XSACT: A Comparison Tool for Structured Search Results *

Ziyang Liu, Sivaramakrishnan Natarajan, Peng Sun, Stephen Booher,
Tim Meehan, Robert Winkler, Yi Chen
Arizona State University
{ziyang.liu, snatara5, peng.sun, stephen.booher, tmeehar, rswinkle, yi}@asu.edu

ABSTRACT

Studies show that about 50% of web search is for information exploration purpose, where a user would like to investigate, compare, evaluate, and synthesize multiple relevant results. Due to the absence of general tools that can effectively analyze and differentiate multiple results, a user has to manually read and comprehend potentially large results in an exploratory search. Such a process is time consuming, labor intensive and error prone. With meta information embedded, keyword search on structured data provides the potential for automating or semi-automating the comparison of multiple results.

In this demo we present a system XSACT for differentiating search results on structured data. XSACT takes as input a set of structured results, and outputs a Differentiation Feature Set (DFS) for each result to highlight their differences within a size bound. The problem of generating DFSs with maximal differences is proved to be NP-hard. XSACT adopts efficient algorithms for DFS generation, and features a user-friendly interface that effectively interacts with the users to help them compare search results.

1. INTRODUCTION

Studies show that about 50% of keyword searches on the web are for *information exploration* purposes, and inherently have multiple relevant results [1]. In contrast to *navigational queries* whose intent is to reach a particular website, the user who issues an *informational query* often would like to investigate, evaluate, compare, and synthesize multiple relevant results for information discovery and decision making purposes. Without the help of tools that can automatically or semi-automatically analyze multiple results, the process of manually reading, comprehending, and analyzing the results can be time consuming, labor-intensive, error prone or even infeasible due to possibly large result sizes.

For example, consider a customer who is looking for *GPS* of brand *TomTom*, who issues a keyword query {*TomTom, GPS*} on a shopping website. There are many results returned; fragments of two results represented in a tree structure are shown in Figure 1, and some statistics information of the results is shown next to the results. As each GPS product may have tens of features, it is very difficult for users to manually check each result, compare and analyze these results to decide which GPS to purchase.

To help users analyze search results, the websites of many banks and online shopping companies, such as Citibank, Buzzillions, Best Buy, etc., provide comparison tools for customers to compare specific products based on a set of pre-defined metrics, and have achieved big success. However, in these websites, only pre-defined types of objects (rather than arbitrary search results) can be compared, and the comparison metrics are pre-defined and static. Such hard coded approaches are inflexible, restrictive and not scalable.

An immediate question is whether a tool that automatically generates a comparison table for query results can be developed, which highlights the differences among results in a small space. A widely used approach to help users judge result relevance is to generate snippets, as supported by every web search engine and some structured data search engines. However, without considering the relationships among results, they generally are not helpful to *compare and differentiate multiple results*. For example, Figure 1 shows two result snippets of query "*TomTom, GPS*" generated by eXtract [2], which highlight the most frequently occurred information in the results. However, snippets are generally not comparable. For instance, from their snippets, we know GPS 1 is best used for *Auto*, but have no idea about GPS 3 in this aspect, since such information is missing in its snippet due to space limitation.

Despite the high demand of a general tool for informative query result comparisons in diverse domains, it is not supported in existing text search engines. The main reason is that text documents are unstructured, making it extremely difficult if not impossible to develop a tool that automatically compares the semantics of two documents.

On the other hand, when searching structured data, the meta information of results presents a potential to enable result comparison. However, many challenges remain, even for enabling structured result comparison. For example, which features in the search results should be selected for result comparison? One desideratum is, of course, such features should maximally highlight the differences among the results. Then, how should we define the difference, and the *degree of differentiation* of a set of features? Another desideratum is, the selected features should reasonably reflect the corresponding results, so that the differences shown in the selected features reflect the differences in the corresponding results. Furthermore, how should we select desirable features from the results efficiently?

To address these challenges, in this demo we present a system XSACT for *comparing* and *differentiating* structured search results. XSACT takes as input a set of structured results, each with a set of

*This material is based on work partially supported by NSF CAREER award IIS-0845647, IIS-0740129, IIS-0915438.

Proceedings of the VLDB Endowment, Vol. 3, No. 2

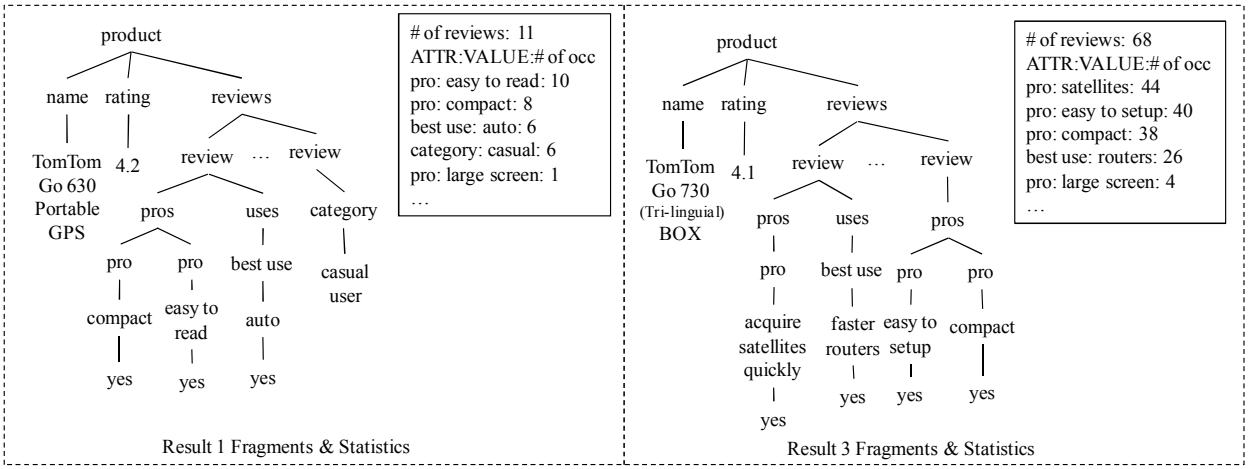
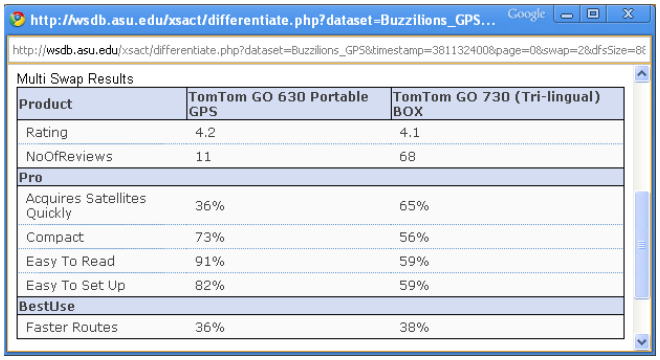


Figure 1: Two Result Fragments of Query {*TomTom, GPS*} and their Statistic Information



Product	TomTom GO 630 Portable GPS	TomTom GO 730 (Tri-lingual) BOX
Rating	4.2	4.1
NoOfReviews	11	68
Pro		
Acquires Satellites Quickly	36%	65%
Compact	73%	56%
Easy To Read	91%	59%
Easy To Set Up	82%	59%
BestUse		
Faster Routes	36%	38%

Figure 2: Comparison Table for the Results in Figure 1

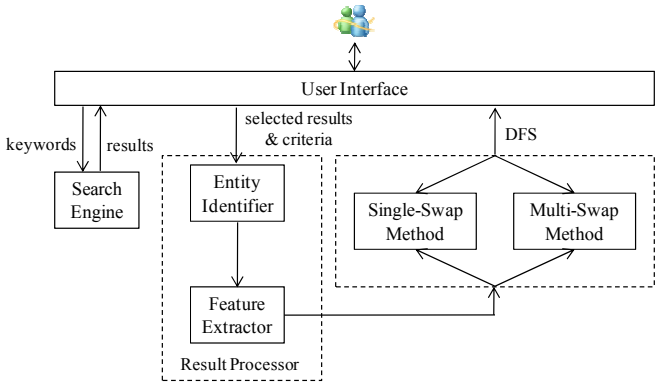


Figure 3: Architecture of XSACT

features, and outputs a Differentiation Feature Set (DFS) for each result to highlight their differences within a size bound. XSACT can handle general structured results, such as relational tuples and XML data, as long as they contain a set of features of form (entity, attribute, value), as to be discussed in Section 2. In the example, XSACT will generate a comparison table for the two results as shown in Figure 2.

To the best of our knowledge, XSACT is the first system that provides automatic result comparison. It can be used to augment any existing search engine for structured data to provide the functionality of helping users easily compare search results.

The technical contributions of XSACT, as detailed in [5] include: (1) We initiate the problem of differentiating structured search results, which is critical for diverse application domains, such as online shopping, employee hiring, job/institution hunting, etc. (2) We identify three desiderata of selecting Differentiation Feature Set (DFS) from search results in order to effectively help users compare and contrast results. (3) We propose an objective function to quantify the degree of differentiation among a set of DFSs, and prove that the problem of identifying valid features that maximize the objective function given a size limit is NP-hard. (4) We tackle the problem efficiently by proposing two local optimality criteria: single-swap optimality and multi-swap optimality, and developing efficient algorithms to achieve these criteria.

2. RESULT COMPARISON

System Architecture. The architecture of XSACT is shown in Figure 3. A User first issues a keyword query to a search engine, which returns a set of results to the user. The user then can select a set of results for comparison, and optionally specify an upper bound of the comparison table size. To make meaningful comparisons, the *result processor* module processes the results and extracts relevant information. Specifically the *entity identifier* infers entities and attributes in the results [3], defined in the spirit of the Entity-Relationship model in relational databases. Then the *feature extractor* identifies features. A feature is defined as a triplet (entity, attribute, value), such as (*Product, Name, TomTom Go 630*), and a feature type as an (entity, attribute) pair, such as (*Product, Name*). Since a result often has a lot of features, the *DFS generator* carefully selects a set of features, Differentiation Feature Set (DFS), which can maximize their differences among the set of selected results. Two alternative modules can generate DFS, *single-swap method* and *multi-swap method*.

Desiderata of Differentiation Feature Set (DFS). We identify a set of desiderata for DFSs: limited size, reasonable summary, and maximal differentiation. Limited size is a necessary condition of a DFS, which ensures that the DFS can be easily checked by a user. Moreover, a DFS should maximize its difference with other DFSs, while at the same time being a reasonable summary of the result to ensure that the differences among DFSs correctly reflects the differences among results.

Desideratum 1: [Being Small] The size of each DFS D, denoted as $|D|$, is defined as the number of features in D. $|D|$ should not

exceed a user-specified upper bound L, i.e., $|D| \leq L$. ∎

For the comparisons based on DFSs to be valid, a DFS should be a reasonable summary of the corresponding result by capturing the main characteristics in the result.

Desideratum 2: [Validity] A feature type is *significant* if it has a large number of occurrences in a result compared with other feature types of the same entity. A DFS is *valid* if feature types are selected into the DFS in the order of their significance, so that it represents a faithful summary of the result. ∎

In Figure 1, we list a table showing the statistic information of each result. The number that follows each feature type indicates the occurrence of that feature type in the result, i.e., the number of reviewers that say Yes to that feature type. For example, for GPS 1, feature type *Pro:Compact* occurs 8/11=73% within its entity (review), indicating that 73% reviewers think this GPS is compact, while feature type *Pro:LargeScreen* only has 9% reviewers think so. Even though *Pro:LargeScreen:Yes* could be used to differentiate this result from another result, GPS 3, which has *Pro:LargeScreen:Yes* occurred 6%, such a comparison does not reflect the most important characteristics of the results.

Furthermore, good DFSs should be able to differentiate results. To illustrate the differentiability of DFSs, let us consider two DFSs, denoted by S_1 and S_3, consisting of exactly the features in the snippets in Figure 1. Several observations can be made: First of all, features of different types are not comparable, e.g., it does not make sense to compare *Product:Name:TomTom Go 630* in S_1 with *Product:Rating:4.1* in S_3. Second, notice that S_1 has *Pro:EasyToRead:Yes* with 91% occurrences, but S_3 does not include features of type *Pro:EasyToRead*. Thus the two results cannot be differentiated by this feature type, as the user does not know whether GPS 3 is easy to read or not. This is analogous to "null" values in databases: the absence of a value only means "unknown", but does not necessarily mean that the value is not what we are looking for.

According to these observations, we define that two results are *comparable* by features of the same type. Two results are *differentiable* by a feature type if their DFSs have different characteristics of this shared feature type. Specifically, DFSs D_1 and D_2 of two results are *differentiable* in a feature type t if and only if there is a feature of t whose occurrences in the two results differ more than x% of the smaller one (threshold x is empirically set to 10% in our system). Then we quantitatively define the *degree of differentiation* of two DFSs $DoD(D_1, D_2)$ as the number of feature types on which the DFSs are differentiable. Accordingly, we define the third desideratum for generating DFSs:

Desideratum 3: [Differentiability] Given a set of results $R_1, R_2, \cdots, R_n$, their DFSs, $D_1, D_2, \cdots, D_n$, should maximize the *total degree of differentiation* computed as the sum of the degree of differentiation of every pair of DFSs:

$$DoD(D_1, D_2, \cdots, D_n) = \sum_{1 \leq i \leq n} \sum_{i < j \leq n} DoD(D_i, D_j)$$

∎

For example, the two DFSs in Figure 1 have a DoD of 2 because only two features types, *Product:Name* and *Pro:Compact*, are differentiable. But for the DFSs generated by our system, shown in Figure 2, their DoD is 5 because three more feature types become comparable.

Problem Definition and NP-hardness. Now we formally define the problem of generating DFSs for search result differentiation. Given a set of results, their DFSs should maximize the DoD, i.e., the total degree of differentiation, and the DFSs should be valid with respect to the corresponding result, and be small.

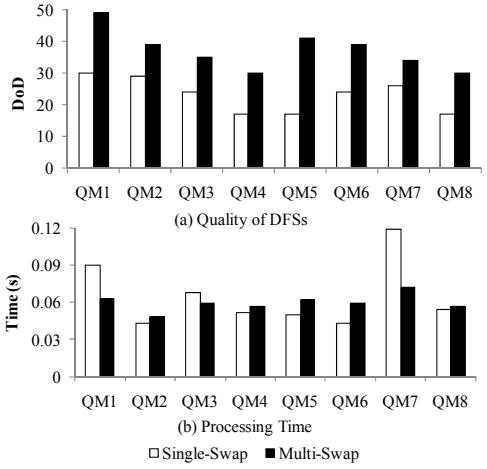


Figure 4: Effectiveness and Efficiency of XSACT

Definition 1: The *DFS construction problem* $(R_1, R_2, \cdots, R_n, m, L)$ is the following: given n search results $R_1, R_2, \cdots, R_n$, each with no more than m feature types, compute a DFS D_i for each result R_i, such that:
(1) $DoD(D_1, D_2, \cdots, D_n)$ is maximized.
(2) $\forall i$, feature types of the same entity in D_i should appear in the order of their number of occurrences in R_i.
(3) $\forall i$, $|D_i| \leq L$. ∎

Theorem 2.1: The DFS construction problem is NP-hard [5]. ∎

Local Optimality and Algorithms. In order to address the problem with good effectiveness and efficiency, we propose two local optimality criteria. A set of DFSs is *single-swap optimal* if by changing or adding one feature in a DFS, while keeping its validity and size limit bound, the degree of differentiation cannot increase. A set of DFSs is *multi-swap optimal* if, by making changes to any number of features in a DFS, while keeping its validity and size limit bound, the degree of differentiation cannot increase. Single-swap optimality can be achieved by iteratively improving a DFS by adding/removing a feature, until it cannot be further improved. Multi-swap optimality is more desirable, while also more challenging to achieve: an intuitive approach that checks every possible combination of features in a DFS involves an exponential time complexity. We proposed a dynamic programming algorithm to achieve it efficiently.

Figure 4 shows the effectiveness and efficiency of the DFSs generated by XSACT over eight queries on a movie data set extracted from IMDB.[1] For each query, we generate DFSs for all its results. As we can see from (a), the multi-swap optimal method generally outperforms the single-swap optimal method in terms of DoD, which is expected since it is able to change multiple features in a DFS at a time, thus increases the chances of achieving a high DoD. Figure 4(b) suggests that the single-swap optimal algorithm is usually more efficient, but the multi-swap optimal algorithm also achieves a better efficiency for some queries. The reason is that although the multi-swap optimal algorithm is more complicated, it increases the DoD faster by adding/changing multiple features in a DFS at a time, thus it may stop faster. It can also be observed that both algorithms are efficient and practical. More experiment

[1] ftp://ftp.sunet.se/pub/tv+movies/imdb/

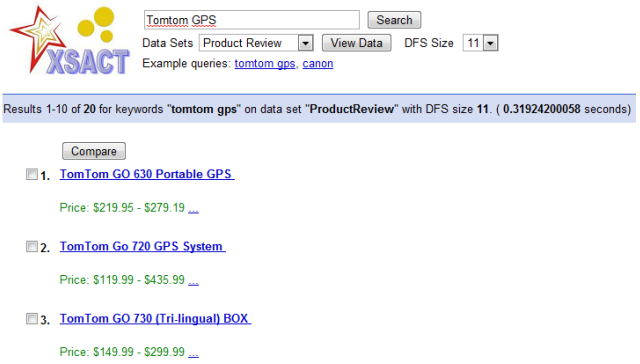


Figure 5: XSACT's Result Page of Query "*TomTom, GPS*"

results can be found in [5].

3. DEMONSTRATION

In the demonstration, we present XSACT, which addresses an open challenge of providing automatic comparisons for keyword search results on structured data. With real datasets in online shopping domain, XSACT shows how result comparison can enhance users' search experiences.

XSACT has a web-based user interface (http://wsdb.asu.edu/xsact). A screen shot of the result page for query {*TomTom, GPS*} is shown in Figure 5. Two real datasets are available in the demo: *Product Reviews* and *Outdoor Retailer*. The Product Reviews dataset, retrieved from www.buzzillions.com, contains a set of GPS, mobile phone and digital camera products, each associated with a price, an aggregated user rating and a set of reviews. Each review consists of the information of the reviewer, as well as a set of features of the product in the reviewer's opinion, such as the pros, cons and best uses. Figure 1 shows fragments of this dataset. The Outdoor Retailer dataset, extracted from www.REI.com, contains a set of brands and products for outdoor recreation and sporting, including bicycles, clothes, footwear, etc. Each brand has a set of products, and each product has a set of features that describes the product. For example, the features of a bicycle include: category, subcategory, gender, number of gears, tires, brakes, frame, etc.

Both datasets are stored in XML format, and XSeek [3, 4] is adopted as the search engine. Other structured form of the data and/or other search engines can also be used as long as the search results contain a set of features.

The demonstration features rich interaction with users. As we can see from Figure 5, XSACT allows users to specify datasets and keywords for retrieval. After the results are generated, their snippets, such as product names and prices, are listed. If the user wants to know more about a specific result, s/he can click the name of the result and the entire result will be shown. Each result has a hyperlink back to the corresponding page at Buzzillions.com or REI.com, so that the user can obtain more information if needed. Given the results, the user can identify which results they are interested in comparing by clicking the checkboxes right before query results (similar as what some bank websites do to let the user compare selected credit cards). Also, the user can optionally specify the size limit of the comparison table. Then after clicking the "comparison" button above the results, a comparison table will be shown in a new browser window, which consists of a DFS for each result, such as the one shown in Figure 2.

Through the demo, the XSACT system will show the feasibility and advantages of automatically generating meaningful and informative comparison tables for structured search results. For example, for Product Review data, a user can manually compare results, which can be very time-consuming as each result may contain hundreds of features. Alternatively, a product review comparison website, buzzillions.com, generates comparison tables literally containing all the features of user selected products. However, it is unable to generate a comparison table which can highlight the differences among products, with a size controlled by the user. On the other hand, using XSACT, comparison tables will be generated dynamically based on the selected products, as shown in the example in Figure 2. XSACT enables the user to easily compare products by selecting features that can maximally differentiate the selected products within a user specified table size upper bound.

On the Outdoor Retailer dataset, XSACT allows users to see the different focuses of different brands through the generated comparison table, which will guide the user's choice of brands when purchasing a particular product, without manually checking the hundreds of products of each brand. For example, if a male user wants to buy a jackets and issues a query "men, jackets", then each result will be a brand selling men's jackets, as well as the products of the brand. From the comparison table of the selected results, the user will learn, for example, brand "Marmot" mainly sells rain jackets, while brand "Columbia" focuses on insulated ski jackets. If a rain jacket is what the user wants, he will likely give a higher priority to "Marmot" as their rain jacket products are likely more reliable. Again, the comparison table generated by XSACT saves a user's time of browsing all selected results and manually analyzing all the products of each brand.

This demonstration will also present the challenges involved, including both the technical challenges discussed in [5] and the ones encountered to handle diverse datasets when building XSACT. For example, due to the large size of the two datasets (a product can have hundreds of reviews in the Product Review dataset, and a brand can have hundreds of products in the Outdoor Retailer dataset), it is demanding to generate meaningful comparison tables in a short period of time. Besides, the large variety of feature types in both datasets invalidates the approach that manually pre-defines the comparison metrics and shows the advantage of automatic generation of comparison tables.

In summary, the XSACT system helps users compare structured search results by generating a set of differentiation feature sets. It can be used to augment existing keyword search engines on structured data to enhance user search experience. Result differentiation is very useful in keyword search applications due to the large presence of exploration queries. It combines with other techniques such as query cleaning, database selection, result generation, result ranking, result snippet generation, etc., to form a suite of techniques in building a full-fledged keyword search engine for structured data. Result differentiation also calls for future works from the database community, such as considering more factors (e.g., interestingness) when selecting features for DFS, and better algorithms (such as one with a guaranteed approximation ratio) for the DFS generation problem.

4. REFERENCES

[1] A. Broder. A Taxonomy of Web Search. *SIGIR*, 2002.
[2] Y. Huang, Z. Liu, and Y. Chen. Query Biased Snippet Generation in XML Search. In *SIGMOD*, 2008.
[3] Z. Liu and Y. Chen. Identifying Meaningful Return Information for XML Keyword Search. In *Proceedings of SIGMOD*, 2007.
[4] Z. Liu and Y. Chen. Reasoning and Identifying Relevant Matches for XML Keyword Search. In *VLDB*, 2008.
[5] Z. Liu, P. Sun, and Y. Chen. Structured search result differentiation. *PVLDB*, 2(1):313–324, 2009.