SEDA: A System for Search, Exploration, Discovery, and Analysis of XML Data

Andrey Balmin
IBM Almaden Research Center

Latha Colby
IBM Almaden Research Center

Emiran Curtmola[†]
UC San Diego

Quanzhong Li
IBM Almaden Research Center

Fatma Özcan
IBM Almaden Research Center

Sharath Srinivas[†]
University of Maryland, College Park

Zografoula Vagena[†]
Micorsoft Research

ABSTRACT

Keyword search in XML repositories is a powerful tool for interactive data exploration. Much work has recently been done on making XML search aware of relationship information embedded in XML document structure, but without a clear winner in all data and query scenarios. Furthermore, due to its imprecise nature, search results cannot easily be analyzed and summarized to gain more insights into the data. We address these shortcomings with *SEDA*: a system for Search, Exploration, Discovery, and Analysis of XML Data. *SEDA* is based on a paradigm of search and user interaction to help users start with simple keyword-style querying and perform rich analysis of XML data by leveraging both the content and structure of the data. *SEDA* is an interactive system that allows the user to refine her query iteratively to explore the XML data and discover interesting relationships.

SEDA first employs a top-k algorithm to compute the most relevant top-k answers fast, and returns tuples of nodes ranked by relevance. *SEDA* provides several novel data structures and techniques for efficient top-k computation over graph-structured XML data. *SEDA* also computes all the contexts in which the query terms are found and all the connection paths that connect the query terms in the XML data. These two summaries enable the user to refine her query by disambiguating the contexts and connections relevant to her query. With the user feedback, the system has enough information to compute *all* query results, not just the top-k. From the complete results, *SEDA* automatically deduces a star schema, which is then instantiated with the query results and augmented with additional values required for a well-defined data cube. The tables computed at this step are input into an OLAP engine for further analysis.

[†]This work was done while the author was at IBM Almaden Research Center

PVLDB '08, August 23-28, 2008, Auckland, New Zealand
Copyright 2008 VLDB Endowment, ACM 978-1-60558-306-8/08/08

1. INTRODUCTION

Querying highly heterogeneous XML collections is very challenging. The heterogeneity in the structure of the XML instances in such data repositories is often the result of data integration and schema evolution. There are also scenarios, where there exist schemas, but they are highly generic, producing structurally different XML documents. For example, in the healthcare scenario, HL7 provides XML schemas for the Clinical Document Architecture (CDA)[1] documents, but they have highly varying structure. Querying such collections using schema-dependent languages like XQuery[5] and XPath is cumbersome and not always viable because XPath expressions are designed to allow structural navigation of XML data and hence require a high-degree of schema knowledge. Searches based on keyword matches alone provide a simple way of retrieving information from such repositories but are insufficient for generating meaningful query results that also capture the connections between data elements and for generating summarizations necessary for gaining more insights into the data.

In recent years, the problem of deriving meaningful results from "underspecified" queries on XML data, i.e., queries expressed with little or no knowledge of the underlying schema of the data, has received a lot of attention. There have been several proposals that use various heuristics to automatically determine the objects and connections that are most meaningful given a set of XML fragments which have been identified (typically by applying keyword search or value-based selections in a user's query). However, as shown in [4], each of these techniques is well-suited for certain types of data scenarios, and hence work in some scenarios but fail in others. In order to reliably determine the set of relationships that are of interest to the user, the system would have to somehow infer the true user intent behind the keyword query, which is an "AI complete" problem.

In this paper, we present *SEDA*[2] a system based on a paradigm of search and user interaction to help users start with simple keyword style querying and perform rich analysis of XML data. Our approach also starts with a "connectedness" heuristic, but augments it with user feedback, thus giving the user full control over the quality of the query results. We facilitate the feedback process by presenting summaries of the intermediate results in a form that enables the user to make good choices. In the *SEDA* system, we chose a

[1] http://www.hl7.org/Special/committees/xml/drafts/drafts.htm

[2] SEDA stands for Search, Explore, Discover and Analyze

ranking heuristic based on the content of nodes satisfying each of the query terms and the compactness of the graph connecting those nodes. We augment the results with a system of user disambiguation of context and connection information. The system employs a top-k processing algorithm that uses several novel data structures and techniques for efficient processing. Unlike traditional relational and XQuery joins where join conditions are specified in the query, the computation of these ranked tuples involves determining if the set of nodes in a result tuple are connected in the data graph, and ranking them based on their content and the compactness of the computed subgraph.

We provide tools that allow the user to explore a given XML collection and discover relevant objects and relationships in the data. After this exploration step, the user is also provided with the option of using the results of her query to compute a data-cube. Next, we provide an overview of the system and illustrate how it operates through an example.

2. SYSTEM OVERVIEW

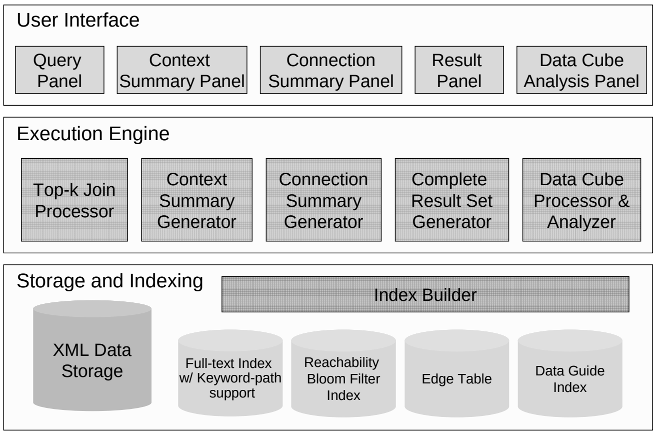


Figure 1: Architecture of *SEDA*

The architecture of *SEDA* is provided in Figure 1. It contains three major components: a user interface, an execution engine, and a storage and indexing component.

The user interacts with the system through various panels in a GUI. The goal of *SEDA* is to help users in the formulation of complex queries through search and discovery. For this purpose, *SEDA* provides context and connection summary panels, as well as an OLAP panel, in addition to the query and result panels. The execution engine contains several units for processing a user's query, guiding her to refine her search for increasing precision and finally for computing an OLAP-style data cube. XML data is stored in DB2® pureXML™[3]. In addition, the storage component contains several indexes to efficiently support these operations, including a Lucene full-text index.

SEDA operates on collections of XML documents and models XML data as a graph in which nodes represent element or attribute nodes and edges represent various relationships between nodes. Each data node has context and content. The *context* of a node is its root-to-leaf path, starting from a root node and following only parent/child edges. The *content* of a node is its textual content. A query in *SEDA* consists of a set of *query terms*. A query term is a pair of the form $(context, search\_query)$, where the context is either a root-to-leaf path or just a node name (including wilcards), and search_query is any full-text search expression comprising a simple bag of keywords, a phrase query or a boolean combination of those. In *SEDA* the $search\_query$ is evaluated over the content of an individual node, as opposed to an entire XML document.

The result of a *SEDA* query is a set of tuples, where each tuple represents a connected sub-graph with m nodes, one for each query term. Query results in *SEDA* are ranked by both their *content score*, as well as their *structural score*. The content score is obtained by combining the TF/IDF-based content scores of the matching nodes. For structural score, *SEDA* employs a common and practical approximation, namely the Euclidean Minimum Spanning Trees (EMST) [1] of the nodes in result tuples. For a *SEDA* query result tuple, $t =< n_1, n_2, ..., n_m >$ and its connected data graph $g(V', E')$, we define the score $S(t)$ as:

$$S(t) = \sum_{i=1}^{m} S_c(n_i) + \sum_{i=1}^{m-1} \frac{1}{d_i} \tag{1}$$

where $S_c(n_i)$ is the content score of node n_i, and $d_1, ..., d_{m-1}$ are $m-1$ edge weights in the MST of $g(V', E')$.

When a user submits a query to *SEDA*, it first employs the top-k join processor to compute the most relevant top-k answers fast. The *SEDA* top-k algorithm retrieves the results from full-text searches and calculates top answers according to its ranking function. The algorithm is based on the family of threshold algorithms (TA)[2]. It employs the multi-way symmetric join, the Bloom filter index, and dynamic join reordering for efficient query processing.

The query processing in *SEDA* requires finding how two nodes are connected. Computing this efficiently is challenging. In order to calculate the score of a query result tuple using EMST, we need to find the shortest distance between every pair of nodes. However, storing the complete transitive reachability table, which contains the shortest paths between every pair of nodes in the data graph, is very costly. Also, searching the shortest path at runtime can be very expensive. To handle these challenges, *SEDA* utilizes *a Bloom filter index* to support efficient connectivity tests during top-k processing over graph data. The bloom filter index encodes the shortest paths between node pairs, excluding paths that traverse only parent/child edges, as these paths can be computed using their Dewey ids[3]. In our actual implementation, we store the Bloom filters in a B+-tree index, such that we can locate each filter efficiently. Note that Bloom filters may generate false positives and this will boost the structure score of some answers, but will not affect the correctness of the returned answers. Because we test connectivity based on component ids as the first step.

In addition to the set of results, *SEDA* also computes all the contexts in which the query terms are found, and all the paths that connect the nodes that satisfy the query terms in the XML data. If the top-k answers contain what the user is looking for, the user may stop her exploration at this point. Otherwise, she may refine her query and resubmit, or use the context and connection summaries to further refine her search. To efficiently find all distinct paths (corresponding to all distinct contexts) in which the query terms appear within the entire collection, we maintain a full-text index which maps individual keywords to the set of distinct paths in which they appear. If the user refines her query by choosing a subset of contexts, *SEDA* employs the top-k algorithm again, but restricts the results to the subset the user is interested in.

There are various heuristics proposed in the literature to decide which connections are more meaningful. But as shown in [4], these heuristics work in some scenarios but fail in others. It is very hard to find an approach that will work in all possible data and query

[3] DB2 pureXML is a trademark of the IBM Corporation

scenarios, because each different connection represents a different semantic relationship and it is impossible to know what the user intention really is. Furthermore, it is very expensive or even infeasible to show all possible connections between matching nodes of a query to the user. Based on these observations, the solution employed by *SEDA* is to choose a subset of "meaningful" connections to present to the user, and let her specify the ones that are relevant for her query. *SEDA* computes a set of connections from the top-k results by using dataguide summaries of the data graph. The user has an option of increasing "k" if an interesting connection is missing. Furthermore, we expect the user to restrict the contexts first so that the number of possible connections is reduced. When the user selects a subset of connections she is interested in, *SEDA* has sufficient information about the users' intentions to compute the *entire* result set, not just the top-k.

At this point, the user has the option of using the results of this exploration to specify an OLAP-style aggregation query. In *SEDA* a dimension (or a measure) is defined in terms of a set of root-to-leaf paths that contain the values of the dimension (or the measure). *SEDA* matches these paths to the contexts of a query term to decide whether the query term corresponds to a known dimension (or measure). *SEDA* also augments the results with extra values, such as keys, that is required to create well-defined fact and dimension tables, and computes an instantiation of the corresponding star schema. To populate the fact and dimension tables, *SEDA* generates and runs several SQL/XML queries. These tables are then input into an OLAP engine, DB2 AlphaBlox$^{\text{TM}}$ [4] to compute data cubes, as well as the desired aggregation functions. It is important to note that each user query may identify different sets of facts and dimensions, resulting in a different star schema. This is in contrast with traditional data warehouses, where there is one fixed star schema. This dynamic behavior of the system allows the users to draw useful insights from aggregation of the data that matches their initial set of query terms.

3. DEMO SCENARIO

We will demonstrate *SEDA* on a dataset constructed by combining the World Factbook 2007 [5] and the Mondial XML [6] data. An example fragment from this data set is shown in Figure 3. The World Factbook is a publicly available database created and maintained by the Central Intelligence Agency (CIA) [7]. It contains comprehensive statistics for every country and territory in the world for each year. The "schema" of this data evolves from year to year. Our dataset contains the six most recent versions of the dataset from 2002 to 2007. The Mondial dataset is a rich compilation of geographical Web data sources on global statistics of world countries, cities, provinces, seas, and international organizations.

In addition to the "containment" relationships represented by the tree-based parent/child connections between nodes, the instances also contain non-tree relationships depicting (1) geographical objects (e.g., countries and oceans) bordering other geographical objects and (2) participation of countries in economic trade relationships.

Consider a scenario where the user is interested in finding facts about various countries and continents using this data set. In this section, we will illustrate how *SEDA* enables a simple search driven exploration of the data by using the following example query.

EXAMPLE 1. **Query Q1:** *Find information about the trade partners of "United States".*

In *SEDA*, a user might approach the initial formulation of this query by specifying two query terms $(*, \text{``}United\ States\text{''})$, and $(trade*, *)$, with the intention of exploring the dataset to see what information about "United States" and "trade" is available.

The context summary of the system interface (see Figure 2) shows that keyword "United States" occurs in 69 distinct paths in the data. The user will browse this list of paths and quickly discover that only the first one is needed to focus the search on the United States articles.

The context summary for the second term contains only two paths: "$/factbook/economy/imports\_partners/item/trade\_percent$", and "$/factbook/economy/exports\_partners/item/trade\_percent$". At this point the user may decide to focus the search only on the United States imports by un-selecting the second path.

While exploring the XML results, the user may notice that the $trade\_percent$ nodes are always associated with $partner\_country$, and decide to include these nodes in the result tuples. One way to do this is to add a third term to the query that reads: $(partner*, *)$. Once again, this term will have two paths in its context summary - one for import partners and one for export partners. Let's assume that the user selects only the import one.

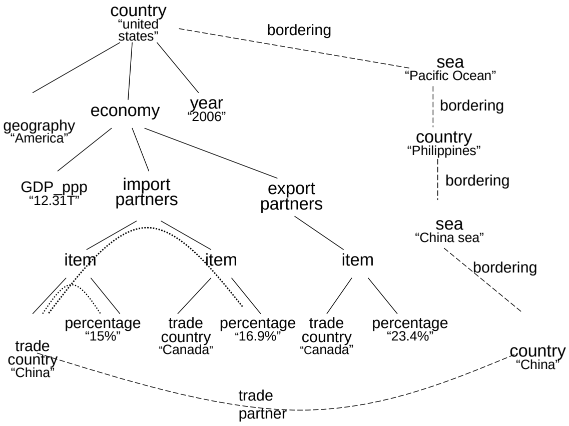


Figure 3: Example data graph from World Factbook and Mondial datasets

After this round of context disambiguation, the user is left with two different ways to connect *partner country* and *trade percent* data nodes, shown as dotted lines in Figure 3. The shorter connection is intuitively much more likely to be meaningful to the user, and its results are going to be ranked first. However, we still give the user a choice to include the results of the second one.

At every stage of the search and disambiguation, the current resulting tuples of nodes are shown to the user on the right side of the screen. Every node is shown with its document title, unique Dewey ID, relevance score, and a snippet of text. In our example, this text is sufficient to show that Germany provided 5.2% of US imports in 2002 and provided 5.3% of US imports in 2004. (first and second result tuples of Figure 2, respectively).

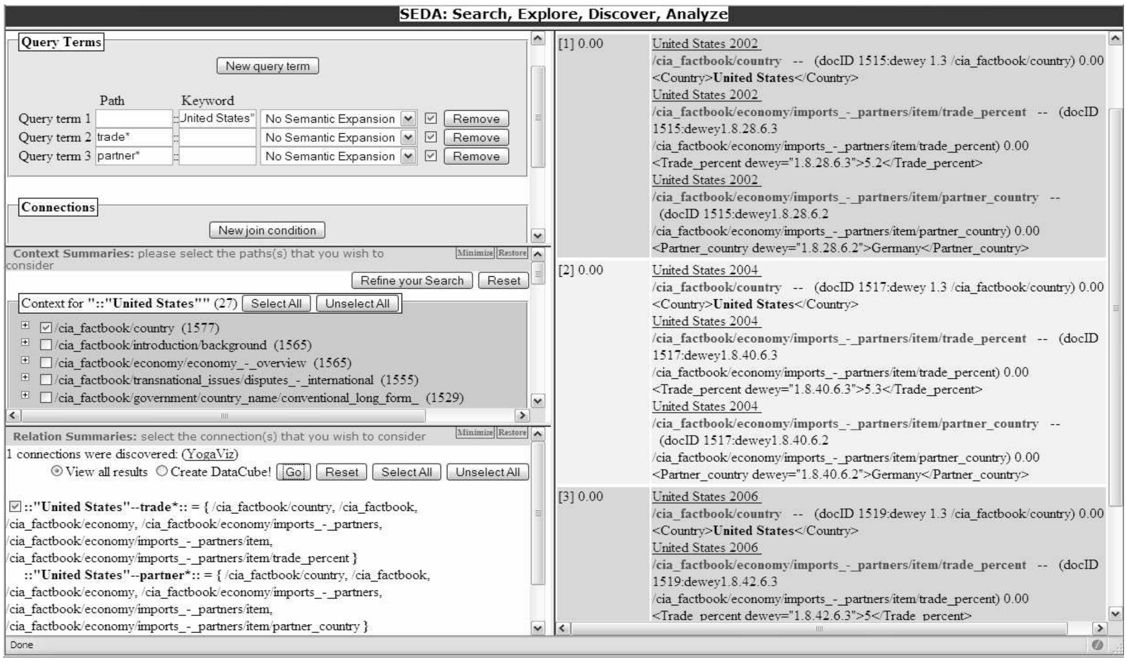
Usually, search systems stop at this point mandating the user to browse the result documents of interest and manually collecting all the data they need for analysis. *SEDA* goes one step further by allowing users to build data cubes out of search results. In this example, the system will identify that the three terms of the query corre-

[4] DB2 AlphaBlox is a trademark of the IBM Corporation

[5] https://www.cia.gov/library/publications/the-world-factbook/

[6] http://www.dbis.informatik.uni-goettingen.de/Mondial/

[7] https://www.cia.gov/



Query Terms

	Path		Keyword			
Query term 1		::	Jnited States"	No Semantic Expansion	☑	Remove
Query term 2	trade*	::		No Semantic Expansion	☑	Remove
Query term 3	partner*	::		No Semantic Expansion	☑	Remove

Connections

Context Summaries: please select the paths(s) that you wish to consider

Context for "::"**United States**"" (27)

- ☑ /cia_factbook/country (1577)
- □ /cia_factbook/introduction/background (1565)
- □ /cia_factbook/economy/economy_-_overview (1565)
- □ /cia_factbook/transnational_issues/disputes_-_international (1555)
- □ /cia_factbook/government/country_name/conventional_long_form_ (1529)

Relation Summaries: select the connection(s) that you wish to consider

1 connections were discovered: (YogaViz)

● View all results ○ Create DataCube!

☑ ::"**United States**"--**trade***:: = { /cia_factbook/country, /cia_factbook, /cia_factbook/economy, /cia_factbook/economy/imports_-_partners, /cia_factbook/economy/imports_-_partners/item, /cia_factbook/economy/imports_-_partners/item/trade_percent }

::"**United States**"--**partner***:: = { /cia_factbook/country, /cia_factbook, /cia_factbook/economy, /cia_factbook/economy/imports_-_partners, /cia_factbook/economy/imports_-_partners/item, /cia_factbook/economy/imports_-_partners/item/partner_country }

[1] 0.00

United States 2002
/cia_factbook/country -- (docID 1515:dewey 1.3 /cia_factbook/country) 0.00
<Country>**United States**</Country>
United States 2002
/cia_factbook/economy/imports_-_partners/item/trade_percent -- (docID 1515:dewey1.8.28.6.3 /cia_factbook/economy/imports_-_partners/item/trade_percent) 0.00
<Trade_percent dewey="1.8.28.6.3">5.2</Trade_percent>
United States 2002
/cia_factbook/economy/imports_-_partners/item/partner_country -- (docID 1515:dewey1.8.28.6.2 /cia_factbook/economy/imports_-_partners/item/partner_country) 0.00
<Partner_country dewey="1.8.28.6.2">Germany</Partner_country>

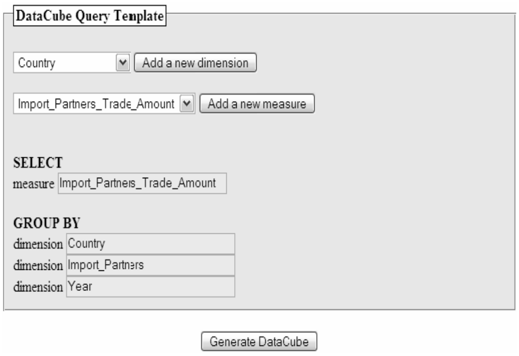
[2] 0.00

United States 2004
/cia_factbook/country -- (docID 1517:dewey 1.3 /cia_factbook/country) 0.00
<Country>**United States**</Country>
United States 2004
/cia_factbook/economy/imports_-_partners/item/trade_percent -- (docID 1517:dewey1.8.40.6.3 /cia_factbook/economy/imports_-_partners/item/trade_percent) 0.00
<Trade_percent dewey="1.8.40.6.3">5.3</Trade_percent>
United States 2004
/cia_factbook/economy/imports_-_partners/item/partner_country -- (docID 1517:dewey1.8.40.6.2 /cia_factbook/economy/imports_-_partners/item/partner_country) 0.00
<Partner_country dewey="1.8.40.6.2">Germany</Partner_country>

[3] 0.00

United States 2006
/cia_factbook/country -- (docID 1519:dewey 1.3 /cia_factbook/country) 0.00
<Country>**United States**</Country>
United States 2006
/cia_factbook/economy/imports_-_partners/item/trade_percent -- (docID 1519:dewey1.8.42.6.3 /cia_factbook/economy/imports_-_partners/item/trade_percent) 0.00
<Trade_percent dewey="1.8.42.6.3">5</Trade_percent>

Figure 2: Example screen shot of *SEDA*, depicting the results for the example query



DataCube Query Template

Country

Import_Partners_Trade_Amount

SELECT

measure Import_Partners_Trade_Amount

GROUP BY

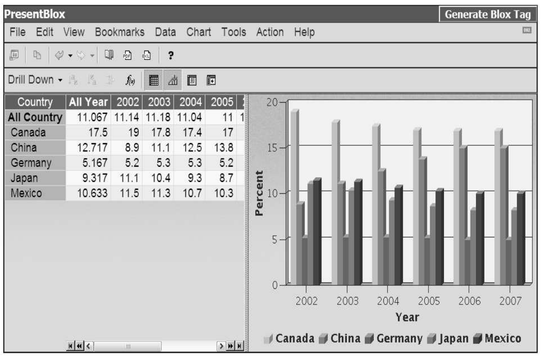
dimension Country
dimension Import_Partners
dimension Year

Figure 4: Data Cube specification user interface.

spond to two dimensions "Country" and "Import_Partners" and one fact "Import_Partners_Trade_Percent". Figure 4 shows the user interface to define a data cube. The system allows the user to choose a subset of these identified facts and dimensions as well as add new ones. Furthermore, the system will determine that one more dimension, namely "Year", is needed to form a unique key for this query result. OLAP engines require that fact and dimension tables have unique keys in order to construct data cubes.

Once the facts and dimensions have been identified, *SEDA* automatically populates the corresponding tables from the results of the query and the database. These tables are then input into the DB2 AlphaBlox OLAP engine to compute the data cube and the desired aggregations of the data. The data cube generated in DB2 AlphaBlox from our running example is shown in Figure 5.

We will demonstrate this query and other similar queries on the World Factbook dataset.



Country	All Year	2002	2003	2004	2005
All Country	11.067	11.14	11.18	11.04	11
Canada	17.5	19	17.8	17.4	17
China	12.717	8.9	11.1	12.5	13.8
Germany	5.167	5.2	5.3	5.3	5.2
Japan	9.317	11.1	10.4	9.3	8.7
Mexico	10.633	11.5	11.3	10.7	10.3

Figure 5: Resulting data cube in DB2 AlphaBlox.

4. REFERENCES

[1] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6(5):407–422, 1991.

[2] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. In *Proc. of PODS*, 2001.

[3] I. Tatarinov et al. Storing and Querying Ordered XML Using a Relational Database System. In *Proc. of SIGMOD*, 2002.

[4] Z. Vagena, L. Colby, F. Özcan, A. Balmin, and Q. Li. On the Effectiveness of Flexible Querying Heuristics for XML Data. In *XSym*, pages 77–91, 2007.

[5] *XQuery 1.0: An XML Query Language*, January 2007. W3C Recommendation, See `http://www.w3.org/TR/xquery`.