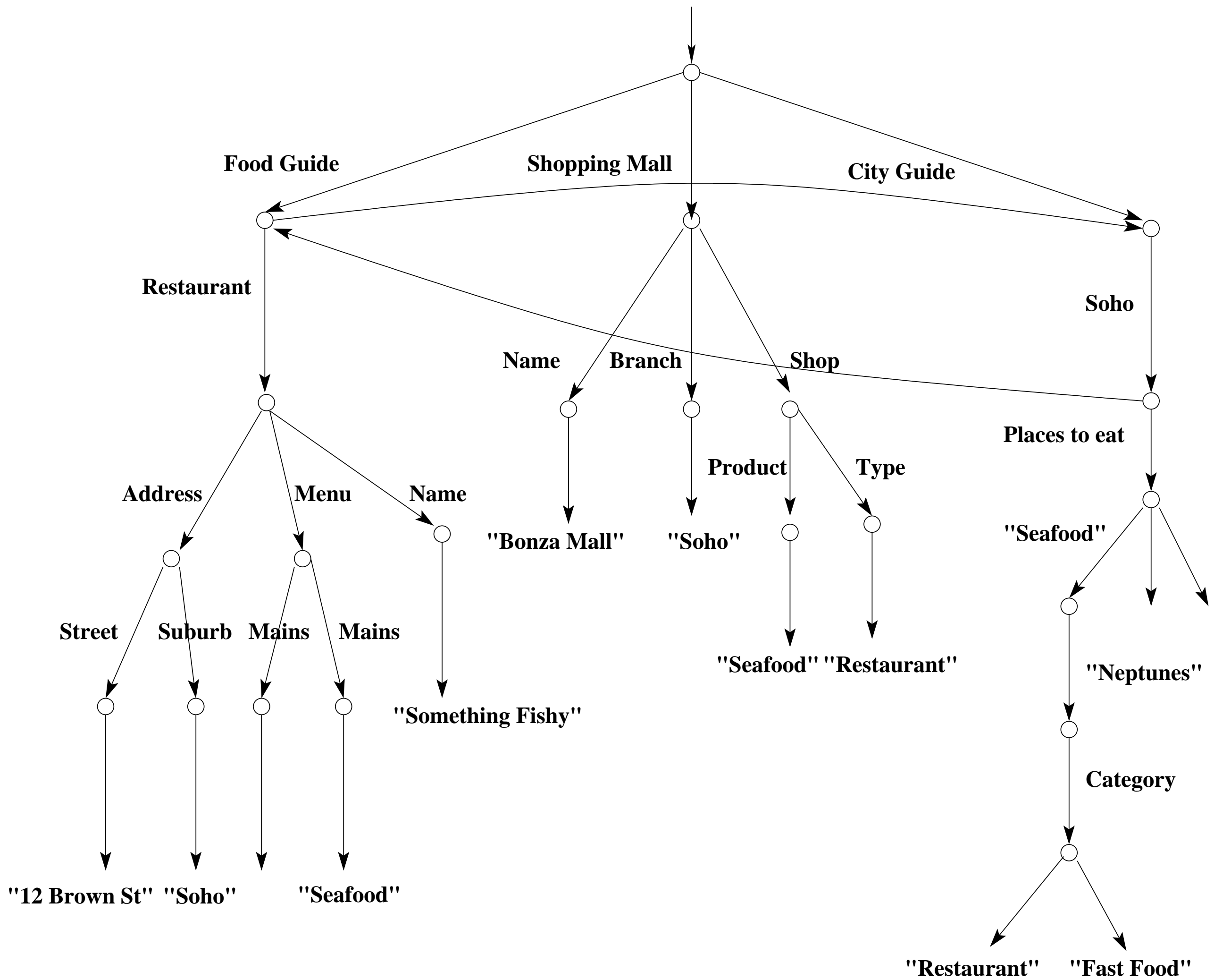

Figure 1: Potential XML Graph of Restaurant Information

Calculating the extent to which two structures deviate in the structural sense is possible to achieve in near constant time by modifying the method presented in [BW01]. Conceptually, this works by ”overlaying” the encoding representing the node of interest on the existing graph, and observing the point at which they deviate. In practice, this ”overlaying” is performed using bitwise comparisons, and tends towards constant time in practice.

We further refine the deviation score, by applying a suite of scoring functions which take into account the likely semantic implications of the observed deviation. Space precludes us from exploring these scoring functions in detail. In general, however, these are based on general observations about the semantic interperetation of general semistructuired doicument construction, and their relationship to other such components. Applying the scoring functions allows us to ”fine tune” the final score, to increase its overall semantic relevance.

5 Cooperative Query Processing

5.1 Converging Order

The order in which we consider edges from the query tree plays an important part in the overall query processing. Informally, we wish to consider paths through the query tree which ”converge” on a result term (it is not important which one). This allows us to progressively calculate the minimum overall score for all query tree edges considered so far.

This order ensures that the head of one edge will become the tail of a future edge wherever possible. By storing the minimum score at this node, therefore, we always have access to the appropriate previous score (which we combine with the minimum score for the current edge) without requiring any additional lookups.

Furthermore, if the last edge we consider in the query tree contains a result term, we can then directly insert this in the final result set, without needing any additional processing to retrieve the result nodes from the graph or any interim data structure. This ensures that the overall algorithm is O(n), where a given query tree has n edges.

We refer to this order of considering edges from the query tree as *converging order*. Figure 4, shows a query tree, with one possible converging order (Note that for any given query tree, converging order is not necessarily unique). Nodes labeled *r* indicate result terms, and edge labels correspond to the order in which edges are considered.

If a leaf in the query tree has no matches in the data set, then this is ignored. Note that any CDATA value (such as `"Bob"` in `name = "Bob"`) is guaranteed to correspond to such a leaf node. As such, we enable the system to return useful answers in cases where specific data values do not exist.

We can now begin to see how converging order assists in co-operative query processing. Consider the query tree and ordering shown in figure 4. The first edge we consider in the query tree is $Q^T(e, h)$, which specifies the query requirement that nodes which match h must be a child of nodes which match e.

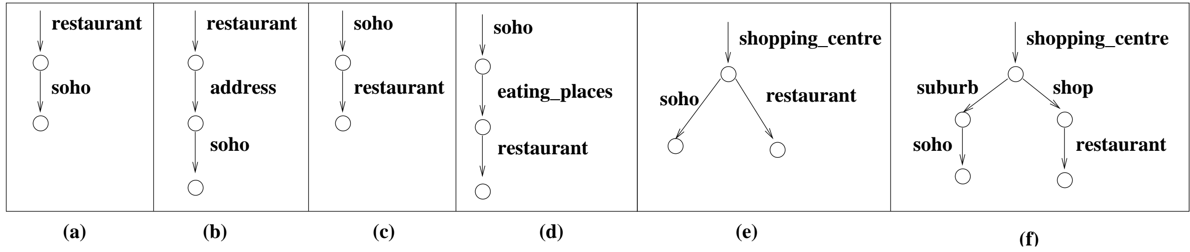

Figure 3: Semantically Similar Topologies

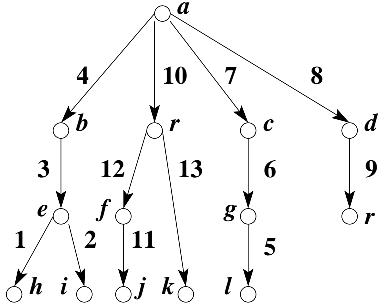

Figure 4: Sample Query Tree

The first pass of the algorithm presented in section 5.2 finds the minimum score of all nodes which match e to the nearest node which matches h. This score is then stored at each e node within the encoding. Note that the actual e and h nodes may have any topological relationship. It is for this reason that traditional tree matching approaches are unsuited to co-operative query processing.

The second pass of the algorithm in section 5.2 (which considers $Q^T(e,i)$) calculates the minimum deviation score of all nodes which match i to the nearest node which matches e. This deviation score calculation requires us to examine the relevant e node in the encoding. At the same time, we obtain the progressive deviation score. This is then combined with the new deviation score for i, and the new progressive score is stored at the relevant e node in the encoding.

In this way, we can always obtain the most recent, relevant, progressive score with no additional lookups.

5.2 Main Query Processing

We can now look at the main algorithm for co-operative query processing. The overall approach has already been suggested in section 5.1. Converging order is used to enable progressive calculation of the overall deviation. Figure 5 shows the algorithm used to process queries, which describes this process in detail.

The remainder of this discussion assumes that H and T are the set of nodes which match the head and tail of a given query tree edge respectively.

The algorithm works by considering edges in converging order (step [1]). For each query tree edge, we first consider the tail (steps [2] through [4]), and then the head (steps [5] through [14]). Considering the tail merely involves ensuring that the tail is represented in the progressive encoding. For each node which matches the head, we calculate the score for that node (step [7]) score Steps [7] and [9] are discussed in more details in sections 5.2.1 and 5.2.2 respectively.

Conceptually, this approach progressively "builds" as much of the graph as is required to represent all nodes which match the various query terms (the progressive encoding from steps [4] and [11]). The "graph" that is actually "built" is the subgraph encoding discussed in section 3. In practice, building the encoding heavily utilises bitwise operations and optimisations, giving very fast performance.

A modified graph based algorithm is then used to examine the encoded graph to determine the score s, for each node. As discussed in section 4, is based on "proximity deviation". Fundamentally, this is the degree to which the path (n, m) satisfies the query edge criteria, where n is the node which matches the head, and m is the node matching the tail which *best* satsfies the query tree edge. This means that for a given query tree edge, the *best* score is chosen for each node matching the head, considering *all* nodes which match the tail. Obviously, this definition potentially potentially leads to serious performance degredation. This issue is discussed in detail in section 5.2.1. Looking ahead a little, however, we see that we can exploit various features of our encoding schemes to perform this step in near constant time.

In practice, each iteration tends toward O($|H| + |T|$) if the tail encodings must be retrieved and included (steps [3] and [4]), or O($|H|$) otherwise. Thus, for a query with n query terms (and thus $n-1$ query tree edges), the worst case performance is O($(|H| + |T|)\times n$). As we consider edges in an order which minimises the number of times step [2] is required, performance frequently tends toward O($|H| \times n$) in practice.

5.2.1 Individual Node Score Calculation

As mentioned in section 5.2 above, determining the score, s, for a single node is potentially very expensive. This is chosen as the minimum score for each head node, based on *all* the tail nodes in the graph. This might suggest the need to compare each head node with *all* tail nodes to. This approach is O($|T| \times |H|$), where T and H are the set of nodes which match the tail and head respectively. Obviously such an approach is untenable.

Furthermore, even if we somehow know the location of the relevant tail node, the score is based on the topological relationship between two nodes. This might suggest the need to fully locate *both* nodes in the graph. This effectively corresponds to additional redundant graph traversals, even if the graph is encoded and traversals are accomplished using bitwise operations.

As mentioned in section 3, we use a modification of the algorithm presented in [BW01]. This method presented a means of finding the distance from all nodes in one set to the *closest* node in a second set, in O($|T|+|H|$). The main approach was to build an encoding which represented the first set of nodes (all nodes which match the tail in our case. Generating the encoding is O($|T|$)). Next, encodings were obtained for each element from the second set of nodes (the head, in our case, in step [5] in figure 5). Each of these was compared with the encoding to obtain the distance to the nearest element (O($|H|$).

We are able to obtain minimum scores using the same approach. Importantly, this method guarantees that after we have generated the encoding of all tail nodes, we can find the *best* deviation score for each head node in near constant time.

Algorithm: Cooperative Query Processing
Input: Query Q, which is represented by the query tree, Q^T.
Output: Set of ranked tuples, <*result*, `score` >

[1] For each edge of the query tree, $Q_i^T(a,b)$, chosen in converging order
[2] If tail($Q^T(a,b)$) has not been represented in the progressive encoding
[3] Obtain the encodings which represent the set of all nodes which match tail($Q^T(a,b)$).
[4] Include these in the progressive encoding, which represents all nodes examined so far.
[5] Obtain the encodings which represent the set of all nodes which match head($Q^T(a,b)$).
[6] For each of these encodings
[7] Calculate the score, s, which indicates the degree to which the node satisfies the criteria specified by $Q^T(a,b)$.
[8] Combine the encoding into the progressive encoding from step [4]
[9] Calculate the minimum total progressive score, S, for this node
[10] If this is not the final edge in the query tree
[11] Include S at this "node" in the progressive encoding from step [4]
[12] Else
[14] Insert <node, S> into result set

Figure 5: Cooperative Query Processing Algorithm

This method additionally avoided redundant graph traversal by storing additional distance information at specific nodes in the encoding. Each encoding of the latter set (H) was conceptually "overlaid" on the encoded subgraph *only until the encoded node and the encoded subgraph diverge*. This allowed the exact distance between two nodes to be calculated, without the need of actually "visiting" each of these nodes. This "overlaying" and distance calculation was implemented using a number of bitwise operations and optimisations, and was shown to be near constant time in practice.

As we have discussed, the algorithm from [BW01] allows us to perform a single iteration of the loop in figure 5 in O($|H|+|T|$), where H and T are the set of nodes which match the head and tail respectively. Thus, for a query with n terms (yielding a query tree with $n-1$ edges), the query processing is O($n \times (|H|+|T|)$).

5.2.2 Progressive Score Combination

As mentioned earlier, considering edges in converging order sometimes requires that progressive scores be combined. This typically occurs at nodes in the query tree with multiple children, or where edges "converge" on result terms from different directions (for example, both top down and bottom up). In such situations, the same node in the query tree is the head of multiple edges, and so is considered separately in different iterations of the algorithm in figure 5.

In such situations, the final progressive score for a given node in the result tree can only be calculated after all edges which contain that node as the head have been considered. For example, in figure 4, the scores for edges $Q^T(e,h)$ and $Q^T(e,i)$ must both be considered before the final progressive score for e can be determined.

Converging order means that the different converging edges will contain independent progressive scores, which must eventually be combined into a single score. This requires us to store these progressive scores independantly for each node where these scores will eventually be combined.

When the last independent progressive score is calculated, all such scores are combined into a single score. The actual operators used for any two progressive scores depend on the operators used in the original query. For example, if a branch in the query tree indicates a boolean AND, progressive scores are added, whereas if a branch indicates a boolean OR, minimum progressive score is used. Progressive scores are combined in the order determined by the normal query parse tree.

6 Results

Figure 6 shows the types of answers obtained for queries posed over heterogeneous XML data.

Queries 1, 2 and 3 were formulated on the basis of "reasonable" or "logical" assumptions regarding the "likely" structure of the data, not on the basis of prior knowledge of the structure of the .

Query 4 was formulated with prior knowledge of the underlying structure of the data. This is representative of the situation where the structure of the data is known, but the required value does not exist. Note that even though the desired information did not exist in the database, the results returned were useful, as they offered a likely means of obtaining the required information, utilising a mechanism external to the database itself (phoning the appropriate department, for example). Similarly, if the user decided it was not overly important to have the phone number of Bob Smith, a list of alternative contacts was provided.

As can be seen, for all queries, low ranked results tend to contain information which is semantically appropriate to the information desired by the user.

Systems which only consider proximity between 2 terms [BW01, GS+98] are also capable of providing results for a query such as query 1. This is the simplest case of cooperative answering, as only 2 terms are involved. Even in this case, however, such mechanisms make no reference at all to any desired structure specified by the user. Both approaches essentially accept 2 keywords, and rank nodes based on the proximity between them, irrespective of the topology. Thus, even in this case the results are likely to less precise (in the semantic sense).

Such approaches are unable to provide any results for more complex queries. Such approaches return a ranked set of results for the 2 specified keywords, but offer no mechanism for combining such sets. Furthermore, even if such an approach was suggested, the topology of both the query and result are completely ignored by these approaches, substantially reducing the quality of any such results in the semantic sense.

7 Related Work

To the authors' knowledge, no other work has been done attempting to implement cooperative queries for semi-structured data.

To address some of the issues in unknown data structure, however, effort from database researchers has been recently paid to provide more flexible and efficient cooperative search on semistructured data. For example, a major problem faced by the existing Web search engines is that they return only physical pages containing query terms. Frequently, however, a web document for a topic

Cooperative Query Results	
Desired Info	Find all restaurants in Soho
Actual Query	//restaurant/Soho
Results	Restaurants with a child named ”Soho”, followed by restaurants with an address in Soho, followed by restaurants contained in a ”Soho” tourist guide document.
Desired Info	Find all phone numbers of restaurants in Soho
Actual Query	//restaurant[./Soho]//phone_number
Results	Phone numbers with an ancestor ”Restaurant” who had a child named ”Soho”, followed by the phone numbers of restaurants with an address in Soho, followed by a 1800 number to an ”eating out” guide which included restaurants in Soho.
Desired Info	Find all hotels in Sydney with rooms less that $100/night
Actual Query	//hotel[.//Sydney][.//price < 100]
Results	Hotels with a descendant of ”Sydney” and a descendant of ”Price” whose value was less than $100, followed by hotels in a ”Sydney” tourism guide with price less than $100 followed by hotels in Sydney in the ”budget” category (where budget was defined as <$100 in that document).
Desired Info	Find the telephone number of the sales manager, ”Bob Smith” of company ”Fishy Enterprises” (NB: Suppose there is no such person at this company)
Actual Query	//sales_manager[./company/name=”Fishy Enterprises” and ./name=”Bob Smith”]/phone_number
Results	Telephone numbers for the sales managers at ”Fishy Enterprises”, followed by the phone number of the sales department, followed by the phone number of the secretaries to the sales managers.

Figure 6: Cooperative Query Results

may span multiple pages and be authored in many different ways. Li and Wu [LW99] have introduced the concept of information unit, which can be viewed as a logical Web document consisting of multiple physical pages as one atomic retrieval unit. A framework of query relaxation by structure is proposed, whereby a set of connected physical pages, contain, as a whole, all query terms which can be retrieved. The work by Tajima et al. [TH+99] extends the concept of information units by considering keyword occurrence frequency and distribution. Such an approach, however, does not allow consideration of the structure *within* a single document.

Another approach has been to rank nodes based on their proximity to one another in the structural sense [BW01, GS+98]. Such a method can produce good results when the user wishes to find one element which is *near* another. Whilst such approaches yields good results for searches involving 2 terms, no mechanism is presented for performing searches involving more terms. Furthermore, such an approach does not consider the topology of either the original query or the returned results. For both these reasons, such an approach is inappropriate for approximating queries on semi-structured data.

8 Conclusions

In this paper we have presented a method for implementing a fast, efficient cooperative query processing mechanism for semi structured data. Our mechanism returns results where the structure of the data approximates the structure specified in the query. Our notion of approximation incorporates both structural and semantic similarity. Our mechanism also returns helpful results in instances where specific data values specified in the query cannot be found. We have described a framework for calculating similarity scores, which incorporate both structural and semantic similarity, in an efficient manner. We extend the mechanism for encoding graphs presented in [BW01], to allow us to efficiently calculate progressive similarity scores, by considering only the encoding held in memory. Our cooperative query processing algorithm exploits these extensions to provide near linear time performance. This is especially impressive when one considers that our similarity is based on structural similarity between all potential target subgraphs. We have further shown that our mechanism produces useful answers for queries where the underlying data structure is unknown.

References

[ABI97] S. Abiteboul. Querying semi-structured data. In *Proceedings of the International Conference on Database Theory (ICDT)*. Springer Verlag, 1997.

[BW99] M. Barg and R. Wong. Building User Profiles for Cooperative Query Answering. *AAAI Fall symposium on Question Answering Systems*, Massachusetts, November, 1999

[BW01] M. Barg and R. Wong Structural Proximity Searching for Large Collections of Semi-Structured Data In *ACM Conference on Information and Knowledge Management*, November, 2001.

[XML] T. Bray, J. Paoli, and C.M. Sperberg-McQueen. Extensible markup language (XML) 1.0. In *W3C Recommendation, World Wide Web Consortium*, 1998; available online at http://www.w3.org/TR/1998/REC-xml-19980210.

[BUN97] P. Buneman. Tutorial: Semistructured data. In *International Conference on PODS*, 1997.

[CY+96] W.W. Chu, H. Yang, K. Chang, M. Minock, G Chow, C Larson. CoBase: A Scalable and Extensible Cooperative Information System. *Journal of intelligent information systems*, 6(2/3): 223-259, May 1996

[GG+98] T. Gaasterland, P. Godfrey and J. Minker. An Overview of Cooperative Answering, *Journal of Intelligent Information Systems*, 1, 123-157, 1992

[GS+98] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina. Proximity Search in Databases. In *International Conference on VLDB*, 26–37, 1998.

[LW99] W.-S. Li and Y.-L. Wu. ” Query Relaxation By Structure for Document Retrieval on the Web. In *Proceedings of 1998 Advanced Database Symposium*, Shinjuku, Japan, December, 1999.

[MH+97] J. McHugh *et al.* Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, September 1997.

[MF+01] I. Manolescu, D. Florescu, and D. Kossmann. Answering XML queries on heterogeneous data sources. *International Conference on VLDB*, to appear, 2001.

[QR+95] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying semistructured heterogeneous information. In *Proceedings of Deductive and Object Oriented Databases*, 1995.

[TH+99] K. Tajima, K. Hatano, T. Matsukura, R. Sano and K. Tanaka. Discovery and Retrieval of Logical Information Units in Web. In *Proceedings of the 1999 ACM Digital Libraries Workshop on Organizing Web Space*, Berkeley, CA, USA, August, 1999.

[SODA] The SODA Research Group. *SODA2: The Semistructured Object Database System, Version 2*, 2000. `http://dba.cse.unsw.edu.au`