DivDB: A System for Diversifying Query Results

Marcos R. Vieira[1], Humberto L. Razente[2], Maria C. N. Barioni[2],
Marios Hadjieleftheriou[3], Divesh Srivastava[3], Caetano Traina Jr.[4], Vassilis J. Tsotras[1]

[1]UC Riverside – USA, [2]UFABC – Brazil, [3]AT&T Labs – USA, [4]USP São Carlos – Brazil

{mvieira,tsotras}@cs.ucr.edu, {humberto.razente,camila.barioni}@ufabc.edu.br,
{marioh,divesh}@research.att.com, caetano@icmc.usp.br

ABSTRACT

With the availability of very large databases, an exploratory query can easily lead to a vast answer set, typically based on an answer's *relevance* (i.e., *top-k*, *tf-idf*) to the user query. Navigating through such an answer set requires huge effort and users give up after perusing through the first few answers, thus some interesting answers hidden further down the answer set can easily be missed. An approach to address this problem is to present the user with the most *diverse* among the answers based on some diversity criterion. In this demonstration we present DivDB, a system we built to provide query result diversification both for advanced and novice users. For the experienced users, who may want to test the performance of existing and new algorithms, we provide an SQL-based extension to formulate queries with diversification. As for the novice users, who may be more interested in the result rather than how to tune the various algorithms' parameters, the DivDB system allows the user to provide a "hint" to the optimizer on speed vs. quality of result. Moreover, novice users can use an interface to dynamically change the tradeoff value between relevance and diversity in the result, and thus visually inspect the result as they interact with this parameter. This is a great feature to the end user because finding a good tradeoff value is a very hard task and it depends on several variables (i.e., query parameters, evaluation algorithms, and dataset properties). In this demonstration we show a study of the DivDB system with two image databases that contain many images of the same object under different settings (e.g., different camera angle). We show how the DivDB helps users to iteratively inspect diversification in the query result, without the need to know how to tune the many different parameters of the several existing algorithms in the DivDB system.

1. INTRODUCTION

Enabling diversity on query results has recently attracted interest for applications that may return large answer sets. Examples range from exploratory and ambiguous keywords searches (e.g., apple, jaguar, eclipse) [1, 8] to diversification of structured databases [3, 6]. Typically, the final result set is computed in two phases. First, a ranking *candidate set* S with elements that are relevant (or similar) to the user's query is retrieved using standard database/information retrieval techniques (e.g. top-k, tf-idf). Then, in the second phase, a *result set* $R \subseteq S$ is computed that returns to the user the most *diverse* (based on some user defined diversity threshold) among the *relevant* elements to the query.

Several techniques have been introduced for diversifying query results, with the majority of them exploring a greedy solution that builds the result set in an incremental way [2, 5, 10]. The element attribute sets on which relevance (or similarity) and diversity are computed may intersect, thus techniques for query result diversification attempt to find a *tradeoff* between the *relevance* and *diversity* components. In our work we model the query result diversification problem as a bi-criteria optimization problem $\mathcal{F}$, where the goal is to find a result set R of size k that maximizes the objective function $\mathcal{F}$, as detailed in Section 2. An advantage of having a *tradeoff* parameter is that users can "tune" between how much relevance or diversity they want in the final result.

This demonstration presents the DivDB system, where using a SQL-like language, experienced users can express how much the results should be diverse, the tradeoff parameter between diversity and relevance, as well as other parameter settings. An example query appears in Figure 1, where the IMAGES database is searched for 2000 images based on a similarity function *sim* to the query image q and it returns the 5 images that maximize $\mathcal{F}$ using threshold between diversity and similarity $\lambda = 0.3$. The DivDB system provides several diversification algorithms in a common framework, where their *speed* and *quality* of the result vary depending on the query parameters. Typically, faster algorithms provide results that may not be the most diverse among all results. Hence in DivDB, the user can provide a *hint* to the system with respect to the *quality* vs. *speed* tradeoff. In Figure 1, the user hint *quality* lets DivDB pick a diversification algorithm that provides better result quality at the expense of speed.

```
select * from (select * from IMAGES
               order by sim(IMAGES,q)
               stop after 2000) as R
group by makeset(5) as S
order by F(S, 0.3, div)
stop after 1
hint 'quality'
```

Figure 1: Example of SQL-like diversity query.

Proceedings of the VLDB Endowment, Vol. 4, No. 12
Copyright 2011 VLDB Endowment 2150-8097/11/08... $ 10.00.

In the DivDB demo users are able to compare the performance of the different diversification algorithms. Given a query, we will show how the different *hints* and threshold λ values affect the results. Moreover, the DivDB system is the first of its kind to provide a SQL-like interface where users can write their own queries. Even though there are several diversification algorithms proposed so far, we are not aware of any system, like DivDB, that provides several different algorithms in a common framework.

Next, we provide a brief overview of the diversity problem and present an outline of the DivDB system. Section 3 describes the different scenarios that will be presented in our demonstration.

2. RESULT DIVERSIFICATION IN DIVDB

In this section we provide a brief description on how the result diversification is computed in the DivDB system (see [9] for more details). Let $S = \{s_1, ..., s_n\}$ be a set of n elements, q a query element and k an integer ($k \leq n$). In DivDB the *Result Diversification Problem* is implemented as an optimization problem (i.e., computing the *k-similar diversification set* R) where there is a *tradeoff* between the similarity component and the diversity one. The relevance (similarity) of each element $s_i \in S$ is specified by a function $\delta_{sim}(q, s_i)$, where $\delta_{sim} : q \times S \rightarrow \mathbb{R}^+$. The diversity between two elements $s_i, s_j \in S$ is specified by the function $\delta_{div}(s_i, s_j) : S \times S \rightarrow \mathbb{R}^+$.

The similarity component finds the k elements from S that have the largest sum of similarity distances to query q than any other set of size k in S:

$$\underset{S' \subseteq S, k=|S'|}{\text{argmax}} \quad sim(q, S') \tag{1}$$

where $sim(q, S') = \sum_{i=1}^{k} \delta_{sim}(q, s_i), s_i \in S'$. Intuitively, $sim(q, S')$ measures the amount of "attractive forces" between q and k elements in S'. Basically, any algorithm that can rank elements in S with respect to δ_{sim} and then extract the top-k elements in the ranked list can evaluate the k-similar set problem. The diversity component finds the k elements of S that maximize the sum of inter-element distances:

$$\underset{S' \subseteq S, k=|S'|}{\text{argmax}} \quad div(S') \tag{2}$$

where $div(S') = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \delta_{div}(s_i, s_j), s_i, s_j \in S'$. Intuitively, $div(S')$ measures the amount of "repulsive forces" among k elements in S'. In both Definitions 1 and 2, other objective functions could be used as well, e.g., max-min, max-avg, min-max, where the most appropriate measure is application dependent.

DEFINITION 1. *Given a* tradeoff λ, $0 \leq \lambda \leq 1$, *between similarity and diversity, the* ***k-similar diversification set*** *R contains k elements in S such that:*

$$R = \underset{S' \subseteq S, k=|S'|}{\text{argmax}} \quad \mathcal{F}(q, S')$$

where $\mathcal{F}(q, S') = (k-1)(1-\lambda) \cdot sim(q, S') + 2\lambda \cdot div(S')$

Since components *sim* and *div* have different number of elements, k and $\frac{k(k-1)}{2}$, respectively, in the above definition the two components are scaled up. The variable λ is a *tradeoff* specified by the user, and it gives the flexibility to return more relevant (when λ is small), or diverse (when λ is large), results for a given query q. The *k-similar diversification set* problem is reduced to the *k-similar set* (Equation 1) when $\lambda = 0$; here the result set R depends only on the query q. When $\lambda = 1$, the problem is reduced to finding the *k-diverse set* (Equation 2) in S (i.e., without considering similarity). Only when $\lambda = 0$ the result R is straightforward to compute. For any other case, $\lambda > 0$, the problem is computationally hard, and a number of algorithms can be employed to compute approximate solutions.

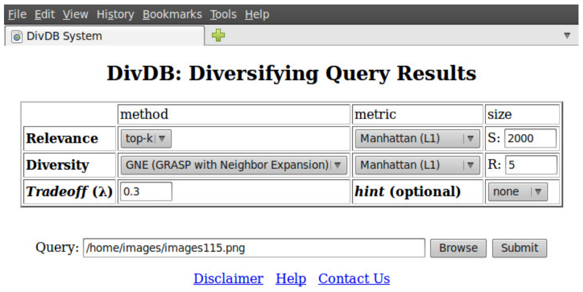
abbrv.	method name	construction of R
Swap	*Swap*	exchanging
BSwap	*BSwap*	exchanging
MMR	*Maximal Marginal Relevance*	incremental
Motley	*Motley*	incremental
MSD	*Max-Sum Dispersion*	incremental
CLT	*Clustering*	exchanging
GMC	*Greedy Marginal Contribution*	incremental
GNE	*GRASP with Neighbor Expansion*	meta-heuristic

Table 1: Available diversifying methods in DivDB.

The collection of algorithms implemented in DivDB is shown in Table 1 (the strategy employed by each algorithm is depicted as well). Based on the heuristic used, some approaches focus on *speed* while others on the *quality* of the diversified result. Thus the user must provide a "hint" to the DivDB together with the query. In the current implementation of DivDB, we distinguish between three hints, namely: *quality*, *speed* and *method*. For the last hint, the experienced user can select a specific method among the ones available in the DivDB. Thus, the user can compare the performance (e.g., running time, precision) among the different methods, as well as new algorithms to be implemented in the DivDB system. For the *quality* and *speed* hints, the DivDB query optimizer selects, based on the hint value and query parameters (e.g., λ, n, k), which algorithm to execute.

3. DEMONSTRATION

In our demonstration we will use two real image datasets, ALOI [4] and Faces [7]. The ALOI dataset is a collection of 72,000 color images of 1,000 objects using different camera views. The Faces dataset is a collection of 300 gray images of 30 people's faces with slight variations of the head positions. To compute the δ_{sim} and δ_{div} functions, we use the histogram of colors and gray-level for ALOI and Faces, respectively. Since in both datasets there are several images that are very similar to each other, we should expect in our demonstration that increasing the λ values, more images of different objects appear in the result.

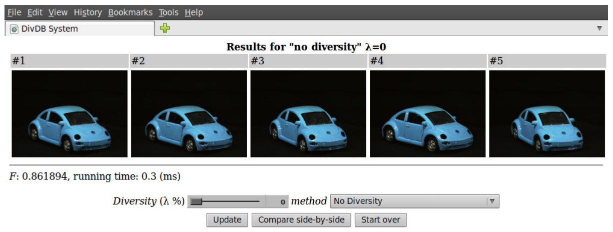


DivDB: Diversifying Query Results

	method	metric	size
Relevance	top-k	Manhattan (L1)	S: 2000
Diversity	GNE (GRASP with Neighbor Expansion)	Manhattan (L1)	R: 5
***Tradeoff* (λ)**	0.3	***hint* (optional)**	none

Query: /home/images/images115.png

Figure 2: The DivDB System Interface.

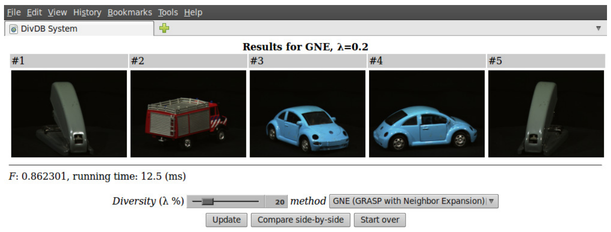


Results for "no diversity" λ=0

F: 0.861894, running time: 0.3 (ms)

Diversity (λ %) 0 *method* No Diversity

(a) $\lambda = 0$ (only top-5).

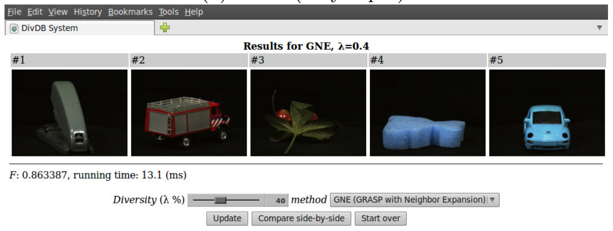


Results for GNE, λ=0.2

F: 0.862301, running time: 12.5 (ms)

Diversity (λ %) 20 *method* GNE (GRASP with Neighbor Expansion)

(b) *GNE* with $\lambda = 0.2$ (low diversification).

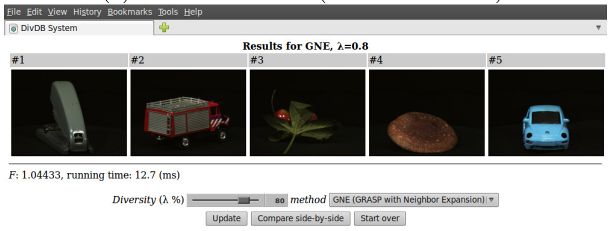


Results for GNE, λ=0.4

F: 0.863387, running time: 13.1 (ms)

Diversity (λ %) 40 *method* GNE (GRASP with Neighbor Expansion)

(c) *GNE* with $\lambda = 0.4$ (moderate diversification).



Results for GNE, λ=0.8

F: 1.04433, running time: 12.7 (ms)

Diversity (λ %) 80 *method* GNE (GRASP with Neighbor Expansion)

(d) *GNE* with $\lambda = 0.8$ (high diversification).

Figure 3: A Sequence of results for ALOI when the user varies λ from 0 to 0.8.

Besides supporting a SQL-like language, the DivDB system also provides users with a GUI to formulate queries with diversification. In summary, the main interface translates the user's query to its SQL representation as described in Section 1. In our demonstration we will focus only on this interface. Using the main interface exemplified in Figure 2, users can select each of the query parameters: the relevance (e.g. *top-k*) and diversity algorithms (e.g. *MMR*, *GNE*, *MSD*), the δ_{sim} and δ_{div} metrics (e.g. cosine, Euclidean distance), the size of S and R, and the diversity threshold λ. Instead of selecting a specific diversity method, users can instead select the *hint* parameter and let DivDB pick an appropriate algorithm.

The user's query is converted to its SQL representation and then passed to the query evaluation module. In this module, the first step retrieves in the database, using the relevance method, the candidate set S with the n most relevant elements, according to the δ_{sim} metric. After S is computed, one of the evaluation algorithms shown in Table 1 is employed to find k diversified elements R.

If the *hint* parameter is specified, then a heuristic-based rule is used to choose one of the diversifying algorithms in Table 1. Our current heuristic uses the *hint* and λ values (more elaborate optimizations can also be employed). For instance, when *hint* is 'speed' and $\lambda = 0.3$, the *MMR* method is chosen since it is the fastest method that does not compromise the quality of the result for low values of λ.

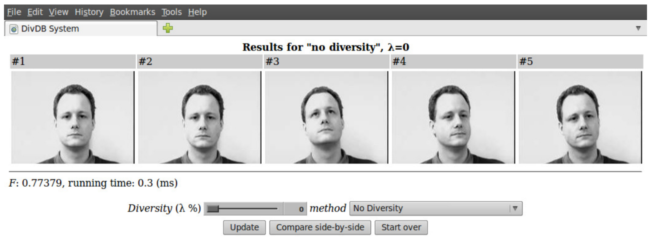
After the query execution, the DivDB system allows the user to update the amount of diversity by changing the λ parameter. This is an important feature since it gives users the flexibility to iteratively increase/decrease the amount of diversity in the result. An example of this feature is shown in Figure 3, for $k = 5$, $S = 200$ and a selected query image from the ALOI dataset using the *GNE* algorithm. In the first result set (Figure 3(a)), no diversification is specified ($\lambda = 0$), and thus the result contains only the *top-k* results. In the next three figures, the user updates the λ value to 0.2 (*low diversification*, Figure 3(b)), 0.4 (*moderate diversification*, Figure 3(c)) and 0.8 (*high diversification*, Figure 3(d)). Since both datasets contain several images that are very similar to each other, the result sets with *no diversification* contain several "almost identical" images (several images of the same object but in a different perspective). As λ increases more diverse results (based on *div*) start to appear. Observe that the result with *high diversification* contains images that are all different from each other, but yet somehow relevant to the user (based on the function *sim*).

Figure 4 shows a user interaction example using the Faces dataset with $k = 5$ and $S = 300$. In this example, λ has values 0, 0.1, 0.3, 0.5, 0.7 and 0.9. Distinct from the previous example, the λ parameter has to be set to a higher value in order to return faces of all different people ($\lambda = 0.9$), indicating that tuning the query parameters with proper values depends on different settings and on the user's expectations. Nevertheless, DivDB allows users to iteratively change the query parameters and inspect the changes in the result.

Moreover, we will show how results vary for different hint choices. Figure 5 shows the same query but with different *hint* parameters: *speed* and *quality*. Users can inspect how the *hint* parameter affects the $\mathcal{F}$ values, running time, and precision (the last compared to the optimal solution computed for small R and k values). Another feature of the DivDB system is to allow a *side-by-side* comparison of two methods for the same query parameters (due to space limitations this part will be shown only at the demonstration).

4. CONCLUSION

The DivDB is designed to provide a very flexible platform for diversifying query results. Since the performance and result of each method varies for different query parameters, and typically users are not aware of which method is the most suitable, we developed some heuristic-based rules to guide DivDB in choosing the most adequate diversification algorithm to run. As for more experienced users, we provide a SQL-like language that allows users to test the performance of different parameters for several different diversification algorithms implemented in DivDB, as well as future algorithms. Users can iteratively change the amount of diversity in the result by controlling the λ values, and then compare, *side-by-side*, the result and performance of different diversifying methods. Furthermore, users can use the *hint* feature of our system to control the performance of

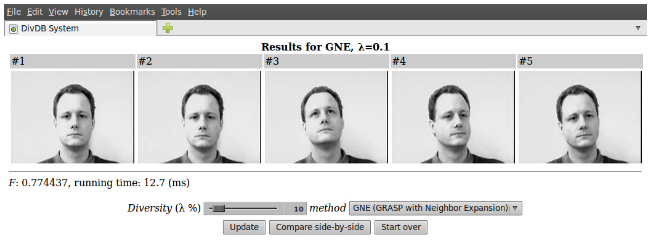


Results for "no diversity", λ=0

F: 0.77379, running time: 0.3 (ms)

Diversity (λ %) 0 method No Diversity

(a) $\lambda = 0$ (only top-5).

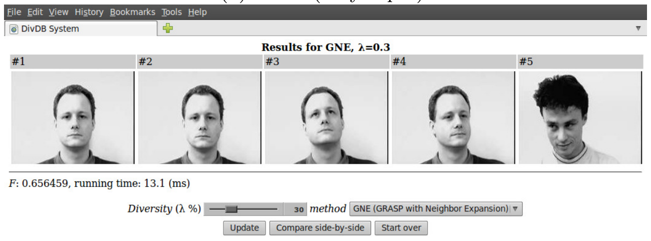


Results for GNE, λ=0.1

F: 0.774437, running time: 12.7 (ms)

Diversity (λ %) 10 method GNE (GRASP with Neighbor Expansion)

(b) *GNE* with $\lambda = 0.1$ (very low diversification).

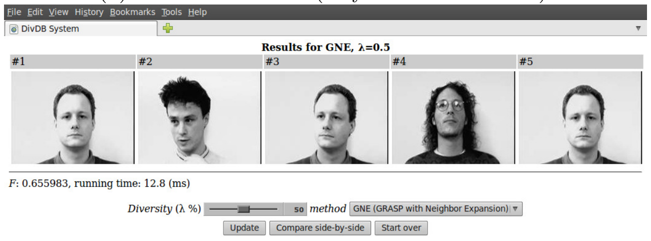


Results for GNE, λ=0.3

F: 0.656459, running time: 13.1 (ms)

Diversity (λ %) 30 method GNE (GRASP with Neighbor Expansion)

(c) *GNE* with $\lambda = 0.3$ (low diversification).

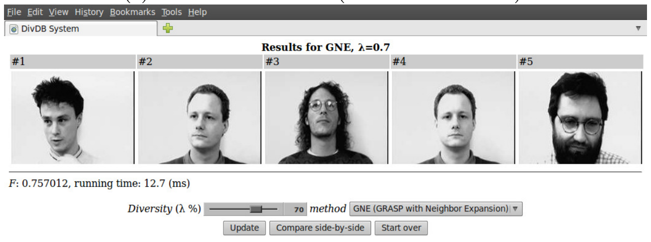


Results for GNE, λ=0.5

F: 0.655983, running time: 12.8 (ms)

Diversity (λ %) 50 method GNE (GRASP with Neighbor Expansion)

(d) *GNE* with $\lambda = 0.5$ (moderate diversification).

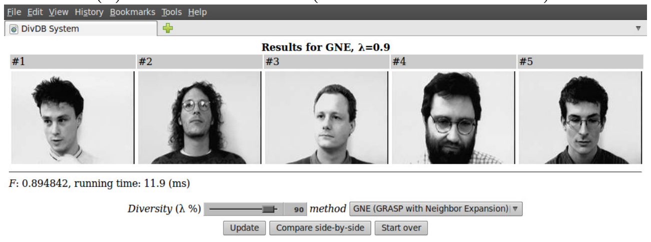


Results for GNE, λ=0.7

F: 0.757012, running time: 12.7 (ms)

Diversity (λ %) 70 method GNE (GRASP with Neighbor Expansion)

(e) *GNE* with $\lambda = 0.7$ (high diversification).



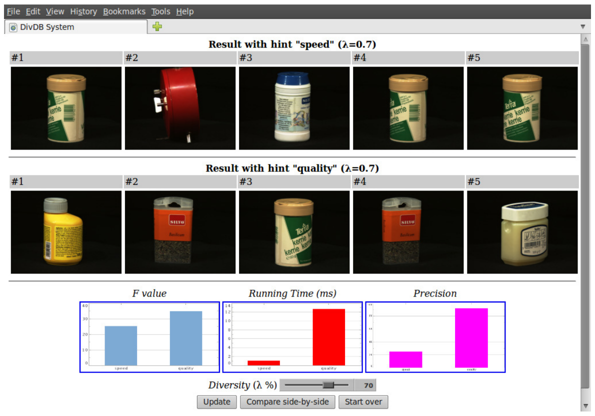
Results for GNE, λ=0.9

F: 0.894842, running time: 11.9 (ms)

Diversity (λ %) 90 method GNE (GRASP with Neighbor Expansion)

(f) *GNE* with $\lambda = 0.9$ (very high diversification).

Figure 4: A Sequence of results for Faces when the user varies λ from 0 to 0.9.



Result with hint "speed" (λ=0.7)

Result with hint "quality" (λ=0.7)

Diversity (λ %) 70

Figure 5: Comparison of different "hints".

the query engine.

To the best of our knowledge, DivDB is the first system allowing users to compare different diversifying algorithms, as well as providing an interface to allow users to inspect the result while tuning the query parameters. These two features are very important to the user since the result depends on several variables that are very difficult to tune, and may give clues to research on what must be explored on the development of new algorithms, in order to conceal the drawbacks of the existing ones.

Acknowledgements: This research was partially supported by NSF IIS grants 0705916, 0803410 and 0910859. Vieira's work was funded by a CAPES/Fulbright Ph.D fellowship. Razente's work was funded by a Fapesp Ph.D. scholarship 06/00336-5.

5. REFERENCES

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *Proc. of the ACM Int'l Conf. on Web Search and Data Mining (WSDM)*, pages 5–14, 2009.
[2] J. Carbonell and J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proc. of the Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 335–336, 1998.
[3] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl. DivQ: Diversification for keyword search over structured databases. In *Proc. of the Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 331–338, 2010.
[4] J.-M. Geusebroek, G. J. Burghouts, and A. W. M. Smeulders. The Amsterdam Library of Object Images. *Int. J. Comput. Vision*, 61(1):103–112, 2005.
[5] A. Jain, P. Sarda, and J. Haritsa. Providing diversity in k-nearest neighbor query results. In *Advances in Knowledge Discovery and Data Mining (PAKDD)*, volume 3056 of *Lecture Notes in Computer Science*, pages 404–413, 2004.
[6] Z. Liu, P. Sun, and Y. Chen. Structured search result differentiation. *Proc. VLDB Endow. (PVLDB)*, 2(1):313–324, August 2009.
[7] U. of Bern. ftp.iam.unibe.ch/pub/Images/FaceImages, 1995.
[8] D. Rafiei, K. Bharat, and A. Shukla. Diversifying web search results. In *Proc. of the Int'l Conf. on World Wide Web (WWW)*, pages 781–790, 2010.
[9] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. T. Jr., and V. J. Tsotras. On query result diversification. In *Proc. of the Int'l Conf. on Data Engineering (ICDE)*, pages 1163–1174, 2011.
[10] C. Yu, L. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *Int'l Conf. on Extending Database Technology (EDBT)*, pages 368–378, 2009.