Bulletin of the Technical Committee on

Data Engineering

December 2009 Vol. 31 No. 4

IEEE Computer Society

Letters

Special Issue on Result Diversity

Conference and Journal Notices

The TC on Data Engineering

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TC on Data Engineering web page is
http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at
http://tab.computer.org/tcde/bull_about.html.

Distribution

IEEE Computer Society
1730 Massachusetts Avenue
Washington, D.C. 20036-1992
(202) 371-1013
jw.daniel@computer.org

Letter from the Editor-in-Chief

About the Bulletin

Work is progressing on converting issues from 1998 onward to a new web page format for accessing individual papers. I would urge you to check out past issues in addition to the current issue to take note of the progress. While more remains to be done, all pages now have a standard, well formatted header. Going forward, this should also (*eventually*) reduce the work I need to do to generate an issue, as pages on both the Computer Society site and the Microsoft site are now identical.

Please do take a look at the web pages providing access to individual papers, and report any problems to me. Thank you.

I am considering dropping some versions of past issues from the web site. Before doing so, I wanted to give you advance notice so that you could provide input to this process. In particular, I plan to drop all the "draft" postscript versions from the web site. I may go on to drop "final" postscript versions as well. That would leave only the PDF version. Essentially everyone has access to a PDF reader, and PDF, being compressed, downloads faster. Again, please provide input to this process, especially if you would like to continue accessing postscript versions.

The Current Issue

The "search" area continues to be of intense interest, both as a technical challenge and as a competitive arena. There is strong recognition that not everyone is interested in the same topic when a collection of key words are entered for search. There is much to be gained in making sure that other meanings are considered, and in many cases, that appropriate pages are also returned as results.

It is not only the case that diversity of result increases the likelihood of a web surfer finding the information desired. It is also true that search providers, when recognizing this diversity, increase the likelihood of placing adds of interest to the web surfer. This translates into more profit, and drives what seens to me to be the virtuous cycle of improved technology helping users and also rewarding commercial ventures that provide it.

In this fast moving field, today's research leads quickly to tomorrow's search offering. Vassilis Tsotras and Marios Hadjieleftheriou have succeeded in bringing together research from both academic and industrial researchers in this wide ranging snapshot of the current state of search and its diverse answers. As I have said before, this is something that the Bulletin strives to do, providing a view of a field that is difficult to find in the more traditional venues. I want to thank both Vassilis and Marios for their hard work and success with this issue. I am sure you will find reading it to be valuable.

Self-managing Database Systems Workshop

I want to draw your attention to the report on the 4th workshop on "self managing database systems". The Technical Committee on Data Engineering has one sub-committee, and it is on self-managing database system. This sub-group organizes an annual workshop held in conjunction with the ICDE conference. This is an active group, and reading the workshop report is a good way to stay up-to-date on this important area.

David Lomet
Microsoft Corporation

Report: 4th Int'l Workshop on Self-Managing Database Systems (SMDB 2009)

Ashraf Aboulnaga and Kenneth Salem
University of Waterloo
{ashraf,kmsalem}@uwaterloo.ca
http://db.uwaterloo.ca/tcde-smdb/smdb09

1 Introduction

The Fourth International Workshop on Self-Managing Database Systems took place on March 29th, 2009 in Shanghai, China, on the day before ICDE. The SMDB workshops bring together researchers and practitioners interested in making data management systems easier to deploy and operate effectively. Topics of interest range from "traditional" management issues, such as database physical design, system tuning, and resource allocation to more recent challenges around scalable and highly-available data services in cloud computing environments. The SMDB Workshops are sponsored by the IEEE TCDE Workgroup on Self-Managing Database Systems.

The SMDB 2009 program began with a keynote presentation by James Hamilton, Vice President and Distinguished Engineer with Amazon Web Services. This was followed by five technical papers, presented in two sessions. The workshop concluded with a panel discussion on "grand challenges" in database self-management. More information about the workshop program, including links to papers and presentation slides, can be found at the SMDB 2009 web site, at http://db.uwaterloo.ca/tcde-smdb/smdb09.

2 Keynote

James Hamilton is a Vice President and Distinguished Engineer with Amazon Web Services. His background includes 15 years in database engine development with IBM and Microsoft and 5 years in the cloud service space with Microsoft and Amazon. He is also the author of widely-read blog, at http://perspectives.mvdirona.com/. James' keynote presentation was titled "Cloud-Computing Economies of Scale". It covered hardware trends affecting cloud computing and a variety of useful techniques for scaling cloud services.

James began his presentation with a discussion of the economies of scale in networking, storage, and administration that are achievable by very large scale cloud service providers like Amazon. He observed that while the largest cost component in enterprise data centers is typically people (administrators), this is not true for very large scale cloud service providers. For these organizations, costs are dominated by hardware, cooling, power and power distribution. His figures showed power and power-related costs approaching 50% of total expenses and trending upwards, while hardware costs are trending down.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

James next addressed the question of what ultimately limits the application of arbitrarily large amounts of computing power to cloud applications. He argued that one limiting factor is power, since it is likely to become the dominant cost component for cloud providers. The other limiting factor is communications - the need to get data to the processors. In James' words, the "CPUs are outpacing all means to feed them". He noted the widening "storage chasm" between DRAM and disks, which is encouraging larger memories and more disks. Finally, he noted the large and growing gap between sequential and random disk bandwidth.

In the last part of his presentation, James covered a variety of useful techniques for building scalable services. First, since scalable and highly-available services demand partitioning and redundancy, it is necessary to build such services on top of large numbers of unreliable, distributed hardware and software components. James argued that recovery-oriented computing [1] is the only affordable administrative model when there are many components. This involves deeply monitoring applications and treating failures as routine occurrences to be handled, when possible, by simple automated recovery actions, such as rebooting and re-imaging. Human administrators become involved only if these automated steps do not work. Speaking of the limits of auto-administration, James noted that one should never allow an automated administration system to make a decision that would not be entrusted to a junior operations engineer.

Second, there are many techniques that can be used to lessen the impact of the disk/DRAM chasm, including sequentialization of I/O workloads, the use of distributed memory caches, I/O cooling, and I/O concentration. On the subject of solid state storage devices (SSDs), James urged caution, noting that they should be used only if they produce a win in terms of work-per-dollar or work-per-joule. He argued that SSDs are "wonderful for some workloads", not plain wonderful.

Third, we should expect user data to be pulled closer to the edges of the network to support highly interactive applications and because of political and other restrictions on data movement. Conversely, we should expect aggregated data to be pulled towards the network core where it can be subjected to analysis. While growth in transactional workloads tends to be related to business growth, growth in analysis workloads is linked to the cost of the analyses. As the cost of analysis drops, more types of analyses become economically viable.

Fourth, the spread between average and peak loads is high, so unmanaged load spikes will result in overload. Admission controls or service degradations are necessary to manage overload conditions. There are considerable potential benefits around resource consumption shaping, e.g., "pushing" peak loads into load troughs.

James concluded by noting that utility computing is here to stay because of the cost advantages he had already described, and that the most difficult aspect of providing cloud computing services is "managing multi-center distributed partitioned redundant, data stores and caches", which "really is a database problem."

3 Paper Sessions

The five papers presented at the workshop dealt with a wide range of research issues related to self-managing database systems. Belknap et al. [2] presented a fully automated workflow for tuning SQL queries that is part of Oracle Database 11g. The goal of this work is to automate the invocation of the SQL Tuning Advisor that is available in Oracle for tuning SQL statements, but that had to be invoked manually by the database administrator. The work identifies the SQL statements in the user workload that would be good candidates for automatic tuning. To ensure that recommended tuning actions do indeed result in improved performance, an interesting approach is adopted in which candidate query execution plans recommended by the tuner are executed and their actual performance is measured to decide the best tuning action. Some safeguards are put in place to ensure that the tuner itself does not consume too much time and resources.

Thiem and Sattler [3] presented integrated performance monitoring in the Ingres database system. This is an approach for monitoring performance data at a high resolution from within the database system, with the goal of providing a basis for better-informed automatic tuning decisions. The collected data includes CPU time, number of I/O's, wall clock time, indexes used, and other performance metrics. A relational view over the in-

memory monitoring data is provided to users and tuning tools, and the data is also written to an on-disk workload database. The authors observe that collecting the monitoring data adds little overhead to the system, but writing it to the workload database is expensive, so a better mechanism for persisting the monitoring data is still needed.

Dash et al. [4] proposed an economic model for self-tuned caching in the cloud, specifically targeting scientific applications. The setting considered by the paper is one in which scientific data is stored in a cloud of databases, and users pose queries against this data. The focus of the paper is developing an economic model for deciding which index structures to create for the offered user queries. This model balances the utility of an index structure to user queries against the cost of creating this index structure and the amount that users are willing to pay for different levels of service.

Schnaitter and Polyzotis [5] presented a benchmark for evaluating online index tuning algorithms. Many algorithms for online index selection have been proposed, so a principled approach for evaluating them is definitely welcome. The proposed benchmark models a database hosting environment in which multiple data sets are hosted on one server and multiple workload suites are executed against these data sets. The workload suites include SQL statements of varying complexity, and the benchmark shifts between these workload suites to model dynamic workload behavior. The benchmark's effectiveness was demonstrated by using it to compare two online index selection algorithms implemented in PostgreSQL.

Finally, Hose et al. [6] presented an online approach for selecting materialized views in an OLAP environment. The proposed approach targets OLAP server middleware that uses a data cube as the logical model and translates multi-dimensional queries into SQL queries over a relational star schema. These SQL queries are typically expensive, and materialized views that pre-aggregate the data are often used to speed them up. The paper presents an approach for choosing the set of materialized views for a workload and adjusting this set over time in an online manner. This requires a query representation that identifies how pre-computed aggregates can be used to answer a query, a cost model to estimate the cost of a query if some set of aggregates is materialized, and a search algorithm to identify the aggregates to pre-compute and materialize at each stage of workload execution.

4 Closing Panel

The workshop concluded with a panel on "Grand Challenges in Database Self-Management", with six distinguished panelists: Anastassia Ailamaki from Ecole Polytechnique Federale de Lausanne, Shivnath Babu from Duke University, Nicolas Bruno from Microsoft Research, Benoit Dageville from Oracle, James Hamilton from Amazon, and Calisto Zuzarte from the IBM Toronto Software Lab.

The panelists presented a wide range of views on the research challenges in the area of database self-management. Bruno presented the need for better modeling of query execution times (e.g., by modeling concurrency and locking) and richer workload models (e.g., taking into account stored procedures with conditionals). He mentioned other challenges such as developing benchmarks for physical design tools (the problem addressed in [5]), extending self-management techniques to the cloud, exploiting large user bases to extract data and diagnose problems, and developing a unifying theory for self-tuning database systems.

Dageville presented some of the work that Oracle has recently done on database manageability, covering areas such as statistics collection, SQL tuning, memory management, and SQL repair. Looking ahead, he saw the need for dealing with bad SQL statements from application developers with little database expertise, and for robust query execution plans. He also saw the need for working towards proactively fixing problems before they occur, for example by leveraging the experience of other users. Another important area he identified is extending manageability solution to the entire software stack, beyond just the database system.

Ailamaki focused on using self-managing database systems in e-science. She mentioned that an important challenge in e-science is the sheer scale of the data. She pointed out that in this setting there are many aspects of a database systems that need to be self-managed, such as logical and physical design, power and energy awareness, and resource utilization. The responsibilities of the database management layer also include ensuring policy

compliance, allocating resources for scaling, and handling failures. She listed open problems including dynamic or underspecified workloads, non-relational data models, changes in database statistics, enabling scalability, and finding a path for deploying research ideas to the scientists using e-science systems.

Hamilton pointed out that relational database systems are losing workloads in the cloud. Many cloud applications use database systems, but most of the complexity (e.g., partitioning) is handled above the database system. Furthermore, many new applications use non-relational data models and many interesting workloads have left relational database systems. For example, analysis workloads are being handled by Hadoop, and caching is handled by memcached and similar solutions. Database infrastructures that focus on simplicity and scalability are becoming increasingly popular. These include BigTable, Amazon SimpleDB, Facebook Cassandra, BerkelyDB, and many others. Hamilton identified several problems causing relational database systems to lose ground in the cloud: lack of scalability, excess administrative complexity, resource intensity due to un-needed features, unpredictable response times, excessively random data access patterns, opaque failure modes, and slow adaptability to new workloads. The solution he proposed involves supporting simple caching models and richer execution models, and embracing recovery-oriented computing to improve robustness.

Zuzarte observed that computing has gone from centralized mainframes to distributed client-server systems, and now back to a centralized system with dumb clients talking to a cloud. The challenges he posed include optimizing for objectives other than minimizing elapsed time or resource consumption; for example, minimizing energy consumption or cloud computing dollar costs. He also observed that the Map-Reduce paradigm can benefit from techniques developed in the context of traditional database systems such as sharing scans, progressive re-optimization, and on-the-fly indexing.

Babu advocated using an experiment-driven approach for tuning and problem determination. This approach provides simple and robust solutions to many manageability problems, but it poses several challenges such as identifying the right type of experiment for a given problem and planning the sequence of experiments to conduct. Other challenges include providing an infrastructure for running experiments (e.g., in the cloud) and combining experiment-driven management with current database tuning tools.

5 SMDB 2010

SMDB 2010 is being organized by Shivnath Babu and Kai-Uwe Sattler, and is scheduled for March 1, 2010 in Long Beach, California, immediately preceding ICDE 2010. Its scope includes emerging research areas around database self-management, such as cloud computing, multi-tenant databases, storage services, and datacenter administration. Visit `http://db.uwaterloo.ca/tcde-smdb/smdb10` for more information.

References

[1] D. Patterson et al. Recovery oriented computing (ROC): Motivation, definition, techniques. Technical Report CSD-02-1175, Berkeley, CA, USA, 2002.

[2] Peter Belknap, Benoit Dageville, Karl Dias, and Khaled Yagoub. Self-tuning for SQL performance in Oracle Database 11g. In *Proc. SMDB'09*, pages 1694–1700. IEEE Computer Society, 2009.

[3] Alexander Thiem and Kai-Uwe Sattler. An integrated approach to performance monitoring for autonomous tuning. In *Proc. SMDB'09*, pages 1671–1678. IEEE Computer Society, 2009.

[4] Debabrata Dash, Verena Kantere, and Anastasia Ailamaki. An economic model for self-tuned cloud caching. In *Proc. SMDB'09*, pages 1687–1693. IEEE Computer Society, 2009.

[5] Karl Schnaitter and Neoklis Polyzotis. A benchmark for online index selection. In *Proc. SMDB'09*, pages 1701–1708. IEEE Computer Society, 2009.

[6] Katja Hose, Daniel Klan, and Kai-Uwe Sattler. Online tuning of aggregation tables for OLAP. In *Proc. SMDB'09*, pages 1679–1686. IEEE Computer Society, 2009.

Letter from the Special Issue Editors

Users typically search the web by entering a small number of keywords in a search engine. The search engine uses an inverted index to identify all documents containing the keywords and ranks the results given some relevance function that depends on many different factors, including the query context, the application domain, location, temporal criteria, and more. Nevertheless, natural language is inherently ambiguous hence, to complicate matters further, expressing a query using a few keywords only, results in even larger inherent ambiguity and loss of relevant context. If a search engine does not consider the Diversity of query results during ranking, especially for queries with large ambiguity, the results produced will be unsatisfactory with high probability. A simple example is one of the most popular queries in YellowPages.com, Yahoo! Local and Google Maps: "Restaurant". Depending on the query location, a typical search engine might rank a large number of Italian or Chinese restaurants before the first Thai restaurant appears. Clearly, the user intent for this particular query is not known, but it is safe to assume that since the user did not explicitly specify a particular cuisine, he/she might be interested in all options available in the vicinity of the query. It can be argued that diversification of results will invariably result in increased user satisfaction.

Query result diversity has been extensively studied the past few years and has gained traction in the industrial sector as well (for example, all major search engines implement some form of diversity ranking). Moreover, diversity is not limited to information retrieval. In general, it can be applied to any query that can return a very large number of results. Diversity based queries on traditional database systems, diversity in recommender systems, diversity through user preferences, diversity in publish/subscribe systems, diversity on structured data, and diverse skyline computation using preference functions are some other application domains were diversification plays a big role in improving the end-user experience. This special issue of the *Data Engineering Bulletin* presents a snapshot of the current state-of-art research on diversity for the aforementioned topics.

The first article [Gollapudi and Sharma] presents an axiomatic framework for diversification, formalizing the problem and setting the stage for what follows. The second article [Haritsa] presents a diversity aware algorithm for k-NN processing in traditional database systems. The third [Ziegler and Lausen] and fourth [Amer-Yahia, Lakshmanan, Vassilvitskii and Yu] articles discuss diversification in recommender systems, and more specifically collaborative filtering. The article by [Chen and Li] focuses on efficient query evaluation based on pre-defined user preferences (as a means of expressing diversity). The article by [Drosou and Pitoura] discusses content diversity in the case of continuous data delivery (in the context of publish/subscribe systems). The next article [Vee, Shanmugasundaram, Amer-Yahia] talks about diversification of query results on structured data in the presence of a preference hierarchy of attributes. The final article [Tao] discusses diversification of skyline query results, based on user preference functions.

The goal of this special issue is to instigate further research into the essential problem of diversification. Even though these past few years there has been an impressive number of contributions of high quality into this field, we feel that there is still a lot of room for growth and improvement. It is our belief that current research has only scratched the surface of diversification in the respective domains, and the applications of diversification will keep growing. We would like to thank all authors for the time and effort they put into editing their excellent articles for this special issue. We would also like to extend a special thanks to Marcos Vieira at UC Riverside for editorial assistance with the issue.

Marios Hadjieleftheriou, Vassilis J. Tsotras
AT&T Labs - Research UC Riverside

An Axiomatic Framework for Result Diversification*

Sreenivas Gollapudi
Microsoft Search Labs, Microsoft Research
sreenig@microsoft.com

Aneesh Sharma
Institute for Computational and Mathematical Engineering, Stanford University
aneeshs@stanford.edu

Abstract

Informally speaking, diversification of search results refers to a trade-off between relevance and diversity in the set of results. In this article, we present a unifying framework for search result diversification using the axiomatic approach. The characterization provided by the axiomatic framework can help design and compare diversification systems, and we illustrate this using several examples. We also show that several diversification objectives can be reduced to the combinatorial optimization problem of facility dispersion. This reduction results in algorithms with provable guarantees for a number of well-known diversification objectives.

1 Introduction

In present day search engines, a user expresses her information need with as few query terms as possible. In such a scenario, a small number of terms often specify the intent in only an implicit manner. In the absence of explicit information representing user intent, the search engine needs to "guess" the results that are most likely to satisfy different intents. In particular for an ambiguous query such as *eclipse*, the search engine could either take the probability ranking principle approach of taking the "best guess" intent and showing the results, or it could choose to present search results that maximize the probability that a user with a random intent finds *at least one* relevant document on the results page. This problem of the user not finding any *any* relevant document in her scanned set of documents is defined as *query abandonment*. Result diversification lends itself as an effective solution to minimizing query abandonment [1, 7, 16].

Intuitively, result diversification implies a trade-off between having more relevant results of the "most probable" intent and having diverse results in the top positions for a given query[4, 6]. The twin objectives of being diverse and being relevant often compete with each other, and any result diversification system must figure out how to trade-off these two objectives appropriately. Therefore, result diversification can be viewed as combining both ranking (presenting more relevant results in the higher positions) and clustering (grouping document satisfying similar intents) and therefore addresses a loosely defined goal of picking a set of most relevant but

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*An earlier version appeared in the Proceedings of the 18th International Conference on World Wide Web, 2009 [8].

novel documents. This resulted in the development of a set of very different objective functions and algorithms ranging from combinatorial optimizations [4, 16, 1] to those based on probabilistic language models [6, 20]. The underlying principles supporting these techniques are often different and therefore admit different trade-off criteria. Given the importance of the problem there has been relatively little work aimed at understanding result diversification independent of the objective functions or the algorithms used to solve the problem.

This article discusses an axiomatic framework for result diversification. We begin with a set of simple and natural properties that any diversification system ought to satisfy and these properties help serve as a *basis* for the space of objective functions for result diversification. We then analyze a few objective functions that satisfy different subsets of these properties. Generally, a diversification function can be thought of as taking two application specific inputs *viz,* a relevance function that specifies the relevance of document for a given query, and a distance function that captures the pairwise similarity between any pair of documents in the set of relevant results for a given query. In the context of web search, one can use the search engine's ranking function[1] as the relevance function. The characterization of the distance function is not that clear. In fact, designing the right distance function becomes a central factor to an effective result diversification. For example, by restricting the distance function to a *metric* by imposing the triangle inequality $d(u, w) \leq d(u, v) + d(v, w)$ for all $u, v, w \in U$, we can exploit efficient approximation algorithms to solve certain class of diversification objectives (see Section 3).

Our work is similar in spirit to earlier work on axiomatization of ranking and clustering systems [3, 11]. We study the functions that arise out of the requirement of satisfying a set of simple properties and show an *impossibility result* which states that there exists no diversification function f that satisfies all the properties. We state the properties in Section 2.

Although we do not aim to completely map the space of objective functions in this study, we show that some diversification objectives reduce to different versions of the well-studied *facility dispersion* problem. Specifically, we pick three functions that satisfy different subsets of the properties and characterize the solutions obtained by well-known approximation algorithms for each of these functions. We also characterize some of the objective functions defined in earlier works [1, 16, 6] using the axioms.

1.1 Related Work

The early work of [4] described the trade-off between relevance and novelty via the choice of a parameterized objective function. Subsequent work on *query abandonment* by [6] is based on the idea that documents should be selected sequentially according to the probability of the document being relevant *conditioned* on the documents that come before. [16] solved a similar problem by using bypass rates of a document to measure the overall likelihood of a user bypassing all documents in a given set. Thus, the objective in their setting was to produce a set that minimized likelihood of completely getting bypassed.

[1] propose a diversification objective that tries to maximize the likelihood of finding a relevant document in the top-k positions given the categorical information of the queries and documents. Other works on topical diversification include [18, 21]. [20, 19] propose a risk minimization framework for information retrieval that allows a user to define an arbitrary *loss* function over the set of returned documents. [17] proposed a method for diversifying query results in online shopping applications wherein the query is presented in a structure form using online forms.

Our work is based on axiomatizations of ranking and clustering systems [2, 11, 3]. [11] proposed a set of three natural axioms for clustering functions and showed that no clustering function satisfies all three axioms. [3] study ranking functions that combine individual votes of agents into a social ranking of the agents and compare them to social choice welfare functions which were first proposed in the classical work on social choice theory by [2].

[1] See [15] and references therein.

We also show a mapping from diversification functions to those used in facility dispersion [13, 12]. The interested reader will find a useful literature in the chapter on facility dispersion in [14, 5].

2 Axiomatic Framework

This section introduces the axiomatic framework and fixes the notation to be used in the remainder of the paper. We are given a set $U = \{u_1, u_2, \ldots, u_n\}$ of $n \geq 2$ of documents, and a finite set of queries Q. Now, given a query $q \in Q$ and an integer k, we want to output a subset $S_k \subseteq U$ (where $|S_k| = k$) of documents that is simultaneously both relevant and diverse.[2] The relevance of each document is specified by a function $w : U \times Q \rightarrow \mathbf{R}^+$, where a higher value implies that the document is more relevant to a particular query. The diversification objective is intuitively thought of as giving preference to dissimilar documents. To formalize this, we define a distance function $d : U \times U \rightarrow \mathbf{R}^+$ between the documents, where smaller the distance, the more similar the two documents are. We also require the distance function to be discriminative, i.e. for any two documents $u, v \in U$, we have $d(u, v) = 0$ if and only if $u = v$, and symmetric, i.e $d(u, v) = d(v, u)$. Note that the distance function need not be a metric.

Formally, the set selection function $f : 2^U \times Q \times w \times d \rightarrow \mathbf{R}$ can be thought of as assigning scores to all possible subsets of U, given a query $q \in Q$, a weight function $w(\cdot)$, a distance function $d(\cdot, \cdot)$. Fixing q, $w(\cdot)$, $d(\cdot, \cdot)$ and a given integer $k \in \mathbf{Z}^+$ ($k \geq 2$), the objective is to select a set $S_k \subseteq U$ of documents such that the value of the function f is maximized, i.e. the objective is to find

$$S_k^* = \underset{S_k \subseteq U,\, |S_k| = k}{\operatorname{argmax}} f(S_k, q, w(\cdot), d(\cdot, \cdot)) \tag{1}$$

where all arguments other than the set S_k are fixed inputs to the function.

An important observation is that the diversification framework is under-specified and even if one assumes that the relevance and distance functions are provided, there are many possible choices of the diversification objective function f. These functions could trade-off relevance and similarity in different ways, and one needs to specify criteria for selection among these functions. A natural mathematical approach in such a situation is to provide axioms that any diversification system should be expected to satisfy and therefore provide *some* basis of comparison between different diversification functions.

2.1 Axioms of diversification

We propose that f is such that it satisfy the set of axioms given below, each of which is a property that is intuitive for the purpose of diversification. In addition, we show that any proper subset of these axioms is *maximal*, i.e. no diversification function can satisfy all these axioms. This provides a natural method of selecting between various objective functions, as one can choose the essential properties for any particular diversification system. In section 3, we will illustrate the use of the axioms in choosing between different diversification objectives. Before we state the axioms, we state the following notation. Fix any q, $w(\cdot)$, $d(\cdot, \cdot)$, k and f, such that f is maximized by S_k^*, i.e., $S_k^* = \operatorname{argmax}_{S_k \subseteq U} f(S_k, q, w(\cdot), d(\cdot, \cdot))$.

1. **Scale invariance**: Informally, this property states that the set selection function should be insensitive to the scaling of the input functions. Consider the optimal set S_k^*. Now, we require f to be such that we have $S_k^* = \operatorname{argmax}_{S_k \subseteq U} f(S_k, q, \alpha \cdot w(\cdot), \alpha \cdot d(\cdot, \cdot))$ for any fixed positive constant $\alpha \in \mathbf{R}$, $\alpha > 0$, i.e. S_k^* still maximizes f even if all relevance and distance values are scaled by some constant. Note that we require

[2] In this work, we focus our attention on the *set selection* problem instead of producing a *ranked list* as the output, which is the ultimate goal in some settings such as web search. This choice is motivated by the fact that the set selection problem captures the core trade-off involved in building a diversification system as we will see shortly. Furthermore, there are several ways to convert the selected set into a ranked list. For instance, one can always rank the results in order of relevance.

the constant to be the same for both $w(\cdot)$ and $d(\cdot,\cdot)$ in order to be consistent with respect to the scale of the problem.

2. **Consistency**: Consistency states that making the output documents more relevant and more diverse, and making other documents less relevant and less diverse should not change the output of the ranking. Now, given any two functions $\alpha : U \to \mathbf{R}^+$ and $\beta : U \times U \to \mathbf{R}^+$, we modify the relevance and weight functions as follows:

$$w(u) = \begin{cases} w(u) + \alpha(u) & , \ u \in S_k^* \\ w(u) - \alpha(u) & , \ \text{otherwise} \end{cases}$$

$$d(u,v) = \begin{cases} d(u,v) + \beta(u,v) & , \ u,v \in S_k^* \\ d(u,v) - \beta(u,v) & , \ \text{otherwise} \end{cases}$$

The ranking function f must be such that it is still maximized by S_k^*. We emphasize that the change in the relevance and diversity for each document can be different, and consistency only requires the function f to be invariant with respect to the right *direction* of the change.

3. **Richness**: Informally speaking, the richness condition states that we should be able to achieve any possible set as the output, given the right choice of relevance and distance function. This property is motivated by the practical fact that one could construct a query (along with corresponding relevance and diversity functions) that would correspond to any given output set. Formally, there exists some $w(\cdot)$ and $d(\cdot,\cdot)$ such that for any $k \geq 2$, there is a unique S_k^* which maximizes f.

4. **Stability**: The stability condition seeks to ensure that the output set does not change arbitrarily with the output size, i.e., the function f should be defined such that $S_k^* \subset S_{k+1}^*$. One can also consider relaxations of the strict stability condition such as $|S_k^* \cap S_{k+1}^*| > 0$. We do not discuss the relaxation further in this work.

5. **Independence of Irrelevant Attributes**: This axiom states that the score of a set is not affected by most attributes of documents outside the set. Specifically, given a set S, we require the function f to be such that $f(S)$ is independent of values of both $w(u)$ and $d(u,v)$ for all $u, v \notin S$.

6. **Monotonicity**: Monotonicity simply states that the addition of any document does not decrease the score of the set. Fix any $w(\cdot)$, $d(\cdot,\cdot)$, f and $S \subseteq U$. Now, for any $x \notin S$, we must have $f(S \cup \{x\}) \geq f(S)$.

Finally, we state two properties that ensure that the diversification system will not trivially ignore one of either relevance or diversity objectives. This ensures that the system will capture the trade-off between the two objectives in a non-degenerate manner.

7. **Strength of Relevance**: This property ensures that no function f ignores the relevance function. Formally, we fix some $w(\cdot)$, $d(\cdot,\cdot)$, f and S. Now, the following properties should hold for any $x \in S$:

   (a) There exist some real numbers $\delta_0 > 0$ and $a_0 > 0$, such that the condition stated below is satisfied after the following modification: obtain a new relevance function $w'(\cdot)$ from $w(\cdot)$, where $w'(\cdot)$ is identical to $w(\cdot)$ except that $w'(x) = a_0 > w(x)$. The remaining relevance and distance values could decrease arbitrarily. Now, we must have

$$f(S, w'(\cdot), d(\cdot,\cdot), k) = f(S, w(\cdot), d(\cdot,\cdot), k) + \delta_0$$

(b) If $f(S \setminus \{x\}) < f(S)$, then there exist some real numbers $\delta_1 > 0$ and $a_1 > 0$ such that the following condition holds: modify the relevance function $w(\cdot)$ to get a new relevance function $w'(\cdot)$ which is identical to $w(\cdot)$ except that $w'(x) = a_1 < w(x)$. Now, we must have

$$f(S, w'(\cdot), d(\cdot,\cdot), k) = f(S, w(\cdot), d(\cdot,\cdot), k) - \delta_1$$

8. **Strength of Similarity**: This property ensures that no function f ignores the similarity function. Formally, we fix some $w(\cdot)$, $d(\cdot,\cdot)$, f and S. Now, the following properties should hold for any $x \in S$:

   (a) There exist some real numbers $\delta_0 > 0$ and $b_0 > 0$, such that the condition stated below is satisfied after the following modification: obtain a new distance function $d'(\cdot,\cdot)$ from $d(\cdot,\cdot)$, where we increase $d(x,u)$ for the required $u \in S$ to ensure that $\min_{u \in S} d(x,u) = b_0$. The remaining relevance and distance values could decrease arbitrarily. Now, we must have

   $$f(S, w(\cdot), d'(\cdot,\cdot), k) = f(S, w(\cdot), d(\cdot,\cdot), k) + \delta_0$$

   (b) If $f(S \setminus \{x\}) < f(S)$, then there exist some real numbers $\delta_1 > 0$ and $b_1 > 0$ such that the following condition holds: modify the distance function $d(\cdot,\cdot)$ by decreasing $d(x,u)$ to ensure that $\max_{u \in S} d(x,u) = b_1$. Call this modified distance function $d'(\cdot,\cdot)$. Now, we must have

   $$f(S, w(\cdot), d'(\cdot,\cdot), k) = f(S, w(\cdot), d(\cdot,\cdot), k) - \delta_1$$

Given these axioms, a natural question is to characterize the set of functions f that satisfy these axioms. A somewhat surprising observation here is that it is impossible to satisfy all of these axioms simultaneously (proof is in the full paper [8]):

Theorem 1: No function f satisfies all 8 axioms stated above.

This result allows us to naturally characterize the set of diversification functions, and selection of a particular function reduces to deciding upon the subset of axioms (or properties) that the function is desired to satisfy. The next section explore this idea further and shows that the axiomatic framework could be a powerful tool in choosing between diversification function. Another advantage of the framework is that it allows a theoretical characterization of the function which is independent of the specifics of the diversification system such as the distance and the relevance function.

3 Objectives and Algorithms

In light of the impossibility result shown in Theorem 1, we can only hope for diversification functions that satisfy a subset of the axioms. We note that the list of such functions is possibly quite large, and indeed several such functions have been previously explored in the literature (see [6, 16, 1], for instance). Further, proposing a diversification objective may not be useful in itself unless one can actually find algorithms to optimize the objective. In this section, we aim to address both of the above issues: we demonstrate the power of the axiomatic framework in choosing objectives, and also propose reductions from a number of natural diversification objectives to the well-studied combinatorial optimization problem of facility dispersion [14]. In particular, we propose two diversification objectives in the following sections, and provide algorithms that optimize these objectives. We also present a brief characterization of the objective functions studied in earlier works [1, 16, 6]. We will use the same notation as in the previous section and have the objective (namely equation 1), where f would vary from one function to another. Also, we assume $w(\cdot)$, $d(\cdot,\cdot)$ and k to be fixed here and hence use the shorthand $f(S)$ for the function.

3.1 Max-sum diversification

A natural bi-criteria objective is to maximize the sum of the relevance and dissimilarity of the selected set. This objective can be encoded in terms of our formulation in terms of the function $f(S)$, which is defined as follows:

$$f(S) = (k-1)\sum_{u \in S} w(u) + 2\lambda \sum_{u,v \in S} d(u,v) \tag{2}$$

where $|S| = k$, and $\lambda > 0$ is a parameter specifying the trade-off between relevance and similarity. Observe that we need to scale up the first sum to balance out the fact that there are $\frac{k(k-1)}{2}$ numbers in the similarity sum, as opposed to k numbers in the relevance sum. We first characterize the objective in terms of the axioms.

Remark 1: The objective function given in equation 2 satisfies all the axioms, except stability.

This objective can be recast in terms of a facility dispersion objective, known as the MAXSUMDISPERSION problem. The MAXSUMDISPERSION problem is a facility dispersion problem having the objective maximizing the sum of all pairwise distances between points in the set S which we show to be equivalent to equation 2. To this end, we define a new distance function $d'(u,v)$ as follows:

$$d'(u,v) = w(u) + w(v) + 2\lambda d(u,v) \tag{3}$$

It is not hard to see the following claim (proof skipped):

Claim 2: $d'(\cdot,\cdot)$ is a metric if the distance $d(\cdot,\cdot)$ constitutes a metric.

Further, note that for some $S \subseteq U$ ($|S| = k$), we have:

$$\sum_{u,v \in S} d'(u,v) = (k-1)\sum_{u \in S} w(u) + 2\lambda \sum_{u,v \in S} d(u,v)$$

using the definition of $d'(u,v)$ and the fact that each $w(u)$ is counted exactly $k-1$ times in the sum (as we consider the complete graph on S). Hence, from equation 2 we have that

$$f(S) = \sum_{u,v \in S} d'(u,v)$$

But this is also the objective of the MAXSUMDISPERSION problem described above where the distance metric is given by $d'(\cdot,\cdot)$.

Given this reduction, we can map known results about MAXSUMDISPERSION to the diversification objective. First of all, we observe that maximizing the objective in equation 2 is NP-hard, but there are known approximation algorithms for the problem. In particular, there is a 2-approximation algorithm for the MAXSUMDISPERSION problem [10, 9] (for the metric case) and is given in algorithm 1. Hence, we can use algorithm 1 for the max-sum objective stated in 2.

3.2 Mono-objective formulation

The space of diversification objectives is quite rich, and indeed there are several examples of objectives that do not reduce to facility dispersion. For instance, the second objective we will explore does not relate to facility dispersion as it combines the relevance and the similarity values into a single value for each *document* (as opposed to each edge for the previous two objectives). The objective can be stated in the notation of our framework in terms of the function $f(S)$, which is defined as follows:

$$f(S) = \sum_{u \in S} w'(u) \tag{4}$$

Algorithm 1 Algorithm for MAXSUMDISPERSION

Require: Universe U, k
Ensure: Set S ($|S| = k$) that maximizes $f(S)$
1: Initialize the set $S = \emptyset$
2: **for** $i \leftarrow 1$ **to** $\lfloor \frac{k}{2} \rfloor$ **do**
3: Find $(u, v) = \text{argmax}_{x,y \in U}\, d(x, y)$
4: Set $S = S \cup \{u, v\}$
5: Delete all edges from E that are incident to u or v
6: **end for**
7: **if** k is odd **then**
8: add an arbitrary document to S
9: **end if**

where the new relevance value $w'(\cdot)$ for each document $u \in U$ is computed as follows:

$$w'(u) = w(u) + \frac{\lambda}{|U| - 1} \sum_{v \in U} d(u, v)$$

for some parameter $\lambda > 0$ specifying the trade-off between relevance and similarity. Intuitively, the value $w'(u)$ computes the "global" importance (i.e. not with respect to any particular set S) of each document u. The axiomatic characterization of this objective is as follows:

Remark 2: The objective in equation 4 satisfies all the axioms except consistency.

Also observe that it is possible to exactly optimize objective 4 by computing the value $w'(u)$ for all $u \in U$ and then picking the documents with the top k values of u for the set S of size k.

3.3 Other objective functions

We note that the link to the facility dispersion problem explored in section 3.1 is particularly rich as many dispersion objectives have been studied in the literature (see [14, 5]). Although we only explored a single objective here in order to illustrate the use of the dispersion objective, similar reductions can be used to obtain algorithms with provable guarantees for many other diversification objectives [8].

The axiomatic framework can also be used to characterize diversification objectives that have been proposed previously. For instance, we note that the DIVERSIFYobjective function in [1] as well as the MINQUERY-ABANDONMENT formulations proposed in [16] violate the stability and the independence of irrelevant attributes axioms.

4 Conclusions

This work presents an approach to characterizing diversification systems using a set of natural axioms. The choice of axioms presents an objective basis for characterizing diversification objectives independent of the algorithms used, and the specific forms of the distance and relevance functions. Specifically, we illustrate the use of the axiomatic framework by studying two objectives satisfying different subsets of axioms.

References

[1] R. Agrawal, S. Gollapudi, A. Halverson, S. Ieong, *Diversifying Search Results*, in Proc. 2nd ACM Intl Conf on Web Search and Data Mining, 5–14, 2009.

[2] K. Arrow, *Social Choice and Individual Values*, Wiley, New York, 1951.

[3] A. Altman, M. Tennenholtz, *On the axiomatic foundations of ranking systems*, in Proc. 19th Int'l Joint Conference on Artificial Intelligence, 917–922, 2005.

[4] J. Carbonell, J. Goldstein, *The use of MMR, diversity-based reranking for reordering documents and producing summaries*, in Proc. of the Int'l Conf. on Research and Development in Information Retrieval (SIGIR), 335–336, 1998.

[5] B. Chandra, M. M. Halldórsson, *Approximation Algorithms for Dispersion Problems*, in J. Algorithms, (38):2, 438–465, 2001.

[6] H. Chen, D. R. Karger, *Less is more: probabilistic models for retrieving fewer relevant documents*, in Proc. of the Int'l Conf. on Research and Development in Information Retrieval (SIGIR), 429–436, 2006.

[7] C. L. A. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, I. MacKinnon, *Novelty and diversity in information retrieval evaluation*, in Proc. of the Int'l Conf. on Research and Development in Information Retrieval (SIGIR), 659–666, 2008.

[8] S. Gollapudi, A. Sharma, *An Axiomatic Approach for Result Diversification*, in Proc. of the Int'l Conf. on World Wide Web, 381–390, 2009.

[9] R. Hassin, S. Rubinstein, A. Tamir, *Approximation algorithms for maximum dispersion*, in Operations Research Letters, (21):3, 133–137, 1997.

[10] B. Korte, D. Hausmann, *An Analysis of the Greedy Heuristic for Independence Systems*, in Algorithmic Aspects of Combinatorics, (2), 65–74, 1978.

[11] J. Kleinberg, *An Impossibility Theorem for Clustering*, in Advances in Neural Information Processing Systems 15: Proc. of the 2002 Conference, MIT Press, 2003.

[12] S. S. Ravi, D. J. Rosenkrantz, G. K. Tayi, *Facility dispersion problems: Heuristics and special cases*, in Proc. 2nd Workshop on Algorithms and Data Structures (WADS), 355–366, 1991.

[13] S. S. Ravi, D. J. Rosenkrantz, G. K. Tayi, *Heuristic and special case algorithms for dispersion problems*, in Operations Research, (42):2, 299–310, 1994.

[14] S. S. Ravi, D. J. Rosenkrantzt, G. K. Tayi, *Approximation Algorithms for Facility Dispersion*, in Handbook of Approximation Algorithms and Metaheuristics, Chapman & Hall/CRC, 2007.

[15] S. Robertson, H. Zaragoza, *On rank-based effectiveness measures and optimization*, in Inf. Retr. (10):3, 321–339, 2007.

[16] A. D. Sarma, S. Gollapudi, S. Ieong, *Bypass rates: reducing query abandonment using negative inferences*, in Proc. of the ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining, 177–185, 2008.

[17] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, S. A. Yahia, *Efficient Computation of Diverse Query Results*, in IEEE 24th Int'l Conference on Data Engineering (ICDE), 228–236, 2008.

[18] C. X. Zhai, W. W. Cohen, J. Lafferty, *Beyond independent relevance: methods and evaluation metrics for subtopic retrieval*, in Proc. of the Int'l Conf. on Research and Development in Information Retrieval (SIGIR), 10–17, 2003.

[19] C. X. Zhai, *Risk Minimization and Language Modeling in Information Retrieval*, in Carnegie Mellon University, 2002.

[20] C. X. Zhai and J. Lafferty, *A risk minimization framework for information retrieval*, in Information Processing and Management, (42):1, 31–55, 2006.

[21] C. N. Ziegler, S. M. McNee, J. A. Konstan, G. Lausen, *Improving recommendation lists through topic diversification*, in Proc. of the Int'l Conf. on World Wide Web, 22–32, 2005.

The KNDN Problem: A Quest for Unity in Diversity

Jayant R. Haritsa
Indian Institute of Science, Bangalore
haritsa@dsl.serc.iisc.ernet.in

Abstract

Given a query location Q in multi-dimensional space, the classical KNN problem is to return the K spatially closest answers in the database with respect to Q. The KNDN (K-Nearest Diverse Neighbor) problem is a semantic extension where the objective is to return the spatially closest result set such that each answer is sufficiently different, or diverse, from the rest of the answers. We review here the KNDN problem formulation, the associated technical challenges, and candidate solution strategies.

1 Introduction

Over the last decade, the issue of diversity in query results produced by information retrieval systems, such as search engines, has been investigated in great depth (see [11] for a recent survey). Curiously, however, there has been relatively little diversity-related work with regard to queries issued on traditional *database systems*. Even more surprisingly, this paucity has occurred in spite of extensive research, during the same period, on supporting vanilla distance-based Top-K (or equivalently, KNN) queries in a host of database environments including online business databases, multimedia databases, mobile databases, stream databases, etc. (see [2, 6, 7] for detailed surveys). The expectation that this prior work would lead on to a vigorous investigation of result-set *semantics*, including aspects such as diversity, has unfortunately not fully materialized. In this article, we review our 2004 study of database result diversity [9], in the hope that it may rekindle interest in the area. Specifically, we cover the KNDN (K-Nearest Diverse Neighbor) problem formulation, the associated technical challenges, and candidate solution strategies.

Motivation. The general model of a KNN query is that there is a database of multidimensional objects against which users submit point queries. Given a metric for measuring distances in the multidimensional space, the system is expected to find the K objects in the database that are spatially closest to the location of the user's query. Typical distance metrics include the Euclidean and Manhattan norms. An example KNN application scenario is given below.

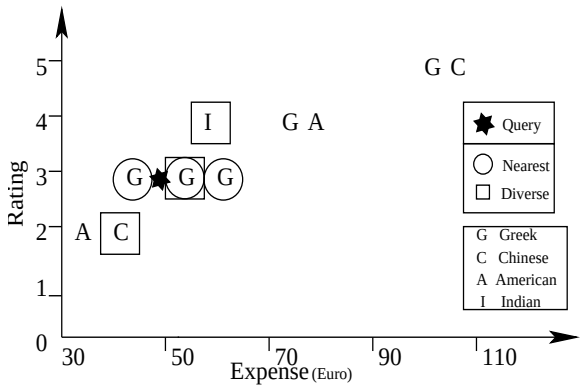
Example 1: Consider the situation where the London tourist office maintains the relation RESTAURANT (Name,Speciality,Rating,Expense), where Name is the name of the restaurant; Speciality indicates the food type (Greek, Chinese, Indian, etc.); Rating is an integer between 1 to 5 indicating restaurant quality; and, Expense is the typical expected expense per person. In this scenario, a visitor to London may wish to submit

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

the KNN query shown in Figure 1(a) to have a choice of three nearby mid-range restaurants where dinner is available for around 50 Euros. (The query is written using the SQL-like notation described in [2].)

```
SELECT * FROM RESTAURANT
WHERE Rating=3 and Expense=50
ORDER 3 BY Euclidean
```

(a) KNN Query



(b) Data Distribution

Figure 1: Motivational Example

For the sample distribution of data points in the RESTAURANT database shown in Figure 1(b), the above KNN query would return the answers shown by the circles. Note that these three answers are very similar: {*Parthenon-α,3,Greek,54*}, {*Parthenon-β,3,Greek,45*}, and {*Parthenon-γ,3,Greek,60*}. In fact, it is even possible that the database may contain exact *duplicates* (wrt the Speciality, Rating and Expense attributes).

Returning three very similar answers may not offer much value to the London tourist. Instead, she might be better served by being told, in addition to {*Parthenon-α,3,Greek,54*}, about a Chinese restaurant {*Hunan,2,Chinese,40*}, and an Indian restaurant {*Taj,4,Indian,60*}, which would provide a close but more heterogeneous set of choices to plan her dinner – these answers are shown by rectangles in Figure 1(b). In short, the user would like to have not just the closest set of answers, but the closest *diverse* set of answers (an oft-repeated quote from Montaigne, the sixteenth century French writer, is *"The most universal quality is diversity"* [15]).

2 The KNDN Problem

Based on the above motivation, we introduced five years ago the **KNDN** (K-Nearest Diverse Neighbor) problem in [9]. The objective here is to identify the spatially closest set of answers to the user's query location such that each answer is sufficiently diverse from all the others in the result set.

We began by modeling the database as composed of N tuples over a D-dimensional space with each tuple representing a point in this space. The user specifies a point query Q over an M-sized subset of these attributes $S(Q) : (s_1, s_2, \ldots, s_M)$, referred to as "spatial attributes"; and lists a L-sized subset of attributes $V(Q) : (v_1, v_2, \ldots, v_L)$, on which she would like to have result diversity – these attributes are referred to as "diversity attributes". The space formed by the diversity attributes is referred to as *diversity-space*. Note that the choice of the diversity attributes is *orthogonal* to the choice of the spatial attributes. Finally, the user also specifies K, the number of desired answers, and the expected result is a set of K diverse tuples from the database. Referring back to the London tourist example, the associated settings in this framework are $D = 4, M = 2, s_1 = Rating, s_2 = Expense, L = 1, v_1 = Speciality, K = 3$.

For ease of exposition, we will hereafter assume that the domains of all attributes are numeric and normalized to the range $[0, 1]$, and that all diversity dimensions are equivalent in the user's perspective. The relaxation of these assumptions is discussed in [8].

2.1 Diversity Measure

The next, and perhaps most important, question we faced was how to define diversity and quantify it as a measure. Here, there were a variety of options, such as hashing-based schemes [1], or taxonomy-based schemes [10], but all these seem primarily suited for the largely unstructured domain of IR applications. In the structured database world, however, we felt that diversity should have a direct *physical association* with the data tuples for it to be meaningful to users. Accordingly, we chose our definition of diversity based on the classical *Gower coefficient* [3]. Here, the difference between two points is defined as a weighted average of the respective attribute value differences. Specifically, we first compute the (absolute) differences between the attribute values of these two points in *diversity-space*, sequence these differences in *decreasing* order of their values, and then label them as $(\delta_1, \delta_2, \ldots, \delta_L)$. Next, we calculate $divdist$, the diversity distance between points P_1 and P_2 with respect to diversity attributes $V(Q)$ as

$$divdist(P_1, P_2, V(Q)) = \sum_{j=1}^{L} (W_j \times \delta_j) \tag{5}$$

where the W_j's are weighting factors for the differences. Since all δ_j's are in the range $[0, 1]$ (recall that the values on all dimensions are normalized to $[0, 1]$), and by virtue of the W_j assignment policy discussed below, diversity distances are also bounded in the range $[0, 1]$.

The assignment of the weights is based on the principle that *larger weights* should be assigned to the *larger differences*. That is, in Equation 5, we need to ensure that $W_i \geq W_j$ if $i < j$. The rationale for this assignment is as follows: Consider the case where point P_1 has values $(0.2, 0.2, 0.3)$, point P_2 has values $(0.19, 0.19, 0.29)$ and point P_3 has values $(0.2, 0.2, 0.27)$. Consider the diversity of P_1 with respect to P_2 and P_3. While the aggregate difference is the same in both cases, yet intuitively we can see that the pair (P_1, P_2) is more homogeneous as compared to the pair (P_1, P_3). This is because P_1 and P_3 differ considerably on the third attribute as compared to the corresponding differences between P_1 and P_2. In a nutshell, a pair of points that have higher variance in their attribute differences are taken to be more diverse than a pair with lower variance.

Now consider the case where P_3 has values $(0.2, 0.2, 0.28)$. Here, although the aggregate difference is higher for the pair (P_1, P_2), yet again it is pair (P_1, P_3) that appears more diverse since its difference on the third attribute is larger than any of the individual differences in pair (P_1, P_2).

Based on the above observations, we decided that the weighting function should have the following properties: First, all weights should be positive, since having differences in any dimension should never decrease the diversity. Second, the sum of the weights should add up to 1 (i.e., $\sum_{j=1}^{L} W_j = 1$) to ensure that $divdist$ values are normalized to the $[0, 1]$ range. Third, the weights should be *monotonically decaying* ($W_i \geq W_j$ if $i < j$) to reflect the preference given to larger differences. Finally, the weighting function should ideally be tunable using a single parameter.

Example 2: A candidate weighting function that materially satisfies the above requirements is the following:

$$W_j = \frac{a^{j-1} \times (1-a)}{1-a^L} \quad (1 \leq j \leq L) \tag{6}$$

where a is a tunable parameter over the range $(0, 1)$. Note that this function implements a *geometric* decay, with the parameter 'a' determining the rate of decay. Values of a that are close to 0 result in faster decay, whereas values close to 1 result in slow decay. When the value of a is nearly 0, almost all weight is given to the maximum difference i.e., $W_1 \simeq 1$, modeling the L_∞ (i.e., Max) distance metric, and when a is nearly 1, all attributes are given similar weights, modeling a L_1 (i.e., Manhattan) distance metric.

Figure 2 shows, for different values of the parameter 'a' in Equation 6, the locus of points which have a diversity distance of 0.1 with respect to the origin $(0, 0)$ in two-dimensional diversity-space.

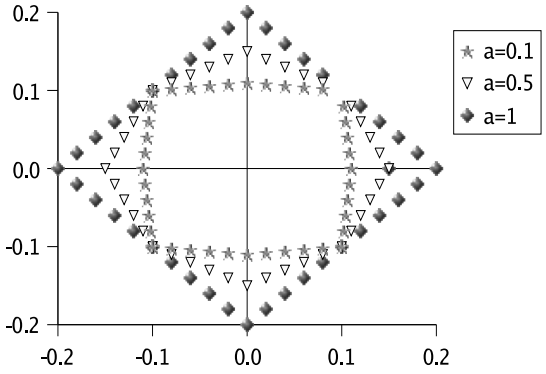


Figure 2: Locus of Diversity=0.1 Points

```
SELECT * FROM RESTAURANT
WHERE Rating=3 and Expense=50
ORDER 3 BY Euclidean
WITH MinDiv=0.1 ON Speciality
```

Figure 3: KNDN Query Example

2.1.1 Minimum Diversity Threshold

The next issue we faced was deciding the mechanism by which the user specified the extent to which diversity was required in the result set. While complicated functions could be constructed for this purpose, we felt that typical users of database systems would be only willing, or knowledgeable enough, to provide a single value characterizing their requirements. Accordingly, we supported a single threshold parameter *MinDiv*, ranging between $[0, 1]$. Given this threshold setting, two points are diverse if the diversity distance between them is greater than or equal to *MinDiv*. That is,

$$DIV(P_1, P_2, V(Q)) = true \;\; iff \;\; divdist(P_1, P_2, V(Q)) \geq MinDiv \tag{7}$$

The advantage of the above simple definition is that it is amenable to a *physical* interpretation that can guide the user in determining the appropriate setting of *MinDiv*. Specifically, the interpretation is that if a pair of points is deemed to be diverse, then these two points have a difference of *MinDiv* or more on atleast one diversity dimension. For example, a *MinDiv* of 0.1 means that any pair of diverse points differ in atleast one diversity dimension by atleast 10% of the associated domain size. In practice, we would expect that *MinDiv* settings would be on the low side, typically not more than 0.2. Finally, note that the DIV function is commutative but not transitive.

The notion of pair-wise diversity is easily extended to the final user requirement that each point in the result set must be diverse with respect to *all* other points in the set. That is, given a result set $\mathcal{A}$ with answer points $A_1, A_2, \ldots, A_K$, we require $DIV(A_i, A_j, V(Q)) =$ true $\forall\, i, j$ such that $i \neq j$ and $1 \leq i, j \leq K$. We call such a result set to be *fully-diverse*.

With the above framework, setting *MinDiv* to zero results in the traditional KNN query, whereas higher values provide more and more importance to diversity at the expense of distance. The London tourist query can now be stated as shown in Figure 3, where *Speciality* is the attribute on which diversity is calculated, and the goal is to produce the spatially closest fully-diverse result set.

2.1.2 Integrating Diversity and Distance

Given a user query location and a diversity requirement, there may be *multiple* fully-diverse answer sets available in the database. Ideally, the set that should be returned is the one that has the closest spatial characteristics. To identify this set, we need to first define a single value that characterizes the aggregate spatial distances of a set of points, and this is done in the following manner: Let function $SpatialDist(P, Q)$ calculate the spatial distance of point P from query point Q. The choice of *SpatialDist* function is based on the user specification and could

be any monotonically increasing distance function such as Euclidean, Manhattan, etc. We combine distances of all points in a set into a single value using an aggregate function Agg which captures the overall distance of the set from Q. While a variety of aggregate functions are possible, the choice is constrained by the fact that the aggregate function should ensure that as the individual points in the set move farther away from the query, the distance of the set should also increase correspondingly. Sample aggregate functions which obey this constraint include the Arithmetic, Geometric, and Harmonic Means. Finally, we use the reciprocal of the aggregate to determine the "closeness score" of the (fully-diverse) result set. Putting all these formulations together, given a query Q and a candidate fully-diverse result set $\mathcal{A}$ with points $A_1, A_2, \ldots, A_K$, the closeness score of $\mathcal{A}$ with respect to Q is computed as

$$CloseScore(\mathcal{A}, Q) = \frac{1}{Agg(SpatialDist(Q, A_1), \ldots, SpatialDist(Q, A_K))} \tag{8}$$

2.1.3 Problem Formulation

The final KNDN problem formulation is as follows: *Given a point query Q on a D-dimensional database, a* MinDiv *diversity threshold on a set of diversity dimensions V(Q), and a desired diverse-result cardinality of K, the goal of the K-Nearest Diverse Neighbor (KNDN) problem is to find the set of K mutually diverse tuples in the database, whose CloseScore is the maximum, after including the spatially nearest tuple to Q in the result set.*

The requirement that the nearest point to the user's query should *always* form part of the result set is because this point, in a sense, *best fits* the user's query. Further, the nearest point A_1 serves to seed the result set since the diversity function is meaningful only for a *pair* of points. Since point A_1 of the result is fixed, the result sets are differentiated based on their remaining $(K-1)$ choices.

3 Solution Techniques

Having described the KNDN problem formulation, we now move on to discussing the issues related to developing solutions for this problem.

Not surprisingly, finding the *optimal* (wrt CloseScore) fully-diverse result set for the KNDN problem turns out to be computationally hard. We can establish this by mapping KNDN to the well known *independent set problem* [4] which is NP-complete. The mapping is achieved by forming a graph corresponding to the dataset in the following manner: Each tuple in the dataset forms a node in the graph and a edge is added between two nodes if the diversity between the associated tuples is less than *MinDiv*. Now any independent set of nodes, that is, a subgraph in which no two nodes are connected, of size K in this graph represents a fully-diverse set of K tuples. But finding *any* independent set, let alone the optimal independent set, is itself computationally hard. Tractable solutions to the independent set problem have been proposed [4], but they require the graph to be sparse and all nodes to have a bounded small degree. In our world, this translates to requiring that all the clusters in diversity-space should be small in size. But, this may not be typically true for the datasets encountered in practice, rendering these solutions not generally applicable, and forcing us to go in for heuristic solutions instead.

3.1 Inadequacy of Clustering-based Solutions

At first glance, it may appear that a simple heuristic solution to the KNDN problem would be to initially group the data into clusters using algorithms such as BIRCH [13], replace all clusters by their representatives, and then apply the traditional KNN approach on this summary database. There are two problems here: First, since the clusters are pre-determined, there is no way to dynamically specify the desired diversity, which may vary from one user to another or may be based on the specific application that is invoking the KNDN search. Second, since the query attributes are not known in advance, we potentially need to identify clusters in all possible subspaces,

which may become infeasible due to the exponential number of such subspaces. Finally, this approach cannot provide the traditional KNN results.

Yet another approach to produce a diverse result set could be to run the standard KNN algorithm, cluster its results, replace the clusters by their representatives, and then output these representatives as the diverse set. The problem with this approach is that it is not clear how to apriori determine the original number of required answers such that there are finally K diverse representatives. If the original number is set too low, then the search process has to be restarted, whereas if the original number is set too high, a lot of wasted work ensues.

3.2 The MOTLEY Algorithm

In light of the above, we designed a diversity-conscious greedy algorithm called **MOTLEY**[1] for solving the KNDN problem, which is described in the remainder of this section.

3.2.1 Database Navigation

A major design issue in Motley was the mechanism chosen to navigate through the database. Two primary approaches for database navigation have been proposed in the extensive literaure on the vanilla KNN problem – the first is based on using standard database statistics (e.g., [2]), while the other is based on distance browsing using spatial indices such as the R-tree (e.g., [6, 12]). We opted to use the latter approach since the detailed experimental evaluation of [6] had indicated that distance browsing outperforms the alternative approaches – further, distance browsing can be seamlessly used for both KNN and KNDN problems.

In the distance browsing approach, the database tuples are processed incrementally in increasing order of their distance from the query location. The approach is predicated on having a containment-based index structure, such as the R-tree[5], built collectively on all dimensions of the database (more precisely, the index needs to cover only those attributes on which spatial predicates may appear in the query workload). This assumption appears practical since most current database systems natively support R-trees.

To implement distance browsing, a priority queue, *pqueue*, is maintained which is initialized with the root node of the R-Tree. The *pqueue* maintains the R-Tree nodes and data tuples in increasing order of their distance from the query location. While the distance between a data point and the query Q is computed in the standard manner, the distance between a R-tree node and Q is computed as the minimum of the distances between Q and all points in the region enclosed by the MBR (Minimum Bounding Rectangle) of the R-tree node. The distance of a node from Q is zero if Q is within the MBR of that node, otherwise it is the distance of the closest point on the MBR periphery. For this, we first need to compute the distances between the MBR and Q along each query dimension – if Q is inside the MBR on a specific dimension, the distance is zero, whereas if Q is outside the MBR on this dimension, it is the distance from Q to either the low end or the high end of the MBR, whichever is nearer. Once the distances along all dimensions are available, they are combined (based on the distance metric in operation) to get the effective distance.

To return the next nearest neighbor, we pick up the first element of the *pqueue*. If it is a tuple, it is immediately returned as the sought for neighbor. However, if the element is an R-tree node, all the children of that node are inserted in the *pqueue*. During this insertion process, the *spatial* distance of the object from the query point is calculated and used as the insertion key. This process is repeated until we find a tuple to be the first element of the queue, which is then returned.

The above distance browsing process continues until either the diverse result set is found, or until all points in the database are exhausted, signaled by the *pqueue* becoming empty.

[1] Motley: A collection containing a variety of sorts of things [14].

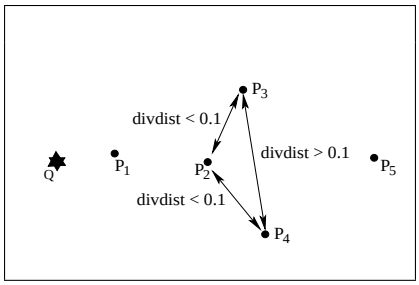


Figure 4: Immediate Greedy

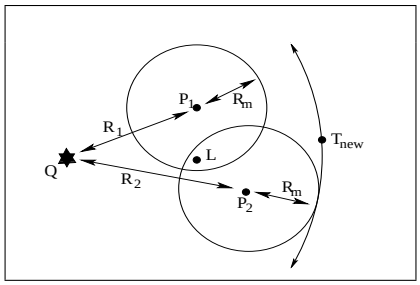


Figure 5: Buffered Greedy

3.2.2 Immediate Greedy

The first solution we attempted is called ImmediateGreedy (IG), wherein distance browsing is used to simply access database tuples in increasing der of their spatial distances from the query point, as discussed above. The first tuple is always inserted into the result set, $\mathcal{A}$, to satisfy the requirement that the closest tuple to the query point must figure in the result set. Subsequently, each new tuple is added to $\mathcal{A}$ if it is diverse with respect to *all* tuples currently in $\mathcal{A}$; otherwise, it is discarded. This process continues until $\mathcal{A}$ grows to contain K tuples.

While the IG approach is straightforward and easy to implement, there are cases where it may make poor choices as shown in Figure 4. Here, Q is the query point, and P_1 through P_5 are the tuples in the database. Let us assume that the goal is to report 3 diverse tuples with *MinDiv* of 0.1. By inspection, we observe that the overall best choice could be $\{P_1,\ P_3,\ P_4\}$. But since $DIV(P_1, P_2, V(Q)) = true$, IG would include P_2 and this would then disqualify the candidatures of P_3 and P_4 as both $DIV(P_2, P_3, V(Q))$ and $DIV(P_2, P_4, V(Q))$ are false. Eventually, Immediate Greedy would give the solution as $\{P_1,\ P_2,\ P_5\}$. Worse, if point P_5 happens to be not present in the database, then this approach will fail to return a fully-diverse set even though such a set, namely $\{P_1,\ P_3,\ P_4\}$, is available.

3.2.3 Buffered Greedy

The above problems are addressed in the BufferedGreedy (BG) method by recognizing that IG gets into difficulty because, at all times, only the diverse points (or "leaders") in the result set, retained. To counter this, BG maintains with each leader a bounded buffered set of "dedicated followers" – a dedicated follower is a point that is not diverse with respect to a specific leader but is diverse with respect to *all remaining* leaders.

Given this additional set of dedicated followers, we adopt the heuristic that a current leader, L_i, is *replaced* in the result set by its dedicated followers $F_i^1, F_i^2, \ldots, F_i^j (j > 1)$ as leaders if (a) these dedicated followers are *all* mutually diverse, and (b) incorporation of these followers as leaders does not result in the premature disqualification of future leaders. The first condition is necessary to ensure that the result set contains only diverse points, while the second is necessary to ensure that we do not produce solutions that are worse than Immediate Greedy. For example, if in Figure 4, point P_5 had happened to be only a little farther than point P_4 such that $DIV(P_2, P_5, V(Q)) = true$, then the replacement of P_2 by P_3 and P_4 could be the wrong choice since $\{P_1, P_2, P_5\}$ may turn out to be the best solution.

To implement the second condition, we must know when it is "safe" to go ahead with a replacement i.e., when it is certain that all future leaders will be diverse from the current set of followers. To achieve this, we do the following: For each point, we consider a hypothetical sphere that contains all points in the domain space that may be non-diverse with respect to it. That is, we set the radius R_m of the sphere equal to the distance of the farthest non-diverse point in the domain space. Note that this sphere may contain some diverse points as well, but our aim is to take a conservative approach. Now, the replacement of a leader by selected dedicated followers can be done as soon as we have reached a distance greater than R_m with respect to the farthest follower from the

query – this is because all future leaders will be diverse with respect to selected dedicated followers and there is no possibility of disqualification beyond this point. To clarify this technique, consider the following example:

Example 3: In Figure 5, the circles around P_1 and P_2 show the areas that contain all points that are not diverse with respect to P_1 and P_2, respectively. During the distance browsing process, when we access the point T_{new} (Figure 5), we know that all future points will be diverse from P_1 and P_2. At this time, if P_1 and P_2 are dedicated followers of L and mutually diverse, then we can replace L by $\{P_1,\ P_2\}$.

Our experimental results (detailed in [9, 8]) on synthetic and real data-sets, indicate that BufferedGreedy can provide high-quality diverse solutions at a low cost in terms of both result distance and processing time.

4 Future Directions

In this article, we reviewed the KNDN problem of finding the closest set of answers such that the user will find each answer sufficiently different from the rest, thereby adding value to the result set. This topic has been largely ignored by the database community, but appears to be a fecund source of interesting and challenging problems. For example, handling diversity in high-dimensional data, and efficient maintenance and usage of spatial index structures for arbitrary subsets of attributes, are promising immediate probems to be addressed. In the long term, modeling the KNDN operations within the query optimizer and integrating the solution with the query processing engine would be of practical import.

References

[1] A. Broder, M. Charikar, A. Frieze and M. Mitzenmacher, *Min-wise independent permutations*, Journal of Computer and System Sciences, 60(3), 2000.

[2] N. Bruno, S. Chaudhuri and L. Gravano, *Top-K Selection Queries over Relational Databases: Mapping Strategies and Performance Evaluation*, ACM Trans. on Database Systems, 27(2), 2002.

[3] J. Gower, *A general coefficient of similarity and some of its properties*, Biometrics 27, 1971.

[4] M. Grohe, *Parameterized Complexity for Database Theorists*, SIGMOD Record 31(4), December 2002.

[5] A. Guttman, *R-trees: A dynamic index structure for spatial searching*, Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1984.

[6] G. Hjaltason and H. Samet, *Distance Browsing in Spatial Databases*, ACM Trans. on Database Systems, 24(2), 1999.

[7] I. Ilyas, G. Beskales and M. Soliman, *A Survey of Top-k Query Processing Techniques in Relational Database Systems*, ACM Computing Surveys, 40(4), October 2008.

[8] A. Jain, P. Sarda and J. Haritsa, *Providing Diversity in K-Nearest Neighbor Query Results*, Tech. Report TR-2003-04, DSL/SERC, Indian Institute of Science, 2003.

[9] A. Jain, P. Sarda, and J. Haritsa, *Providing Diversity in K-Nearest Neighbor Query Results*, Proc. of 8th Pacific Asia Conf. on Knowledge Discovery and Data Mining, 2004.

[10] S. Gollapudi and A. Sharma, *An Axiomatic Approach for Result Diversification*, Proc. of World Wide Web Conf., 2009.

[11] E. Minack, G. Demartini and W. Nejdl, *Current Approaches to Search Result Diversification*, Proc. of 1st Intl. Workshop on Living Web, 2009.

[12] N. Roussopoulos, S. Kelley and F.Vincent, *Nearest Neighbor Queries*, Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1995.

[13] T. Zhang, R. Ramakrishnan and M. Livny, *BIRCH: An Efficient Data Clustering Method for Very Large Databases*, Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1996.

[14] `www.thefreedictionarity.com`

[15] `www.elibronquotations.com`

Making Product Recommendations More Diverse

Cai-Nicolas Ziegler* Georg Lausen

Institut für Informatik, Universität Freiburg
Georges-Köhler-Allee, Gebäude 51
79110 Freiburg i. Br., Germany

{cziegler,lausen}@informatik.uni-freiburg.de

Abstract

Past research in recommender systems has mainly focused on improving accuracy, i.e., making each single recommendation get as close to the user's information need as possible. However, while this approach works well when focusing on single recommendations as atomic entities, its usefulness to the consumer appears limited when considering entire recommendation lists along with their overall utility, which often appear to provide rather an unbalanced diet to the user: Recommendation lists seldom reflect the consumer's entire spectrum of interest but rather hook on to small portions that appear particularly favorable with regard to accuracy optimization. We analyze the diversification issue in detail and present a framework that is geared towards making lists as interesting and colorful as possible, trading a minimum of accuracy in exchange for the gain in diversity. Empirical evaluations aiming for actual user satisfaction underpin the cogency of our approach.

1 Introduction

Recommender systems [10] intend to provide people with recommendations of products they will appreciate, based on their past preferences. Many of the most successful systems make use of collaborative filtering [6], and numerous commercial systems, e.g., Amazon.com's recommender [9], exploit these techniques to offer personalized recommendation lists to their customers. Though the *accuracy* of state-of-the-art collaborative filtering systems, i.e., the probability that the active user[1] will appreciate the products recommended, is excellent, some implications affecting user satisfaction have been observed in practice. Thus, on Amazon.com (*http://www.amazon.com*), many recommendations seem to be "similar" with respect to content. For instance, customers that have purchased many of Hermann Hesse's prose may happen to obtain recommendation lists where all top-5 entries contain books by that respective author only. When considering pure accuracy, all these recommendations appear excellent since the active user clearly appreciates books written by Hermann Hesse.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Now working at The Boston Consulting Group, Munich. Contact at ziegler.cai-nicolas@bcg.com.

[1]The term "active user" refers to the person for whom recommendations are made.

On the other hand, assuming that the active user has several interests other than Hermann Hesse, e.g., historical novels, the recommended set of items appears poor, owing to its lack of diversity.

Traditionally, recommender system projects have focused on optimizing accuracy. Now research has reached the point where going beyond pure accuracy and toward real user experience becomes indispensable for further advances. This work looks specifically at impacts of recommendation *lists*, regarding them as entities in their own right rather than mere aggregations of single and independent suggestions.

2 On Collaborative Filtering

Collaborative filtering (CF) still represents the most commonly adopted technique in recommender systems. Its basic idea is to make recommendations based upon ratings that users have assigned to products. Ratings can either be explicit, i.e., by having the user state his opinion about a given product, or implicit, when the mere act of purchasing an item counts as an expression of appreciation. While implicit ratings are more facile to collect, their usage implies adding noise to the collected information.

User-based Collaborative Filtering has been explored in-depth during the last 10-15 years and is the most popular recommendation algorithm [6], owing to its compelling simplicity and excellent quality of recommendations. CF operates on a set of users $A = \{a_1, a_2, \ldots, a_n\}$, a set of products $B = \{b_1, b_2, \ldots, b_m\}$, and partial rating functions $r_i : B \to [-1, +1]^{\perp}$ for each user a_i. Negative values $r_i(b_k)$ denote utter dislike, while positive values express a_i's liking of product b_k. If ratings are implicit only, we represent them by set $R_i \subseteq B$, equivalent to $\{b_k \in B \mid r_i(b_k) \neq \perp\}$. The user-based CF's working process can be broken down as follows:

- **Neighborhood formation.** Assuming a_i as the active user, similarity values $c(a_i, a_j) \in [-1, +1]$ for all $a_j \in A \setminus \{a_i\}$ are computed, based upon the similarity of their respective rating functions r_i, r_j. In general, Pearson correlation [12, 4] or cosine distance [6] are used for computing $c(a_i, a_j)$. The top-M most similar users a_j become members of a_i's neighborhood, $clique(a_i) \subseteq A$.
- **Rating prediction.** Taking all the products b_k that a_i's neighbors $a_j \in clique(a_i)$ have rated and which are new to a_i, i.e., $r_i(b_k) = \perp$, a prediction of liking $w_i(b_k)$ is produced. Value $w_i(b_k)$ hereby depends on both the similarity $c(a_i, a_j)$ of a_j with $r_j(b_k) \neq \perp$, as well as the ratings $r_j(b_k)$ these a_j assigned to b_k.

Eventually, a list $P_{w_i} : \{1, 2, \ldots, N\} \to B$ of top-N recommendations is computed, based on predictions w_i. P_{w_i} is injective and reflects recommendation ranking in *descending* order, giving highest predictions first.

Item-based CF has favorable computational complexity characteristics and allows to decouple the model computation process from actual prediction making. Specifically for cases where $|A| \gg |B|$, item-based CF's computational performance has been shown superior to user-based CF [11]. Its success also extends to many commercial recommender systems, such as Amazon.com's [9].

As with user-based CF, recommendation making is based upon ratings $r_i(b_k)$ that users $a_i \in A$ provided for products $b_k \in B$. However, unlike user-based CF, similarity values c are computed for *items* rather than *users*, hence $c : B \times B \to [-1, +1]$. Roughly speaking, two items b_k, b_e are similar, i.e., have large $c(b_k, b_e)$, if users who rate one of them tend to rate the other, and if users tend to assign them identical or similar ratings. Moreover, for each b_k, its neighborhood $clique(b_k) \subseteq B$ of top-M most similar items is defined.

3 Evaluation Metrics

Evaluation metrics are essential in order to judge the performance of recommender systems, even though they are still in their infancies. Most evaluations concentrate on accuracy measurements only and neglect other fac-

tors, e.g., novelty and serendipity of recommendations, and the diversity of the recommended list's items. The following sections give a short overview of popular metrics. An extensive survey of accuracy metrics is in [7].

3.1 Accuracy Metrics

Accuracy metrics have been proposed for two major tasks: First, to judge the **accuracy of single predictions**, i.e., how much predictions $w_i(b_k)$ for products b_k deviate from a_i's actual ratings $r_i(b_k)$. These metrics are particularly suited for tasks where predictions are displayed along with the product, e.g., annotation in context [7]. Examples include the mean absolute error (MAE), and mean squared error (MSE).

Second, **decision-support metrics** evaluate the effectiveness of helping users to select high-quality items from the set of all products, generally supposing binary preferences. Typical decision-support metrics include the well-known precision and recall metrics, known from information retrieval, as well as ROC, the "receiver operating characteristic" (see, e.g., [5]).

3.2 Beyond Accuracy

Though accuracy metrics are an important facet of usefulness, there are traits of user satisfaction they are unable to capture. However, non-accuracy metrics have largely been denied major research interest so far. Among all non-accuracy evaluation metrics, **coverage** has been the most frequently used [6, 5]. Coverage measures the percentage of elements part of the problem domain for which predictions can be made.

Some recommenders produce highly accurate results that are still useless in practice, e.g., suggesting bananas to customers in grocery stores. Though being highly accurate, note that almost everybody likes and buys bananas. Hence, their recommending appears far too obvious and of little help to the shopper. **Novelty** and **serendipity** metrics thus measure the "non-obviousness" of recommendations made, avoiding "cherry-picking" [7]. For some simple measure of serendipity, take the average popularity of recommended items.

3.3 Intra-List Similarity

We present a new metric that intends to capture the diversity of a list. Hereby, diversity may refer to all kinds of features, e.g., genre, author. Based upon an arbitrary function $c_\circ : B \times B \rightarrow [-1, +1]$ measuring the similarity $c_\circ(b_k, b_e)$ between products b_k, b_e according to some custom-defined criterion, we define intra-list similarity for a_i's list P_{w_i} as follows:

$$ILS(P_{w_i}) = \frac{\sum_{b_k \in \Im P_{w_i}} \sum_{b_e \in \Im P_{w_i}, b_k \neq b_e} c_\circ(b_k, b_e)}{2} \tag{9}$$

Symbol $\Im P_{w_i}$ denotes the *image* of map P_{w_i}, i.e., all items part of the recommendation list. Higher scores of $ILS(P_{w_i})$ denote lower diversity. An interesting mathematical feature of $ILS(P_{w_i})$ we are referring to in later sections is permutation-insensitivity, i.e., let S_N be the symmetric group of all permutations on $N = |P_{w_i}|$ symbols, then $\forall \sigma_i, \sigma_j \in S_N : ILS(P_{w_i} \circ \sigma_i) = ILS(P_{w_i} \circ \sigma_j)$ holds. Hence, simply rearranging positions of recommendations in a top-N list P_{w_i} does not affect P_{w_i}'s intra-list similarity.

4 Topic Diversification

One major issue with accuracy metrics is their inability to capture the broader aspects of satisfaction, hiding blatant flaws in existing systems. For instance, suggesting a list of very similar items, e.g., with respect to the author, genre, or topic, may be of little use for the user, even though this list's average accuracy might be high. The issue has been perceived by other researchers before, coined "portfolio effect" by Ali and van Stam [1]. We believe that item-based CF systems in particular are susceptible to that effect.

procedure diversify (P_{w_i}, Θ_F) {
  $B_i \leftarrow \Im P_{w_i}$; $P_{w_i*}(1) \leftarrow P_{w_i}(1)$;
  for $z \leftarrow 2$ **to** N **do**
    set $B'_i \leftarrow B_i \setminus \{P_{w_i*}(k) \mid k \in [1, z[\,\}$;
    $\forall b \in B'$: compute $c^*(\{b\}, \{P_{w_i*}(k) \mid k \in [1, z[\,\})$;
    compute $P_{c^*} : \{1, 2, \ldots, |B'_i|\} \rightarrow B'_i$ using c^*;
    for all $b \in B'_i$ **do**
      $P_{c^*}^{\text{rev}^{-1}}(b) \leftarrow |B'_i| - P_{c^*}^{-1}(b)$;
      $w_i^*(b) \leftarrow P_{w_i}^{-1}(b) \cdot (1 - \Theta_F) + P_{c^*}^{\text{rev}^{-1}}(b) \cdot \Theta_F$;
    end do
    $P_{w_i*}(z) \leftarrow min\{w_i^*(b) \mid b \in B'_i\}$;
  end do
  return P_{w_i*};
}

Algorithm 1: Sequential topic diversification

We propose an approach we call *topic diversification* to deal with the problem at hand and make recommended lists more diverse and thus more useful. Our method represents an extension to existing recommender algorithms and is applied on top of recommendation lists.

4.1 Taxonomy-based Similarity Metric

Function $c^* : 2^B \times 2^B \rightarrow [-1, +1]$, quantifying the similarity between two product sets, forms an essential part of topic diversification. We instantiate c^* with our metric for taxonomy-driven filtering [13], though other content-based similarity measures may appear likewise suitable. Our metric computes the similarity between product sets based upon their classification. Each product belongs to one or more classes that are hierarchically arranged in classification taxonomies, describing the products in machine-readable ways.

Classification taxonomies exist for various domains. Amazon.com crafts very large taxonomies for books, DVDs, CDs, electronic goods, and apparel. Moreover, all products on Amazon.com bear content descriptions relating to these domain taxonomies. Featured topics could include author, genre, and audience.

4.2 Topic Diversification Algorithm

Algorithm 1 shows the complete topic diversification algorithm, a brief textual sketch is given in the next paragraphs.

Function P_{w_i*} denotes the new recommendation list, resulting from applying topic diversification. For every list entry $z \in [2, N]$, we collect those b from the candidate products set B_i that do not occur in positions $o < z$ in P_{w_i*} and compute their similarity with $\{P_{w_i*}(k) \mid k \in [1, z[\,\}$ containing all new recommendations preceding rank z.

Sorting all products b according to $c^*(b)$ in reverse order, we obtain the *dissimilarity rank* $P_{c^*}^{\text{rev}}$. This rank is then merged with the original recommendation rank P_{w_i} according to diversification factor Θ_F, yielding final rank P_{w_i*}. Factor Θ_F defines the *impact* that dissimilarity rank $P_{c^*}^{\text{rev}}$ exerts on the eventual overall output.

Large $\Theta_F \in [0.5, 1]$ favors diversification over a_i's original relevance order, while low $\Theta_F \in [0, 0.5[$ produces recommendation lists closer to the original rank P_{w_i}. For experimental analysis, we used factors $\Theta_F \in [0, 0.9]$.

Note that ordered input lists P_{w_i} must be considerably larger than the final top-N list. For our later experiments, we used top-50 input lists for eventual top-10 recommendations.

The effect of dissimilarity bears traits similar to that of **osmotic pressure** and selective permeability known from molecular biology; this concept allows cells (i.e, recommendation lists) to maintain their internal composition of substances (i.e., topics) at required levels (i.e., gauging accuracy versus dissimilarity) [14].

5 Empirical Analysis

We conducted offline evaluations to understand the ramifications of topic diversification on accuracy metrics, and online analysis to investigate how our method affects actual user satisfaction. We applied topic diversification with $\Theta_F \in \{0, 0.1, 0.2, \ldots 0.9\}$ to lists generated by both user-based CF and item-based CF, observing effects that occur when steadily increasing Θ_F and analyzing how both approaches respond to diversification.

We based online and offline analyses on data gathered from BookCrossing (*http://www.bookcrossing.com*). The latter community caters for book lovers exchanging books all around the world and sharing their experiences with others. We collected data on $278,858$ members of BookCrossing and $1,157,112$ ratings, both implicit and explicit, referring to $271,379$ distinct ISBNs. Invalid ISBNs were excluded from the outset. The complete BookCrossing dataset, featuring fully anonymized information, is available via the first author's homepage (*http://www.informatik.uni-freiburg.de/∼cziegler*).

Next, we mined Amazon.com's book taxonomy, comprising 13,525 distinct topics. In order to be able to apply topic diversification, we mined content information, focusing on taxonomic descriptions that relate books to taxonomy nodes from Amazon.com. Since many books on BookCrossing refer to rare, non-English books, or outdated titles not in print anymore, we were able to garner background knowledge for only $175,721$ books. In total, $466,573$ topic descriptors were found, giving an average of 2.66 topics per book.

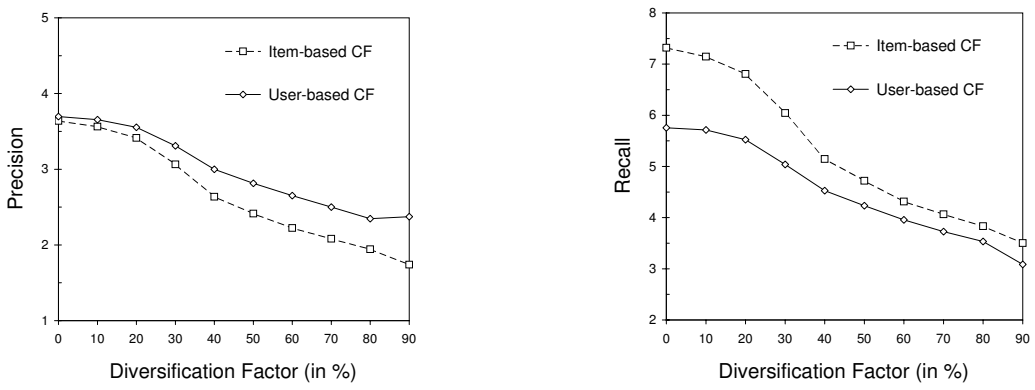
Owing to the BookCrossing dataset's extreme sparsity, we decided to further condense the set in order to obtain more meaningful results from CF algorithms when computing recommendations. Hence, we discarded all books missing taxonomic descriptions, along with all ratings referring to them. Next, we also removed book titles with fewer than 20 overall mentions. Only community members with at least 5 ratings each were kept.

The resulting dataset's dimensions were considerably more moderate, featuring $10,339$ users, $6,708$ books, and $361,349$ book ratings.

5.1 Offline Experiments

We performed offline experiments comparing precision, recall, and intra-list similarity scores for 20 different list setups. Half these recommendation lists were based upon user-based CF with different degrees of diversification, the others on item-based CF. Note that we did not compute MAE metric values since we are dealing with implicit rather than explicit ratings.

For cross-validation of precision and recall metrics of all $10,339$ users, we adopted K-folding with parameter $K = 4$. Hence, rating profiles R_i were effectively split into training sets R_i^x and test sets $T_i^x, x \in \{1, \ldots, 4\}$, at a ratio of $3:1$. For each of the $41,356$ different training sets, we computed 20 top-10 recommendation lists. To generate the diversified lists, we computed top-50 lists based upon pure, i.e., non-diversified, item-based CF and pure user-based CF. Next, we applied the diversification algorithm to both base cases, applying Θ_F factors ranging from 10% up to 90%. For evaluation, all lists were truncated to contain 10 books only.



(a) (b)

Figure 1: Precision (a) and recall (b) for increasing Θ_F

5.1.1 Result Analysis

We were interested in seeing how accuracy, captured by precision and recall, behaves when increasing Θ_F from 0.1 up to 0.9. Since topic diversification may make books with high predicted accuracy trickle down the list, we hypothesized that accuracy will *deteriorate* for $\Theta_F \rightarrow 0.9$. Moreover, in order to find out if our novel algorithm has any significant, positive effects on the diversity of items featured, we also applied our intra-list similarity metric. An overlap analysis for diversified lists, $\Theta_F \geq 0.1$, versus their respective non-diversified pendants indicates how many items stayed the same for increasing diversification factors.

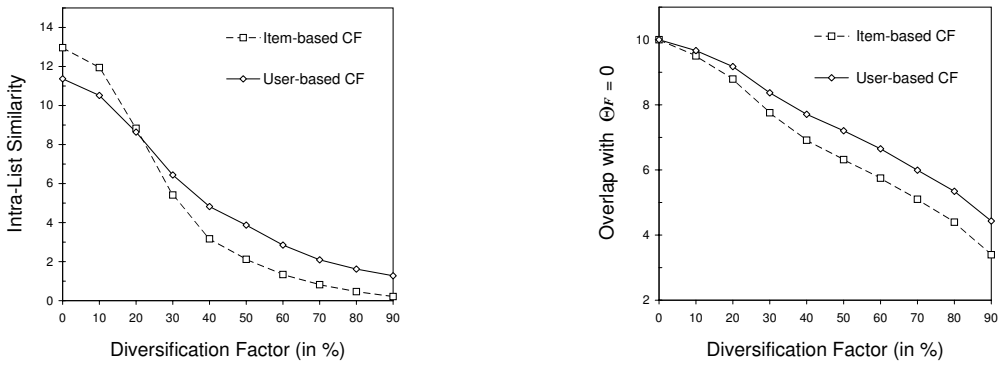
Precision and Recall. First, we analyzed precision and recall scores for both non-diversified base cases, i.e., when $\Theta_F = 0$. For item-based CF we had a precision of 3.64, and recall of 7.32. For user-based CF, the results were 3.69 and 5.76, respectively. Thus, user-based and item-based CF exhibit almost identical accuracy, indicated by precision values. Their recall values differ considerably, hinting at deviating behavior with respect to the types of users they are scoring for.

Next, we analyzed the behavior of user-based and item-based CF when steadily increasing Θ_F by increments of 10%, depicted in Figure 1. The two charts reveal that diversification has detrimental effects on both metrics and on both CF algorithms. Interestingly, corresponding precision and recall curves have almost identical shape.

The loss in accuracy is more pronounced for item-based than for user-based CF. Furthermore, for either metric and either CF algorithm, the drop is most distinctive for $\Theta_F \in [0.2, 0.4]$. For lower Θ_F, negative impacts on accuracy are marginal. We believe this last observation due to the fact that precision and recall are permutation-insensitive, i.e., the mere order of recommendations within a top-N list does not influence the metric value, as opposed to Breese score [2, 7]. However, for low Θ_F, the pressure that the dissimilarity rank exerts on the top-N list's makeup is still too weak to make many new items diffuse into the top-N list. Hence, we conjecture that rather the *positions* of current top-N items change, which does not affect either precision or recall.

Intra-List Similarity. Knowing that our diversification method exerts a significant, negative impact on accuracy metrics, we wanted to know how our approach affected the intra-list similarity measure. Similar to the precision and recall experiments, we computed values for user-based and item-based CF with $\Theta_F \in [0, 0.9]$ each. Results obtained from intra-list similarity analysis are given in Figure 2(*a*).

The topic diversification method lowers the pairwise similarity between list items, thus making top-N recommendation lists more diverse. Diversification appears to affect item-based CF stronger than its user-based counterpart, in line with our findings about precision and recall. For lower Θ_F, curves are less steep than



(a) (b)

Figure 2: Intra-list similarity behavior (a) and overlap with original list (b) for increasing Θ_F

for $\Theta_F \in [0.2, 0.4]$, which also well aligns with precision and recall analysis. Again, the latter phenomenon can be explained by one of the metric's inherent features, i.e., like precision and recall, intra-list similarity is permutation-insensitive.

Figure 2(*b*) shows the number of recommended items staying the same when increasing Θ_F with respect to the original list's content. Both curves exhibit roughly linear shapes, being less steep for low Θ_F, though. Interestingly, for factors $\Theta_F \leq 0.4$, at most 3 recommendations change on average.

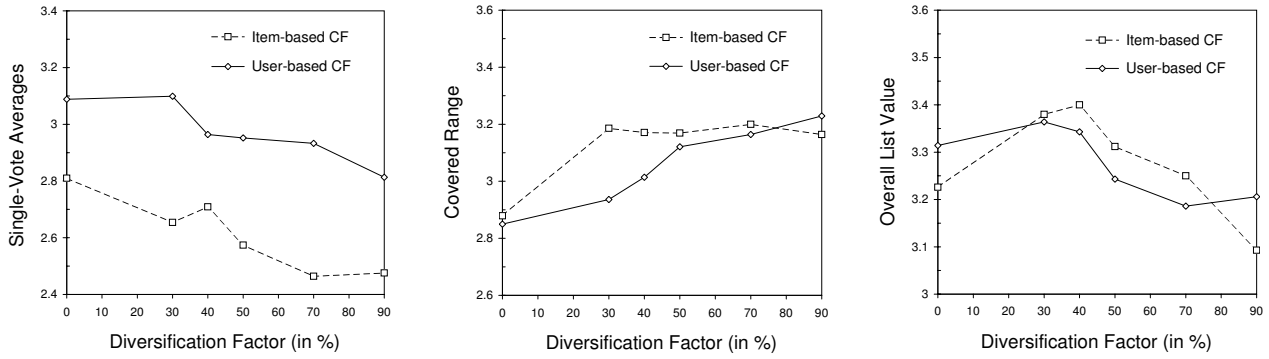
5.2 Conclusion

We found that diversification appears detrimental to both user-based and item-based CF along precision and recall metrics. In fact, this outcome aligns with our expectations, considering the nature of those two accuracy metrics and the way that the topic diversification method works. Moreover, we found that item-based CF seems more susceptible to topic diversification than user-based CF, backed by results from precision, recall and intra-list similarity metric analysis.

5.3 Online Experiments

Offline experiments helped us in understanding the implications of topic diversification on both CF algorithms. We could also observe that the effects of our approach are different on different algorithms. However, we wanted to assess actual user satisfaction for various degrees of diversification, thus necessitating an online survey. For the online study, we computed each recommendation list type anew for users in the denser BookCrossing dataset, though without K-folding. We mailed all eligible users via the community mailing system, asking them to participate in our online study. Each mail contained a personal link to our online survey pages. In order to make sure that only the users themselves would complete their survey, links contained unique, encrypted access codes. During the 3-week survey phase, $2,125$ users participated and completed the study.

The survey consisted of several screens that would tell the prospective participant about this study's nature and his task, show all his ratings used for making recommendations, and finally present a top-10 recommendation list, asking several questions thereafter. For each book, users could state their interest on a 5-point rating scale. Scales ranged from "not much" to "very much", mapped to values 1 to 4, and offered the user to indicate that he had already read the book, mapped to value 5. In order to successfully complete the study, users were not required to rate all their top-10 recommendations. Neutral values were assumed for non-votes instead. However, we required users to answer all further questions, concerning the list as a whole rather than its single recommendations, before submitting their results.



(a) (b) (c)

Figure 3: Results for single-vote averages (a), covered range of interests (b), and overall satisfaction (c)

The one top-10 list for each user was chosen among 12 candidate lists, either user-based CF or item-based with $\Theta_F \in \{0, 0.3, 0.4, 0.5, 0.7, 0.9\}$ each. The assignment of a specific list to the current user was done dynamically, at the time of the participant entering the survey, and in a round-robin fashion. Thus, we could guarantee that the number of users per list type was roughly identical.

5.3.1 Result Analysis

For the analysis of our inter-subject survey, we were mostly interested in the following three aspects: First, the average rating users gave to their 10 single recommendations. We expected results to roughly align with scores obtained from precision and recall. Second, we wanted to know if users perceived their list as well-diversified, asking them to tell whether the lists reflected rather a broad or narrow range of their interests. Referring to the intra-list similarity metric, we expected users' perceived range of topics to increase with increasing Θ_F. Third, we were curious about the overall satisfaction of users with their lists, the measure to compare performance.

Both latter-mentioned questions were answered by each user on a 5-point likert scale (higher scores denoting better performance) and we averaged the eventual results by the number of users. Statistical significance of all mean values was measured by parametric one-factor ANOVA, where $p < 0.05$ if not indicated otherwise.

Single-Vote Averages. Users perceived recommendations made by user-based CF systems on average as more accurate than those made by item-based CF systems, see Fig. 3(*a*). At each featured diversification level Θ_F, differences between the two CF types are significant, $p \ll 0.01$. Moreover, for each algorithm, higher diversification factors obviously entail lower single-vote average scores, which confirms our hypothesis stated before. The item-based CF's cusp at $\Theta_F \in [0.3, 0.5]$ appears as a notable outlier, but differences between the 3 means at $\Theta_F \in [0.3, 0.5]$ are not statistically significant, $p > 0.15$. Contrarily, differences between all factors Θ_F are significant for item-based CF, $p \ll 0.01$, and for user-based CF, $p < 0.1$.

Hence, topic diversification *negatively* correlates with pure accuracy. Besides, users perceived the performance of user-based CF as significantly better than item-based CF for all corresponding levels Θ_F.

Covered Range. Next, we analyzed if users actually *perceived* the variety-augmenting effects caused by topic diversification, illustrated before through the measurement of intra-list similarity. Users' reactions to steadily incrementing Θ_F are illustrated in Figure 3(*b*). First, between both algorithms on corresponding Θ_F levels, only the difference of means at $\Theta_F = 0.3$ shows statistical significance. Studying the trend of user-based CF for increasing Θ_F, we notice that the perceived range of reading interests covered by users' recommendation lists also increases. Hereby, the curve's first derivative maintains an approximately constant level, exhibiting slight peaks between $\Theta_F \in [0.4, 0.5]$. Statistical significance holds for user-based CF between means at $\Theta_F = 0$ and

$\Theta_F > 0.5$, and between $\Theta_F = 0.3$ and $\Theta_F = 0.9$. On the contrary, the item-based curve exhibits a drastically different behavior. While soaring at $\Theta_F = 0.3$ to 3.186, reaching a score almost identical to the user-based CF's peak at $\Theta_F = 0.9$, the curve barely rises for $\Theta_F \in [0.4, 0.9]$, remaining rather stable and showing a slight, though insignificant, upward trend. Statistical significance was shown for $\Theta_F = 0$ with respect to all other samples taken from $\Theta_F \in [0.3, 0.9]$. Hence, our online results do not perfectly align with findings obtained from offline analysis. While the intra-list similarity chart in Figure 2 indicates that diversity increases when increasing Θ_F, the item-based CF chart defies this trend, first soaring then flattening.

Overall List Value. The third feature variable, the overall value users assigned to their personal list, effectively represents the *target value* of our studies, measuring actual satisfaction. Owing to our conjecture that user satisfaction is a mere composite of accuracy and other factors, such as the list's diversity, we hypothesized that the application of topic diversification would *increase* satisfaction. At the same time, considering the downward trend of precision and recall for increasing Θ_F, in accordance with declining single-vote averages, we expected user satisfaction to drop off for large Θ_F. Hence, we supposed an arc-shaped curve for both algorithms.

Results for overall list value are given in Figure 3(c). For user-based CF we observe that the curve does *not* follow our hypothesis. Slightly improving at $\Theta_F = 0.3$ over the non-diversified case, scores drop for $\Theta_F \in [0.4, 0.7]$, culminating in a slight upturn at $\Theta_F = 0.9$. While lacking reasonable explanations, the curve's data-points de facto bear no statistical significance for $p < 0.1$. Hence, we conclude that topic diversification has a marginal, largely negligible impact on overall user satisfaction, initial positive effects eventually being offset by declining accuracy. On the contrary, for item-based CF, results look different. In compliance with our previous hypothesis, the curve's shape follows an arc, peaking at $\Theta_F = 0.4$. Taking the three data-points defining the arc, we obtain significance for $p < 0.1$. Since the endpoint's score at $\Theta_F = 0.9$ is inferior to the non-diversified case's, we find that too much diversification appears detrimental.

Eventually, for overall list value analysis, we come to conclude that topic diversification has no measurable effects on user-based CF, but significantly improves item-based CF performance for diversification factors Θ_F around 40%. In order to verify whether diversification appears as important ingredient of satisfaction, we also conducted multiple linear regresiion trials. Owing to space limitations, these trials are not reported here but can be accessed by resorting to [14].

6 Related Work

Few efforts have addressed the problem of diversifying top-N lists. Only considering literature on collaborative filtering and recommender systems in general, none have been presented before, to our best knowledge.

However, some work related to our topic diversification approach can be found in information retrieval, specifically meta-search engines. A critical aspect of meta-search engine design is the merging of several top-N lists into one single top-N list. Intuitively, this merged top-N list should reflect the highest quality ranking possible, also known as the "rank aggregation problem" [3].

More related to our idea of creating lists that represent the whole plethora of the user's topic interests, Kummamuru *et al.* [8] present their clustering scheme that groups search results into clusters of related topics. The user can then conveniently browse topic folders relevant to his search interest. The commercially available search engine NORTHERN LIGHT (*http://www.northernlight.com*) incorporates similar functionalities.

7 Conclusion

We presented topic diversification, an algorithmic framework to increase the diversity of a top-N list of recommended products. We also introduced our new intra-list similarity metric.

Contrasting precision and recall metrics for user-based and item-based CF with results obtained from a large-scale user survey, we showed that the user's overall liking of recommendation lists goes beyond accuracy and involves other factors, e.g., the users' perceived list diversity. We thus could demonstrate that lists are more than mere aggregations of single recommendations, but bear an intrinsic, added value.

Though effects of diversification were largely marginal on user-based CF, item-based CF performance improved significantly, an indication that there are some behavioral differences between both CF classes. Moreover, while pure item-based CF appeared slightly inferior to pure user-based CF in overall satisfaction, diversifying item-based CF with factors $\Theta_F \in [0.3, 0.4]$ made item-based CF outperform user-based CF. Interestingly for $\Theta_F \leq 0.4$, no more than three items tend to change with respect to the original list, shown in Figure 2. Small changes thus have high impact.

We believe our findings especially valuable for practical application scenarios, since many commercial recommender systems, e.g., Amazon.com [9] and TiVo [1], are item-based.

References

[1] ALI, K., AND VAN STAM, W. TiVo: Making show recommendations using a distributed collaborative filtering architecture. In *Proceedings of the 2004 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Seattle, WA, USA, 2004), ACM Press, pp. 394–401.

[2] BREESE, J., HECKERMAN, D., AND KADIE, C. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence* (Madison, WI, USA, July 1998), Morgan Kaufmann, pp. 43–52.

[3] DWORK, C., KUMAR, R., NAOR, M., AND SIVAKUMAR, D. Rank aggregation methods for the Web. In *Proceedings of the Tenth International Conference on World Wide Web* (Hong Kong, China, 2001), ACM Press, pp. 613–622.

[4] GOLDBERG, D., NICHOLS, D., OKI, B., AND TERRY, D. Using collaborative filtering to weave an information tapestry. *Communications of the ACM 35*, 12 (1992), 61–70.

[5] GOOD, N., SCHAFER, B., KONSTAN, J., BORCHERS, A., SARWAR, B., HERLOCKER, J., AND RIEDL, J. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the 16th National Conference on Artificial Intelligence and Innovative Applications of Artificial Intelligence* (Orlando, FL, USA, 1999), American Association for Artificial Intelligence, pp. 439–446.

[6] HERLOCKER, J., KONSTAN, J., BORCHERS, A., AND RIEDL, J. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Berkeley, CA, USA, 1999), ACM Press, pp. 230–237.

[7] HERLOCKER, J., KONSTAN, J., TERVEEN, L., AND RIEDL, J. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems 22*, 1 (2004), 5–53.

[8] KUMMAMURU, K., LOTLIKAR, R., ROY, S., SINGAL, K., AND KRISHNAPURAM, R. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *Proceedings of the 13th International Conference on World Wide Web* (New York, NY, USA, 2004), ACM Press, pp. 658–665.

[9] LINDEN, G., SMITH, B., AND YORK, J. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing 4*, 1 (January 2003).

[10] RESNICK, P., AND VARIAN, H. Recommender systems. *Communications of the ACM 40*, 3 (1997), 56–58.

[11] SARWAR, B., KARYPIS, G., KONSTAN, J., AND RIEDL, J. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the Tenth International World Wide Web Conference* (Hong Kong, China, May 2001).

[12] SHARDANAND, U., AND MAES, P. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems* (Denver, CO, USA, May 1995), ACM Press, pp. 210–217.

[13] ZIEGLER, C.-N., LAUSEN, G., AND SCHMIDT-THIEME, L. Taxonomy-driven computation of product recommendations. In *Proceedings of the 2004 ACM CIKM Conference on Information and Knowledge Management* (Washington, D.C., USA, November 2004), ACM Press, pp. 406–415.

[14] ZIEGLER, C.-N., MCNEE, S., KONSTAN, J., AND LAUSEN, G. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International World Wide Web Conference* (Chiba, Japan, May 2005), ACM Press.

Battling Predictability and Overconcentration in Recommender Systems

Sihem Amer-Yahia[†], Laks V.S. Lakshmanan[‡], Sergei Vassilvitskii[†], Cong Yu[†]
[†] Yahoo! Research, New York, NY, USA
{sihem,sergei,congyu}@yahoo-inc.com
[‡] University of British Columbia, Vancouver, BC, Canada
laks@cs.ubc.ca

1 Introduction

Today's recommendation systems have evolved far beyond the initial approaches from more than a decade ago, and have seen a great deal of commercial success (c.f., Amazon and Netflix). However, they are still far from perfect, and face tremendous challenges in increasing the overall utility of the recommendations. These challenges are present in all stages of the recommendation process. They begin with the *cold start* problem: given a new user how do we recommend items to the user without forcing her to give feedback on a starter set of items. Similarly, given a new item introduced into the system, how do we know when (and to whom) we should recommend it.

A different problem is that of *data sparsity*. Although every system has 'super' raters, that is people who rate thousands (and in some cases tens of thousands) of items, the number of such people is low. In fact, most people give feedback on only a handful of items, resulting in a large, but very sparse dataset. The data also exhibits the long tail phenomenon, with the majority of items getting just a handful of ratings. However, as recent studies have shown, understanding the tail is crucial—while no item in the tail is rated by too many people, the vast majority of people rate at least a few items in the tail [8].

In this work, we focus on a different question facing recommender systems: what determines the utility of a recommendation? In the past, accuracy, usually measured on some held out dataset, has served as the gold standard; indeed this was the only measure used by the famed Netflix prize. However, we must be careful to remember that accuracy is not identical to relevancy or usefulness of a recommendation. Accuracy is the leading component—if the accuracy of recommendations is low the system is not trusted by the user. However, once a recommender system achieves some accuracy bar, obtaining even higher accuracy does not necessarily make a recommendation more relevant. Indeed, recent work (e.g., see Sharma and Carenini [12]) has criticized the use of mean absolute error (MAE) as the sole measure of effectiveness of a recommender system and has proposed alternative measures inspired by decision theory. In this paper, we take a different tack and contend that once an accuracy bar (which may be based on a measure such as MAE) is achieved, *diversity* and *freshness* of recommendations play a crucial role in making a recommendation more relevant and useful to a user.

Diversity One of the problems plaguing recommender systems overly focused on accuracy is *overconcentration*. This is the problem of recommending a set of items, in which all of the items are too similar to each other. For example, the system may learn that a user Alice has a certain penchant for fantasy books and may

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

recommend other books from this specific genre. While Alice surely likes most of these books, she may also enjoy reading detective stories. However, if her tastes in detective stories are more finicky, an accuracy-driven system would never take the risk in recommending an Agatha Christie book that Alice may or may not like over a Lewis Carrol novel that she is sure to enjoy. The result of never taking such risks is a series of recommendations that are very homogeneous. As noted in [3], a series of recommendations that look too similar to each other could easily turn off the user and lower her interest in the site over time.

In this case, the solution is to diversify the set of results, that is to insist that the items returned by the recommendation system are sufficiently different from one another. How do we ensure that the recommended items are different from each other? If intrinsic properties of items, in the form of attributes, are available, we can check that they differ from each other on some attributes. If these are not available or expensive to compute (e.g., images, video, etc.), we must turn to other means. This is a topic we address in this paper and argue that *explanations* of recommendations constitute an excellent piece of information with which to measure item (dis)similarity. It is worth noting that augmenting recommendations with explanations has been recognized as an important step in building user trust and adoption [3] and generating explanations has been recognized as one of the outstanding open problems for recommender systems [9].

Freshness A more subtle problem, not usually addressed is that of *predictability*. Even if the set of results returned by the system is diverse, i.e., the items being recommended are quite different from one another, it may still lack *serendipity*, that is, be already known to the user. Imagine, for example, a system that only recommends best sellers from every genre. While such a system would likely have high accuracy scores (these items are best sellers for a reason), and the set of recommended items is diverse (since recommendations come from many different genres), such a system would rarely be useful. We must be cognizant of the fact that the recommender system is not the sole way by which users learn of new items. In fact, they watch television, they ask their friends for advice, etc. It is generally believed that there is no need to recommend best sellers, precisely because a typical user already knows about these books, however many systems simply draw the line at best sellers without trying to estimate the likelihood that the user has already been exposed to an item through other means.

In order to better estimate this probability, we begin by dividing the item space into a set of *regions*. We think of regions as representing different broad types of items (for example genres of movies), however they need not have any semantic correlation, and can be produced automatically by a separate classifier. We then define the *stickiness* of a user to a region, which describes how often a user rates an item from that region, as compared to a background distribution. Intuitively, if a user (or the user's network) has high stickiness for a region, she is likely to be familiar with the items in that region (since she is likely to have been exposed to them through her network), and hence, is not in need for further recommendations from that region. On the other hand, a user having below average stickiness to a region can be a signal of one of two actions: either the user *likes* these items but is not familiar with them, in which case the system should recommend an item from this region, or the user knows these items, but does not like them. To distinguish between the two cases we rely on the fact that we started with a high accuracy system, and recommend only items with relevance scores above a certain threshold.

Diversity vs. Freshness Notice that the problems related to (lack of) diversity and (lack of) freshness, although closely related, are quite independent. You can have a system making recommendations that are sufficiently dissimilar from each other and yet are predictable for a given user and vice versa. As pointed out above, recommendations consisting of bestsellers, while being diverse, may be utterly predictable for most users. On the other hand, a series of mystery recommendations to Alice in our example above, while fresh in that she may not be sufficiently exposed to mystery books, may end up in recommendations too similar to each other. Unfortunately, in the recommender systems literature, these two problems are conflated and are often referred to using a single term (lack of) diversity. Indeed, one of the approaches suggested for solving this problem is randomization [3]. More precisely, injecting a few random choices into the set of recommendations generated by a system is believed to make the recommendations diverse. With respect to the issues of diversity and freshness (as defined in this work), randomization may yield results that have higher diversity and higher

freshness, although *there is no guarantee.* In this paper, we make a clear distinction between these two problems and summarize some of our recent work that addresses both of them.

2 Recommendation Strategies Overview

In this section, we review the two most basic recommendation approaches: *item-based* and *collaborative filtering* [3]. Despite having been around for a long time, these two approaches are still at the core of the most recent advancements in recommendation systems.

Item Based Strategies Earlier recommendation strategies are mostly item based. They aim to recommend items similar to the ones the end-user preferred in the past. The rating of an item $i \in \mathcal{I}$ (the set of all items) by a current user $u \in \mathcal{U}$ (the set of all users) is estimated as follows: $\texttt{relevance}(u, i) = \Sigma_{i' \in \mathcal{I}} \texttt{ItemSim}(i, i') \times \texttt{rating}(u, i')$.

Here, $\texttt{ItemSim}(i, i')$ returns a measure of similarity between two items i and i', and $\texttt{rating}(u, i')$ indicates the rating of item i' by user u (it is 0 if u has not rated i'). Item based strategies are very effective when the given user has a long history of tagging/rating activities. However, it does have two drawbacks. First, items recommended by item based strategies are often very similar to what the user already knows [10] and therefore other interesting items that the user might like have little chance of being recommended. Second, item based strategy does not work well when a user first joins the system. To partially address those drawbacks, collaborative filtering strategies have been proposed.

Collaborative Filtering Strategies Collaborative filtering (CF) strategies aim to recommend to an end-user items which are highly rated by users who share similar interests with or have declared relationships with her. CF strategies can be classified into two categories: memory-based and model-based. Memory-based CF startegies are similar in spirit to, and can be regarded as dual of, item-based strategies. Here, the rating of an item i by the end-user u is estimated as follows: $\texttt{relevance}(u, i) = \Sigma_{u' \in \mathcal{U}} \texttt{UserSim}(u, u') \times \texttt{rating}(u', i)$.

Here, $\texttt{UserSim}(u, u')$ returns a measure of similarity or connectivity between two users u and u' (it is 0 if u and u' are not connected). Collaborative filtering strategies broaden the scope of items being recommended to the user and have become increasingly popular. Note that in both strategies, we use $relevance(u, i)$ to denote the estimated rating of i by u.

In model-based CF strategies, a model is built, using a machine learning technique such as Bayes Network, clustering, or using association rules, and the model is used for making predictions of the rating of an item by a given user. See [3, 9] for an excellent survey of various recommendation algorithms.

3 Battling Overconcentration through Diversification

The problem of overconcentration can be addressed by *diversification*, where the goal is to ensure that the set of recommended items are dissimilar with each other, but nonetheless relevant to the user's interests. Recommendation diversification is well studied in the recommendation systems community. However, to the best of our knowledge, almost all of the techniques proposed so far for recommendation diversification [10] are *attribute-based*. In other words, the diversity of the recommended list is defined by how much each item in the list differs from the others in terms of their attribute values. For example, in Yahoo! Movies, recommended movies are post-processed based on the set of attributes that are intrinsic to each movie (such as genre, director, etc.). Typically, a heuristic distance formula is defined on the set of attributes in order to allow the system to quantify the distance between a pair of items. In [15], diversity is measured based on a specific attribute of the item which is defined by the taxonomy in which the item is classified. This can also be considered as attribute-based since the classification of each item is intrinsic to that item.

3.1 Drawbacks of Using Attribute-Based Diversification

While suitable for traditional recommender systems where the items are well-described by their attributes (e.g., books in Amazon and movies in Netflix), attribute-based diversification has two main drawbacks.

First, there can be a *lack of attribute descriptions for items*. Particularly, items in social network sites often *lack a comprehensive set of attributes*, unlike their counterparts in traditional recommender systems. For example, URLs in del.icio.us are only described by the set of tags associated with them, so are images in Flickr and videos in YouTube. Tags are attached to the items by individual users and are fundamentally different from intrinsic descriptions of the items. Analyzing intrinsic properties of these items, on the other hand, is extremely difficult or computationally prohibitive. For example, to obtain intrinsic descriptions of each URL in del.icio.us, one is required to crawl and analyze the web page. This is very costly to do for the mere purpose of diversification and certainly beyond the purpose and capability of del.icio.us. For multimedia sites like Flickr and YouTube, while image processing and video analysis techniques do exist, the effort required to extract feature vectors for images and videos may be considered too high a price to pay just for diversifying recommendations.

Second, there can be *substantial computational overhead due to attribute retrieval for the purpose of diversification*. A diversification approach based on attributes often requires accessing item attributes in order to perform the diversification: a step that can become costly when the database is very large as in many of the collaborative tagging sites, especially when the attributes are not leveraged by the recommendation strategies (e.g., many collaborative filtering strategies).

3.2 Explanation-Based Diversification

To overcome the above drawbacks, we describe the novel approach of explanation-based diversification introduced in [14, 13]. To begin with, we denote by $\texttt{RecItems}(u)$, the set of candidate recommended items generated by one of the recommendation strategies described above. The size of this set is typically larger than the final desired number of recommendations.

An explanation for a recommended item depends on the underlying recommendation strategy used. If an item i is recommended to user u by a content-based strategy, then an *explanation* for recommendation i is defined as: $\texttt{Expl}(u,i) = \{i' \in \mathcal{I} \mid \texttt{ItemSim}(i,i') > 0 \ \& \ i' \in \texttt{Items}(u)\}$, i.e., the set of items similar to items (i') that user u has rated in the past. The explanation may contain more information such as the similarity weight $\texttt{ItemSim}(i,i') \times \texttt{rating}(u,i')$. If an item i is recommended to user u by a collaborative filtering strategy, then an *explanation* for a recommendation i is: $\texttt{Expl}(u,i) = \{u' \in \mathcal{U} \mid \texttt{UserSim}(u,u') > 0 \ \& \ i \in \texttt{Items}(u')\}$, i.e., the set of users similar to u who have rated item i. Similarly, we can augment each user u' with the similarity weight $\texttt{UserSim}(u,u') \times \texttt{rating}(u',i)$.

Note that in all cases, the explanation of a recommendation is either a set of items or a set of users. Based on this, we can define diversity of recommendations as follows. Let i, i' be a pair of items (recommended to user u) with associated explanation sets $\texttt{Expl}(u,i)$ and $\texttt{Expl}(u,i')$. Then the *diversity distance* between i, i' for user u can be defined as a similarity measure based on standard metrics such as Jaccard similarity coefficient or cosine similarity. E.g., the *Jaccard diversity distance* between recommendations i and i' is:

$$DD_u^J(i,i') = 1 - \frac{|\texttt{Expl}(u,i) \cap \texttt{Expl}(u,i')|}{|\texttt{Expl}(u,i) \cup \texttt{Expl}(u,i')|}.$$

Note that this is defined as the complement of the standard Jaccard coefficient, since we want to use this as a distance measure. Intuitively, in the case of collaborative filtering, the distance between items depends on the ratio between the number of users who recommend both items and the total number of users who recommend these items.

When weights are incorporated, the *cosine diversity distance* between recommendations i and i' is defined by treating the explanations $\texttt{Expl}(u,i)$ and $\texttt{Expl}(u,i')$ as vectors in a multi-dimensional space and defining $DD_u^C(i,i')$ as 1 minus standard cosine similarity between the vectors. We refer the readers to the Vector Space Model [1] for more details. In the case of collaborative filtering, the weighted diversity distance depends on

the ratio between the number of similar users who recommend both items and the total number of users who recommend these items. When we want to be neutral with respect to the type of diversity distance measure used and opt for the notation $DD_u(i, i')$. Depending on the context, it represents $DD_u^J(i, i')$ or $DD_u^C(i, i')$.

For a set of items $S \subseteq \texttt{RecItems}(u)$, we define: $DD_u(S) = avg\{DD_u(i, i') \mid i, i' \in S\}$, i.e., $DD_u(S)$ is the average diversity distance between pairs of items in the set S. Algorithms can then be designed to leverage this distance measure to strike a good balance between relevance and diversity in the recommendations. In our previous work [13], we describe two efficient heuristic algorithms for achieving this balance: *Algorithm Swap* and *Algorithm Greedy*.

3.3 Brief Description of Diversification Algorithms

The basic idea behind **Algorithm Swap** is to start with the K highest scoring items, and swap the item which contributes the least to the diversity of the entire set with the next highest scoring item among the remaining items. At each iteration, a candidate item with a lower relevance is swapped into the top-k set if and only if it increases the overall diversity of the resulting set. To prevent a sudden drop of the overall relevance of the resulting set, an optional pre-defined upper-bound `UB` (on how much drop in relevance is tolerated) can be used to stop the swapping when the lowest relevance of the remaining items is no longer high enough to justify the swap. The selection of the item to be swapped is accomplished by searching over all items in the current top-K set S and picking item i with the minimum diversity. More precisely, let $D_S^i = \sum_{j \in S, j \neq j} DD_u(i, j)$, we pick the item $i \in S$ with the minimum D_S^i as the candidate for swap.

Another heuristic approach is to start with the most relevant item, greedily add the next most relevant item if and only if that item is far enough away (compared with a distance bound) from existing items in the set, and stop when there are K items. The drawback, however, is that the distance bound is often hard to obtain and can vary from user to user. To address this problem, **Algorithm Greedy** relies on iteratively refining two diversity thresholds: an upper-bound `UB`, initially set to 1, and a lower-bound, `LB`, initially set to 0. The algorithm first iterates through the set of candidate items in the order of their relevances and generates two lists: `DivList`, and `SimList`. The `DivList` contains items that are all maximally distant from each other, while the `SimList` contains items that have zero distance to some items in `DivList`. Because items are accessed in the order of their relevance, `SimiList` essentially contains items that we no longer consider. If `DivList` already contains enough items, we pick K most relevant items from it can return. If not, we adjust `UB` and `LB` and reiterate through the candidate items. At each iteration, the `KeepList` tracks items that will definitely be in the result set while the `DiscardList` tracks items that should be eliminated from consideration. These two lists are merged with `DivList` and `SimList`, respectively, at the end of each iteration. The algorithm stops when K are found in `DivList` or the difference between `UB` and `LB` drops below a threshold.

4 Battling Predictability through Thinking Outside-The-Box

As stated in the introduction, most previous work focuses on increasing diversity to address predictability (see [11] and [15].) For example, DailyLearner [7], a content-based recommender system, filters out highly relevant items which are too similar to items the user has rated in the past. In our work [2], we develop a formal solution to recommendation predictability which relies on identifying *item regions* that guide the system in *taking some risk* to help users make *fresh discoveries*, while *maintaining high relevance*. We refer to our solution as $\mathcal{OTB}$, for Outside-The-Box.

In [2], we experimentally show two important benefits of our approach. First, $\mathcal{OTB}$ recommendations are of high quality as measured by standard metrics. Second, they differ significantly from traditional ones, and this difference is *not* simply in the tail end of the recommendations—we find that they contribute significantly to overall quality. This argues that $\mathcal{OTB}$ recommendations are complementary to conventional ones, and that both

novelty and relevance are necessary to achieve user satisfaction.

4.1 Going Outside the Box

A *region* (i.e., the "box") is a group of similar items. $R^{\mathcal{I}}$ denotes a (potentially overlapping) assignment of items $\mathcal{I}$ into regions. Regions in $R^{\mathcal{I}}$ are produced using similarity distances between items. We explore two such similarities: *attribute-based* and *activity-based*. Let A be a set of item attributes, also called dimensions. Attribute-based similarity relies on a function d_A. For any two items i and j the distance $d_A(i,j) = 0$ iff $\forall a \in A$ we have $i.a. = j.a$, and $d_A(i,j) = \infty$ otherwise.

For movies, region dimensions may include movie attributes like genre and directors. A region instance is often identified by its dimensions and their values (e.g., {(genre=comedy), (producer=Disney)}).

Activity-based similarity produces regions which identify items rated by a large enough number of similar users. More formally, for any two items i and j, and action a let $a(i)$ and $a(j)$ define the respective sets of users that performed action a on the item. Then let $d(i,j)$ be the Jaccard dissimilarity between $a(i)$ and $a(j)$:

$$d(i,j) = 1 - \frac{|a(i) \cap a(j)|}{|a(i) \cup a(j)|}$$

To produce an assignment of items into regions given a similarity function, we turn to the k-means^{++} algorithm [6]. The algorithm is known to converge quickly, and for clustering n items requires only $O(nkr)$ distance computations, where k is the number of clusters and r is the number of rounds performed by k-means.

4.1.1 Region $\mathcal{OTB}$-ness

We let $\texttt{items}(u)$ denote the set of items that are rated by u, $\texttt{items}(r)$ the set of items belong to region r, $\texttt{items}(u,r)$ the set of items belong to region r that are rated by u, i.e., $\texttt{items}(u,r) = \texttt{items}(u) \cap \texttt{items}(r)$ and, $\texttt{rating}(u,i)$ the known rating of user u for item i;

The *stickiness of a user u to a region r*, $\texttt{stick}(u,r)$ is the fraction between the number of items rated by u which belong to r over the total number of items rated by u. That is, $\texttt{stick}(u,r) = \frac{|\texttt{items}(u,r)|}{|\texttt{items}(u)|}$. For example, a user who rated 500 movies, 50 of which are Drama, would have a stickiness of 10% for the region {(genre=Drama)}. Intuitively, stickiness measures the degree of familiarity of a user toward a given region: the higher the stickiness, the more likely the user already knows about items within the region. Note that if the given region is the entire set of items ($\mathcal{I}$), then $\texttt{stick}(u,\mathcal{I}) = 1$ for any user u.

Similarly, we define the stickiness of a group of users (i.e., a network) N to a region r, $\texttt{stick}(N,r)$ is the average of each individual member's stickiness. Hence, $\texttt{stick}(N,r) = \frac{1}{|N|}\Sigma_{u \in N}(\texttt{stick}(u,r))$. Furthermore, we have the deviation of stickiness:

$$\texttt{stickDev}(N,r) = \sqrt{\frac{1}{|N|}\Sigma_{u \in N}(\texttt{stick}(u,r) - \texttt{stick}(N,r))^2}.$$

The network stickiness measures the familiarity toward the given region by a group of users collectively. The deviation of stickiness measures how consistent each member's stickiness is with the others. The lower the deviation, the more likely every member in the group is familiar (or unfamiliar) with items in the given region. When N is the entire group of users ($\mathcal{U}$), we have the global stickiness, $\texttt{stick}(\mathcal{U},r)$, and deviation, $\texttt{stickDev}(\mathcal{U},r)$, for the region.

There are two main factors in measuring a region's $\mathcal{OTB}$-ness for a given user: the level of unfamiliarity and the (under-)exposure potential. We combine those two factor in the definition of $\mathcal{OTB}$-ness and define the $\mathcal{OTB}$-ness for a region r by a given user u as $\texttt{otb}(u,r) = \texttt{otbBase}(u,r) \times \texttt{otbFactor}(u,r)$. where the *level of unfamiliarity* for r by u is defined as:

$$\texttt{otbBase}(u,r) = \frac{\texttt{stick}(\mathcal{U},r) - \texttt{stick}(u,r)}{\texttt{stickDev}(\mathcal{U},r)},\ \text{if } \texttt{stick}(\mathcal{U},r) > \texttt{stick}(u,r),$$

and 0 otherwise. And the *exposure factor* for r by u is defined as:

$$\texttt{otbFactor}(u,r) = \frac{\texttt{stick}(\mathcal{U},r) - \texttt{stick}(N,r)}{\texttt{stickDev}(\mathcal{U},r) + \texttt{stickDev}(N,r)} \times 2,\ \text{if } \texttt{stick}(\mathcal{U},r) > \texttt{stick}(N,r),$$

and 0 otherwise, where N is u's network. Here, normalization by the global deviation in `otbBase` is done to identify the regions whose unfamiliarity are the most statistically significant. And, a region has a high `otbFactor` if the user's network is unfamiliar with items in the region.

4.1.2 Region-Based Relevance

Identifying good items within $\mathcal{OTB}$ regions now becomes a challenge since neither the user nor the user's network knows much about items in those regions with a high $\mathcal{OTB}$-ness for that user (according to the definitions in Section 4.1.1). As a result, computing the user's expected rating, i.e., relevance, for items within those regions requires special attention. To address this question, we propose the notion of *region-region correlation* to identify the set of regions that *implies* each $\mathcal{OTB}$ region. We then construct an *expanded region network*, which consists of users who are similar to the target user based on items in those correlated regions.

We use association rules [5] to identify region-region correlations of the form $r \Rightarrow r'$ where r and r' are different regions in $R^{\mathcal{I}}$. A region s is a source region of a region r if and only if at least $x\%$ of the users who rate items in s also rate items in r, where x is a custom defined threshold. Source regions indicate general trends such as *people who rate Woody Allen movies also rate David Lynch movies*. We use $\texttt{sources}(r)$, to denote the set of all source regions of a region r. Based on this, $\texttt{exSim}(u,u',r)$, the expanded similarity of two users given a region can be defined as:

$\texttt{exSim}(u,u',r) = max_{r' \in \texttt{sources}(r)} \texttt{userSim}(u,u',r')$

where $\texttt{userSim}(u,u',r)$ is a similarity between two users restricted to region r, formally defined as:

$$\texttt{userSim}(u,u',r) = \frac{|\{i | i \in \texttt{items}(u,r) \ \wedge \ i \in \texttt{items}(u',r) \ \wedge \ |\texttt{rating}(u,i) - \texttt{rating}(u',i)| \leq 2\}|}{|\{i | i \in \texttt{items}(u,r) \ \vee \ i \in \texttt{items}(u',r)\}|}$$

where two ratings with a difference less than 2 (on a scale of 0 to 5) is considered to be similar. This is number is user customizable and chosen based on our experience.

The expanded region network, $\texttt{exNet}(u,r)$, for a user u and a region r is the set of users $u' \in \mathcal{U}$ such that $\texttt{exSim}(u,u',r) \geq \theta$ where θ is an application-dependent threshold. Intuitively, $\texttt{exNet}(u,r)$ is the set formed by users who share similar interests with u over source regions of r. We can now have:

$\texttt{relevance}(u,r,i) = \Sigma_{u' \in \texttt{exNet}(u,r)} \texttt{exSim}(u,u',r) \times \texttt{rating}(u',i)$.

4.2 Consolidation

Finally, we define the *overall score* of an item i as follows:

$$\texttt{overall}(u,i) := \Sigma_{r \in \texttt{regions}(i)} \texttt{otb}(u,r) \times \texttt{relevance}(u,r,i)$$

where $\texttt{regions}(i)$ is the set of all regions an item belongs to, $\texttt{otb}(u,r)$ denotes the $\mathcal{OTB}$ score of region r for user u and $\texttt{relevance}(u,r,i)$ is the region-specific relevance score of item i for user u.

5 Summary and Open Problems

The first hurdle facing any recommender system is that of accuracy, a system that cannot predict a user's affinity for an item cannot provide useful recommendations. However, we argue that once an acceptable accuracy bar (based on standard measures such as MAE) is reached, the system must begin to pay attention to other metrics such as diversity and freshness. Otherwise a system maximizing accuracy leads to recommendations that are predictable and boring, which adversely affect user loyalty.

Diversification is a clear answer; however, in the recommender systems literature, the term diversity conflates solutions to two related, but very different, problems. We distinguish between these and refer to them as overconcentration and predictability. In the former, the system recommends a homogeneous set of items. We tackle this problem by generating recommendations that are dissimilar to each other by leveraging explanations associated with recommendations. The latter has the system being overly safe, and recommending items that the user is likely to already be familiar with. Our approach for ensuring freshness is based on modeling what regions of the item space a user, her network, and the entire network, has been exposed to, and then generating recommendations that take this information into account.

The algorithms and approaches we described in this paper are applicable to memory-based recommender systems. Extending these approaches for model-based algorithms is an important open problem. More recently, there has been a push for context-aware recommender systems [4]. Understanding the interplay between the issues of diversity and freshness and the notion of context in such recommender systems is another interesting direction for future work.

References

[1] `http://en.wikipedia.org/wiki/Vector_space_model`

[2] Z. Abbassi, S. Amer-Yahia, L. V. S. Lakshmanan, S. Vassilvitskii, C. Yu, *Getting recommender systems to think outside the box*, in Proc. of the Recommender Systems Conference (RecSys), 285–288, 2009.

[3] G. Adomavicius, A. Tuzhilin, *Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions*, in IEEE Trans. Knowl. Data Eng., (17):6, 2005.

[4] G. Adomavicius, A. Tuzhilin, *Context-aware recommender systems*, in ACM Int. Conf. Recommender Systems, 2008.

[5] R. Agrawal, R. Srikant, *Fast Algorithms for Mining Association Rules in Large Databases*, in VLDB, 1994.

[6] D. Arthur, S. Vassilvitskii, *K-Means++: the advantages of careful seeding*, in SODA, 2007.

[7] D. Billsus, M. Pazzani, *User Modeling for Adaptive News Access*, in User Modeling and User-Adapted Interaction, (10):2/3, 147–180, 2000.

[8] S. Goel, A. Broder, E. Gabrilovich, B. Pang, *Anatomy of the Long Tail: Ordinary People with Extraordinary Tastes*, in WSDM, 2010.

[9] Y. Koren, *Recent Progress in Collaborative Filtering*, in ACM Int. Conf. Recommender Systems, 2008.

[10] S. M. McNee, J. Riedl, J. A. Konstan, *Being accurate is not enough: how accuracy metrics have hurt recommender systems*, in CHI Extended Abstracts, 2006.

[11] S. A. Munson, D. X. Zhou, P. Resnick, *Sidelines: An Algorithm for Increasing Diversity in News and Opinion Aggregators*, in AAAI, 2009.

[12] R. Sharma, G. Carenini, *Exploring more Realistic Measures for Collaborative Filtering*, in AAAI, 2004.

[13] C. Yu, L. Lakshmanan, S. Amer-Yahia, *It Takes Variety to Make a World: Diversification in Recommender Systems*, in Int'l Conf. on Extending Database Technology (EDBT), 2009.

[14] C. Yu, L. Lakshmanan, S. Amer-Yahia, *Recommendation Diversification Using Explanations*, in ICDE, 2009.

[15] C. N. Ziegler, S. M. McNee, J. A. Konstan, G. Lausen, *Improving Recommendation Lists Through Topic Diversification*, in Proc. of the World Wide Web Conference (WWW), 2005.

Addressing Diverse User Preferences: A Framework for Query Results Navigation

Zhiyuan Chen
University of Maryland, Baltimore County
zhchen@umbc.edu

Tao Li
Florida International University
taoli@cs.fiu.edu

Abstract

Database queries are often exploratory and users often find their queries return too many answers, many of them irrelevant. Existing approaches include categorization, ranking, and query refinement. The success of all these approaches depends on the utilization of user preferences. However, most existing work assumes that all users have the same user preferences, but in real life different users often have different preferences. In this paper, we propose a framework that addresses diverse user preferences in query results navigation.

1 Introduction

Database queries are often exploratory and users often find their queries return too many answers, many of them irrelevant. Three approaches have been proposed to solve this problem. The first approach [4] categorizes query results into a *navigational tree*. The second approach [2, 5, 1] ranks the results. The third approach refines queries based on user feedbacks [8]. The success of all three approaches depends on the utilization of user preferences. However, most existing work assumes that all users have the same user preferences, but in real life different users often have different preferences. In this paper, we propose a framework that addresses diverse user preferences in query results navigation. This framework uses a tree-based method and a cluster-based method to help user navigation. The tree-based method was originally proposed in [6]. Next we describe these methods in Section 2 and Section 3. Section 4 describes some future research directions.

2 Tree-based Method

We first describe a motivating example in Section 2.1. We then describe the tree based method in Section 2.2.

2.1 A Motivating Example

Example 1. Consider a mutual fund selection website. Figure 1 shows a fraction of a navigational tree generated using a method proposed in [4] over 193 mutual funds returned by a query with the condition fund_name like '%Vanguard%'. Each tree node specifies the range or equality conditions on an attribute, and the number in the

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

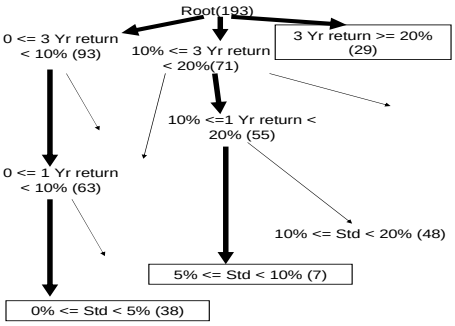


Figure 1: Tree generated by the existing method [4].

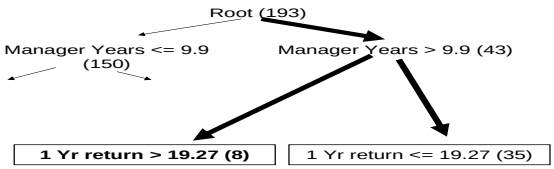


Figure 2: Tree generated by our tree-based method.

parentheses is the number of data records satisfying all conditions from the root to the current node. Users can use this tree to select the funds that are interesting to them. For example, a user interested in funds with very high returns may select those funds with "3 Yr return" over 20% (the right most node in the first level). Now only 29 records instead of 193 records need to be examined in the query results.

Consider four users U1, U2, U3, and U4. U1 and U2 prefer funds with high returns and are willing to take higher risks, U3 prefers funds with low risks and is willing to sacrifice some returns, and U4 prefers both high return funds and low risk funds (these two types of funds typically do not overlap). The existing method assumes that all users have the same preferences. Thus it places attributes "3 Year return" and "1 Year return" at the first two levels of the tree because more users are concerned with returns than risks. However, the attribute characterizing the risks of a fund is "standard deviation", and is placed at multiple nodes in the third level of the tree. Suppose U3 and U4 want to visit low risk funds at the two bottom-left nodes (bounded with rectangles) in Figure 1, they need to visit many nodes (including nodes on the paths from the root plus the sibling nodes because users need to check the labels of the siblings to decide which path to follow.).

2.2 Tree-based Navigation Method

User preferences are often difficult to obtain because users do not want to spend extra efforts to specify their preferences. Thus there are two major challenges to address the diversity issue of user preferences: (1) how to summarize diverse user preferences from the behavior of all users already in the system, and (2) how to decide the subset of user preferences associated with a specific user. As most existing work [4, 2, 5] did, we use query history to infer the behavior of all users in the system.

One well-known solution to the second challenge is to define a user profile for each user and use the profile to decide his/her preferences. However, in real life user profiles may not be available because users do not want to or can not specify their preferences (if they can, then they can form the appropriate query and there is no need for either ranking or categorization). One could try to derive a profile from the query history of a certain user, but this method will not work if the user is new to the system, which is exactly when the user needs help the most.

Clustering query history: We propose a two-step approach to address both challenges for the categorization case. The first step occurs offline. It analyzes query history of all users already in the system and generates a set of non overlapping clusters over the data, each corresponding to one type of user preferences. Each cluster has an associated probability of users being interested in that cluster. We assume that an individual user's preference can be represented as a subset of these clusters, and each cluster will appear in the subset with its associated probability. The system stores these clusters (by adding a class label to each data record) and probabilities for each cluster.

We propose two preprocessing steps to prune unimportant queries and merge similar queries. The algorithm is described in Figure 3. At step 1 and 2, the algorithm prunes unimportant queries and merges similar queries into a single query. At step 3 to 5, the clusters are generated.

Input: query results D_Q, data D, and query history $H = \{(Q_1, U_1, F_1), \ldots (Q_k, U_k, F_K)\}$ where Q_i is a query, U_i is session ID, and F_i is the weight of Q_i.

1. H'=Prune(H, D).
2. $\{QC_1, \ldots, QC_k\} = Merge(H')$ where QC_i is a merged query.
3. For each record $r_i \in D_Q$, identify $S_i = \{QC_p | \exists Q_j \in QC_p$ such that r_i is returned by $Q_j\}$.
4. Group records in D_Q by S_i.
5. Output each group as a cluster C_j, assign a class label for each cluster, and compute probability $P_j = \frac{\sum_{Q_i \in S_j} F_i}{\sum_{Q_p \in H'} F_p}$.

Figure 3: The algorithm to generate preference-based clusters.

In Example 1, the first step of our approach generates three clusters, one for high return funds, one for low risk funds, and the third for the remaining funds (i.e., those no users will be interested in).

Building a navigational tree: The second step occurs online when a specific user asks a query. It first intersects the set of clusters generated in the first step with the answers of the query. It then automatically constructs a navigational tree over these intersected clusters on the fly. This tree is then presented to the user. The user first navigates this tree to select the subset of clusters matching his/her needs. The user can then browse, rank, or categorize the results in the selected clusters.

Note that here we do not intend to select the most interesting records for the user. Instead, it is up to the user to do so. We just provide a navigational tree that "best describes" the differences of different clusters such that the user can easily locate the subset of clusters matching his/her needs. This step also does not assume the availability of a user profile or a meaningful query.

The diversity issue is addressed in two aspects. First, the first step of our approach captures diverse user preferences by identifying all types of user preferences in the format of clusters. Second, the second step uses a navigational tree to let a specific user select the subset of clusters (or types of user preferences) matching his needs. Let k be the number of clusters, the user can potentially select up to 2^k different preferences (subsets). This gives users more flexibility than the profiling approach because in the latter case each user is forced to use only one profile.

The navigational tree is generated in a similar way as constructing a decision tree over the query results. The major difference is the splitting criteria because we want to minimize the cost of navigating this tree. The standard splitting criteria for decision tree does not consider the difference between visiting a leaf with many records and visiting a leaf with very few records, as long as both leaves have similar distribution of classes. We try to maximize the following splitting criteria:

$$\frac{IGain(t, t_1, t_2)/E(t)}{(\sum_{j=1,2} N(t_j)(\sum_{C_i \cap t_j \neq \emptyset} P_i))/(N(t) \sum_{C_l \cap t \neq \emptyset} P_l)}. \tag{10}$$

Here t is the current tree node that will be split into child nodes t_1 and t_2. $IGain(t, t_1, t_2)$ is the information gain of splitting t into t_1 and t_2. $E(t)$ is the entropy of t. $N(t_j)$ is the size of child t_j. P_i is the probability of cluster C_i, which is computed in the clustering step.

The split should reduce the impurity in each leaf such that each leaf should contain records mostly from the same cluster. This is measured by the the nominator, which is the information gain normalized by the entropy of t. The split should also reduce the cost of visiting the leaf records. This is measured by the denominator, which is the cost of visiting leaf records after split, normalized by the cost before split.

For example, a navigational tree generated by our tree-based method is shown in Figure 2. Attribute "manager years" is selected in the first level of the tree. The intuition is that funds managed by very senior managers often have either very high returns, or very low risks because otherwise those managers may get fired for their

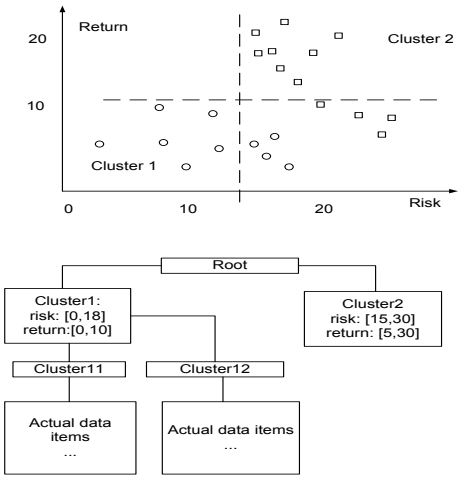


Figure 4: An example for the cluster-based method.

ExpandOneNode$(A,\ Q,\ c,\ A_u)$

1. sort $A - A_u$ in descending order of weights.
2. A_c = first k attributes in $A - A_u$ that appear in Q.
3. for each pair of attributes (A_i, A_j) in A_c
4. Generate clusters for records in c using (A_i, A_j)
5. Compute cost(c) using equation (13)
6. endfor
7. select the pair of attributes (A_{min_1}, A_{min_2}) with the lowest cost(c)
8. $A_u = A_u \cup \{A_{min_1}, A_{min_2}\}$
9. return A_{min_1}, A_{min_2}, and the set of clusters generated.

Figure 5: Algorithm to expand one node.

poor performance. "manager years" is not selected by the existing categorization method [4] because the most frequently used attribute is "3 Yr return" in the query history. Next "1 year return" is selected to separate the remaining two clusters. Now "high return" funds and "low risk" funds can be found at two leaf nodes with bounding rectangles. All four users only need to visit 4 tree nodes (2 in first level and 2 in the second level) excluding the root, while using the tree in Figure 1, U3 and U4 need to visit 12 nodes (3 in the first level, 5 in the second level, and 4 in the third level).

The results of experimental evaluation of the tree-based method can be found in [6]. The proposed method has been compared to a method without clustering query history (i.e., not considering diverse user preferences) and a method using standard decision tree splitting criteria (i.e., not considering the navigational cost). The results show that our method beats both alternatives in terms of navigational cost and user satisfaction.

3 Cluster-based Method

One major problem of the tree based approach is that it considers one attribute at a time. However, this is often inappropriate when the data contains correlated attributes. For example, the top half of Figure 4 shows the results of mutual funds where "risk" and "return" are correlated. As a result, the results can be divided into two clusters. However, the boundary of clusters can not be easily specified using a categorization tree. For instance, if we consider dividing the data along "risk"=15, part of cluster 1 will be on the same side as cluster 2. Similarly, it is difficult to partition along "return". Instead, we propose a cluster-based method to address this problem.

The proposed method uses a hierarchical navigational structure where each node in the hierarchy consists of a small number of clusters that have a concise description and can be easily visualized. For example, in the bottom half of Figure 4, we present to user a hierarchical structure where the first level contains two clusters: cluster 1 and cluster 2. Each cluster can be further divided into smaller clusters. The leaf level contains actual data records. To help users navigate the tree, each node also contains description of the cluster associated with this node. The description may include a representative record (e.g., using the centroid of the cluster), the ranges of some important attributes in the cluster, or a graphic visualization of clusters.

We propose a method to learn the distance function for clustering based on the correlation of query workload and data. We also propose a cost-based Greedy algorithm that constructs a hierarchy. Section 3.1 describes the learning algorithm. Section 3.2 describes the algorithm to construct the hierarchy. Section 3.3 reports the experimental evaluation of the cluster-based method.

3.1 Learning the Clustering Distance Function

In this section, we propose the algorithm to learn the distance function for clustering. In this paper we use weighted Minkowski distance function as follows.

$$d_w(A, B) = (\sum_{i=1}^{m} w_i |a_i - b_i|^p)^{\frac{1}{p}}. \quad (11)$$

Here A and B are two records, a_i or b_i is the i-th attribute of A or B, and w_i is the weight on the i-th attribute.

Our goal in the paper is to construct a cluster-based navigational hierarchy and we utilize the user preferences (in the form of query workloads) to dynamically assign weights to attributes. To automatically determine the weights for attributes, we use the metric learning approaches [9], which learn appropriate distance functions based on the correlations between attributes and workloads. We assume that when data records are similar, they tend to share similar preferences by many users, i.e., retrieved together by many queries.

Let n be the number of data records and m be the number of attributes. We can view the data as an $n \times m$ matrix R where the i-th row R_i is the i-th data record and $R_{ik}, 1 \leq k \leq m$, represents the k-th attribute of the i-th data record. Let q be the number of queries, the workload can also be represented as a $n \times q$ matrix Q where the i-th row Q_i indicates the user preferences for the i-th data records. Note that $Q_i = (Q_{i1}, \cdots, Q_{iq})$ where $Q_{i,v}, 1 \leq v \leq q$, is one if the i-th data record is selected by the v-th query and zero otherwise. In other words, Q is a binary matrix where each entry is either 1 or 0.

We use $d_R(i, j; \mathbf{w}) = \sqrt{\sum_{k=1}^{m} w_k (R_{i,k} - R_{j,k})^2}$ to denote the distance measurement among all pairs of records, based on the values of these records and parameterized by weights $\mathbf{w} = \{w_1, \cdots, w_m\}$. We assume values of numerical attributes have been normalized to [0,1]. If an attribute is categorical, we use as many binary attributes as the number of original attribute values. The value of a binary attribute corresponding to $\langle attribute_1, value_1 \rangle$ would be 1 if $attribute_1$ had $value_1$, and 0 otherwise. The binary attributes generated from the same categorical attribute have the same weight.

We use $d_Q(i, j) = \sqrt{\sum_{k=1}^{q} (Q_{i,k} - Q_{j,k})^2}$ to denote the distance measurement among all pairs of records, based on user preferences. Given that the user preferences are subjective judgments, a good weighting scheme for attributes should lead to a distance measurement that is consistent with the one based on user preferences. Thus to learn appropriate weights $\mathbf{w}$ for the attributes, we enforce the consistency between distance computation $d_R(i, j; \mathbf{w})$ and $d_Q(i, j)$, which leads to the following optimization problem:

$$\mathbf{w}^* = \arg\min \sum_{i \neq j} (d_R(i, j; \mathbf{w}) - d_R(i, j))^2 s.t. w_i \geq 0, w_i \in \mathbf{w} \quad (12)$$

We use the quadratic programming technique [3] to solve the above optimization problem. Given that user preferences can be noisy and may not accurately reflect the interests of individual users, we also introduce a regularization term as L_1 norm of weights, into the objective function in Equation (12) to eliminate small weights that are caused by accidental matches in workload. We also reduce the cost of quadratic programming by a condensation step. This step creates a large number of clusters (say 100) using an equal-weight distance function. Records in each cluster are condensed to the centroid of the cluster. Each cluster is then associated with the union of the sets of queries associated with each record in the cluster. We then run the quadratic programming algorithm on these centroid and their corresponding queries.

3.2 Hierarchy Construction Algorithm

In this section we propose a cost-based algorithm to construct the hierarchy. We first describe a cost estimation method, then propose an algorithm to select clustering attributes and generate clusters for a single node, and finally describe the complete algorithm.

Cost Estimation: We use the same model as in [4] to estimate the navigational cost in our hierarchy construction algorithm. Given a node v, let c be cluster associated with v and $c_i, 1 \leq i \leq m$ be the clusters associated with its children. Let $P(c_i)$ be the probability that users will visit c_i once they reach c, and $cost(c)$ be the cost for users to navigate the subtree rooted at v. Let K be the cost of checking the description of a cluster. The cost for the subtree rooted at v is given by the following formula:

$$Cost(c) = |c| * (1 - \sum_{i=1}^{m} P(c_i)) + \sum_{i=1}^{m} P(c_i) * (m * K + \sum_{i=1}^{m} P(c_i) * Cost(c_i)) \quad (13)$$

$\sum_{i=1}^{m} P(c_i)$ is the probability of visiting any of children of c. The cost is the sum of two terms. The first term computes the cost if users want to see all data records (the number of records is denoted as $|c|$) in c without visiting its children. The second term computes the cost of visiting its children and the subtree rooted at its children. Thus given $P(c_i)$ for every node in the hierarchy, we can recursively compute the total cost of navigating a hierarchy.

$P(c_i)$ can be estimated based on the query workload. Suppose there are q queries in the workload that overlap with cluster c (i.e., these queries select at least one record in c), and q_i queries in the workload that overlap with cluster c_i, then $P(c_i) = q_i/q$.

Cluster Generation for One Node Figure 5 shows the algorithm to expand a single node in the hierarchy. Here A is the set of attributes, Q is the query workload, c is the cluster for current node v, and A_u is the set of attributes that have been used to generate clusters associated with v and its ancestors.

The algorithm uses two heuristics to select clustering attributes: (1) those attributes with higher weights are important and shall be considered first; (2) attributes frequently appearing in queries are also important for users. To combine these two heuristics, we sort available attributes in $A - A_u$ in the descending order of weights (obtained using the method described in Section 3.1) and select the first k attributes that also appear in the query workload. We use $k = 3$ in this paper because we find it works quite well in experiments. In line 3 to 6, we consider all possible pairs of these k attributes to generate several clustering schemes, and estimate the cost for each clustering scheme. Line 7 selects the clustering scheme with the lowest cost.

We use the K-Means algorithm for clustering. K-Means requires us specify the number of clusters. We run K-Means with several numbers (3, 4, and 5 in this paper), and select the one with the minimal cost. This number can also be set by users and we find 3-5 work quite well in experiments. Finally, line 8 adds selected attributes to the set of used attributes A_u, and returns the selected attributes and generated clusters.

Hierarchy Generation: We propose a Greedy algorithm to generate the hierarchy. The algorithm first learns weights of attributes. We only consider attributes in workload. The algorithm then starts from the root, and repeatedly calls ExpandOneNode to expand the leaves in the partially constructed hierarchy if there are more than t records in the node. t is a parameter that can be set by users. We use $t = 10$ in our experiments.

ExpandOneNode considers $k(k-1)/2$ pairs of attributes. The clustering algorithm will be called 3 times for each pair (since we run K-Means for 3, 4, 5 clusters). We use $k = 3$ in this paper so K-Means is called 9 times per node. The total cost of the algorithm depends on the number of internal nodes in the hierarchy (leaf nodes do not need to be clustered). Each node can have at most 5 children (since we generate at most 5 clusters). Let q be the number of attributes in Q. The maximal height of the tree is $q/2$ because we do not use the same attribute twice in any root-to-leaf path. Thus the cost of the algorithm is $O(5^{q/2-1}9C)$ where C is the cost of one execution of K-Means. In practice, we can select only those attributes frequently appearing in workload. Thus q is usually small.

3.3 Experimental Evaluation

In this section we present the results of an empirical study.

Experimental Setup: We conducted our experiments on a machine with Intel PIII 996MHZ CPU, 512M RAM, and running Windows XP Professional version 2002.

Table 1: Navigational Cost Per Fund

User	Greedy	Non-Clustering	UPGMA-All	UPGMA-Q	Random-All	Random-Q	Freq-Q
User1	6.66	15.83	9.33	11.33	14.5	9.5	9.83
User2	12	22.5	29.83	23.83	22.5	21.16	14.6
User3	7.75	13.5	10.25	12.75	21.75	11.25	10.5
User4	10.25	13.25	11	12	13.5	12.5	14
User5	7.5	12	20	15	23.5	19	15
User6	8.5	17.16	18.5	15.66	19.5	15.2	18.2
User7	14.4	24	15.2	16.6	25.6	20.2	19.2
Average	9.75	17.69	16.57	15.66	19.84	15.42	14.54

We used a dataset of 18537 Mutual funds downloaded from *www.scottrade.com*. There are 33 attributes, 28 with numerical values and 5 with categorical values.

We asked 7 users to ask queries against the dataset. Each query had 0 to 6 range conditions. We collected altogether 62 queries. There were six attributes in search conditions of these queries and each query had 4.4 conditions on average.

We implemented the proposed Greedy algorithm using JAVA. We compare the Greedy algorithm with 6 other algorithms: (1) Non-Clustering: the non-clustering based algorithm proposed in [4]. (2) UPGMA-all: this is a well known hierarchical clustering algorithm [7]. It first uses K-Means to create clusters, using the same weight over all attributes. It then generates a hierarchy by repeatedly merging clusters with the minimal inter-cluster distances [7]. (3) UPGMA-Q: this is the same as UPGMA but using only attributes in query workload, and still using the same weight on those attributes. (4) Random-all: this algorithm creates the hierarchy by using 2 attributes for clustering at each node. However, the clustering attributes are randomly selected from all attributes in the data and each clustering attribute has the same weight. (5) Random-Q: same as Random-all but only selects attributes in the query workload. (6) Freq-Q: same as Random-all but uses a heuristics to choose clustering attributes. Attributes in the workload are sorted in descending order of their frequencies in the workload, and the first pair is used for clustering data in the first level of the hierarchy, and the second pair is used for the second level, and so forth.

We use the average navigational cost to measure the quality of hierarchies. The same set of users were asked to provide us several mutual funds that they found worth investing. Each user selected 2-6 funds and 33 funds were selected in total. For each user, we computed the total cost to retrieve these funds. We then divided this cost by the number of funds that the user selected.

Results: We have run all algorithms on the results of various SQL queries. Due to space limit, we only report the results of the following SQL query: "select * from AllFunds where FundFamily = 'Vanguard' ". The running time of all algorithms is well under one second. Table 1 reports the average navigational cost for each user. The results for Random-all and Random-Q are the average of 5 runs.

Greedy is the best among all algorithms for all users, and is on average about 103%, 58%, and 49% better than Random-All, Random-Q, Freq-Q, respectively. The average cost of Greedy is about 81% smaller than the cost for Non-Clustering. Non-Clustering algorithm generates very deep trees because it ignores the correlations in data. The results also show that Greedy is on average over 60% better than the two conventional hierarchical clustering algorithms UPGMA-All and UPGMA-Q. Furthermore, the hierarchy generated by hierarchical clustering algorithms is difficult for users to navigate because clusters in the hierarchy are generated over many attributes (33 for UPGMA-All and 6 for UPGMA-Q), making it difficult for users to visualize.

Interestingly, the performance of Freq-Q is only slightly better than Random-Q. The reason is that simply looking at the frequencies of attributes in the workload may not give the correct order of importance. For example, '1 year total return' and '3 year total return' are the most frequent attributes in the workload. However, these two attributes are highly correlated in data. Thus clustering using these two attributes do not provide more

information than clustering using just one of them.

Our distance learning algorithm is based on the assumption that when data records are similar, they tend to share similar preferences by many users, i.e., retrieved together by many queries. The experimental results verify this assumption. For example, we found that most users were aware of the correlation between attributes such as "1 year return" and "3 year return", so they often included only one of these two attributes in the queries along with other attributes such as 'risk' and 'minimal investment'. Thus the proposed distance learning algorithm gives higher weights to '1 year total return', 'risk', and 'minimal investment', leading to clusters that give more information for users.

4 Outlook

In this paper, we show how to address diverse user preferences when helping users navigate SQL query results. There are many directions for future research. First, it will be interesting to study how to address diverse user preferences for the two other approaches: ranking and query refinement. Further, most research efforts so far focus on reducing the amount of information presented to user (e.g., through clustering, grouping, or ranking). Another interesting direction is to study how to better visualize the results. More rigorous usability study is also needed in this field. So far, most research work only uses 10-20 participants and reports the degree of satisfaction. More rigorous studies need to recruit more participants, consider more human factors (e.g., gender and age), and collect more measurements (e.g., the time spent by a participant).

References

[1] R. Agrawal, R. Rantzau, and E. Terzi. Context-sensitive ranking. In *SIGMOD Conference*, pages 383–394, 2006.

[2] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *CIDR*, 2003.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[4] K. Chakrabarti, S. Chaudhuri, and S. won Hwang. Automatic categorization of query results. In *SIGMOD*, pages 755–766, 2004.

[5] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic ranking of database query results. In *VLDB*, pages 888–899, 2004.

[6] Z. Chen and T. Li. Addressing diverse user preferences in SQL-query-result navigation. In *SIGMOD '07*, pages 641–652, 2007.

[7] C. D. Michener and R. R. Sokal. A quantitative approach to a problem in classification. *Evolution*, 11:130–162, 1957.

[8] N. Sarkas, N. Bansal, G. Das, and N. Koudas. Measure-driven keyword-query expansion. *PVLDB*, 2(1):121–132, 2009.

[9] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems 15*, pages 505–512, 2003.

Diversity over Continuous Data

Marina Drosou
Computer Science Department
University of Ioannina, Greece
mdrosou@cs.uoi.gr

Evaggelia Pitoura
Computer Science Department
University of Ioannina, Greece
pitoura@cs.uoi.gr

Abstract

Result diversification has recently attracted much attention as a means of increasing user satisfaction in recommendation systems and web search. In this work, we focus on achieving content diversity in the case of continuous data delivery, such as in the context of publish/subscribe systems. We define sliding-window diversity and present a suite of heuristics for its efficient computation along with some performance results.

1 Introduction

With the explosion of the amount of information currently available online, publish/subscribe systems offer an attractive alternative to searching by providing a proactive model of information supply. In such systems, users express their interest in specific pieces of data (or events) via subscriptions. Then, they are notified whenever some other user generates (or publishes) an event that matches one of their subscriptions. Typically, all subscriptions are considered equally important and users are notified whenever a published event matches any of their subscriptions. However, getting notified about all matching events may lead to overwhelming the users with large amounts of notifications, thus hurting the acceptability of publish/subscribe systems.

Today, most user searches have an exploratory nature, in the sense that users are mostly interested in retrieving various information about their search topic. Therefore, recently, *result diversification* has attracted considerable attention as a means of enhancing user satisfaction in recommender systems and web search (e.g. [18, 14]). We argue that diversification could also be employed in the context of publish/subscribe systems to improve the overall quality of notifications delivered to users.

In this paper, we tackle the problem of selecting k diverse information pieces (or *items*) among the information content being forwarded to users and delivering only these items to the users instead of overwhelming them with all relevant information. Diverse items may be defined in three different ways, namely in terms of (i) *novelty*, i.e. choosing to deliver items that contain new information when compared to previously delivered ones (e.g. [5, 17]), (ii) *coverage*, i.e. choosing to deliver items that belong to different categories (e.g. [3]) and (iii) *content* (or *similarity*), i.e. choosing to deliver items that are dissimilar to each other (e.g. [16]). Motivated by the fact that publish/subscribe systems are simultaneously used by many different publishing sources that often publish overlapping information, in this work, we focus on content diversity among the items (i.e. events)

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

of a publish/subscribe system and seek to process the continuous flow of information in such a way as to locate and deliver to users events that are distant (dissimilar) to each other, given a budget of k.

In this paper, we introduce the problem of diversity over continuous data, present initial algorithms and suggest issues for further research. The rest of this paper is structured as follows. In Section 2, we define the k-diversity problem and present efficient solutions, while in Section 3, we adapt these solutions for continuous data. Section 4 focuses on the combination of multiple criteria, namely diversity and relevance, for the final ranking of information. Section 5 briefly reviews related work and, finally, Section 6 concludes this paper.

2 Content Diversity

There are various forms of diversity. Here, we focus on content diversity, so that the users receive dissimilar content. Specifically, given a set of n items, we aim at selecting k items out of them, such that, the average pairwise distance between the selected items is maximized. More formally:

Definition 1: Let $P = \{p_1, \ldots, p_n\}$ be a set of n items and k an integer with $k \leq n$. Let also $d(p_i, p_j)$, $p_i, p_j \in P$, be a distance metric between items p_i and p_j. The k-diversity problem is to locate a subset S^* of P, such that:

$$S^* = \underset{\substack{S \subseteq P \\ |S|=k}}{\operatorname{argmax}} f_D(S), \text{ where } f_D(S) = \frac{1}{k(k-1)} \sum_{i=1}^{k} \sum_{j>i}^{k} d(p_i, p_j) \text{ and } p_i, p_j \in S \tag{14}$$

The problem of selecting the k items having the maximum average pairwise distance out of n items is similar to the *p-dispersion* problem. This problem, as well as a number of its variations (e.g. select p out of n items so that the minimum distance between any two of them is maximized), have been extensively studied in operations research and are in general known to be NP-hard [8, 9]. Thus, to solve large instances of the problem, we need to rely on heuristics. A number of heuristics have been proposed in the literature (e.g. [9]), ranging from applying exhaustive algorithms to adapting traditional optimization techniques. A general issue that hinders the development of efficient incremental solutions to the problem is that the $k-1$ most diverse items of a set P are not necessarily a subset of its k most diverse items. For example, consider as items the points on the circumstance of a circle and their euclidean distances. The two furthest apart points are (any) two antidiametric ones. However, no antidiametric points belong to the three most diverse points.

There are two families of heuristics that locate good solutions for the k-diversity problem at a reasonable time: *greedy* heuristics and *interchange* heuristics (the reader is referred to [6] for a study of various heuristics and their behavior). Greedy heuristics make use of two sets: the initial set P and a set S which will eventually contain the selected items. Items are iteratively moved from P to S and vice versa until $|S| = k$ and $|P| = n-k$. There are two main variations. In the *Greedy Construction* heuristic, initially, $|P| = n$ and $|S| = 0$. First, the two furthest apart items of P are added to S. Then, at each iteration, one more item is added to S. The item that is added is the one that has the maximum item-set distance from S. We define the *item-set distance* $setdist(p_i, S)$ between an item p_i and a set of items S as the average distance between p_i and the items in S, that is:

$$setdist(p_i, S) = \frac{1}{|S|} \sum_{p_j \in S} d(p_i, p_j) \tag{15}$$

In the *Greedy Deletion* heuristic, initially, $|P| = 0$ and $|S| = n$. At each iteration, the two closest items of S are located. One of them is then moved to P. The choice is based on the minimum item-set distance from the remaining items of S.

Generally, the Greedy Construction heuristic (denoted GC, Algorithm 2) performs better, both in terms of the achieved diversity as well as in terms of execution time. Therefore, we will consider it further in this work.

Algorithm 2 Greedy Construction Heuristic (GC).

Input: The initial set of items P, the number of wanted items k and a threshold r.
Output: The set S with the k most diverse items.

1: Set R to be a random subset of P with $|R| = r$
2: find p_1, p_2, s.t. $d(p_1, p_2) = \max\{d(p_1, p_2) : p_1, p_2 \in R, p_1 \neq p_2\}$
3: $S \leftarrow \{p_1, p_2\}$
4: **while** $|S| < k$ **do**
5: find $p_i \in P \backslash S$, s.t. $setdist(p_i, S) = \max\{setdist(p_j, S) : p_j \in P \backslash S\}$
6: $S \leftarrow S \cup \{p_i\}$
7: **end while**
8: **return** S

Algorithm 3 First Pairwise Interchange Heuristic (FI).

Input: The initial set of items P, the number of wanted items k and a threshold h.
Output: The set S with the k most diverse items.

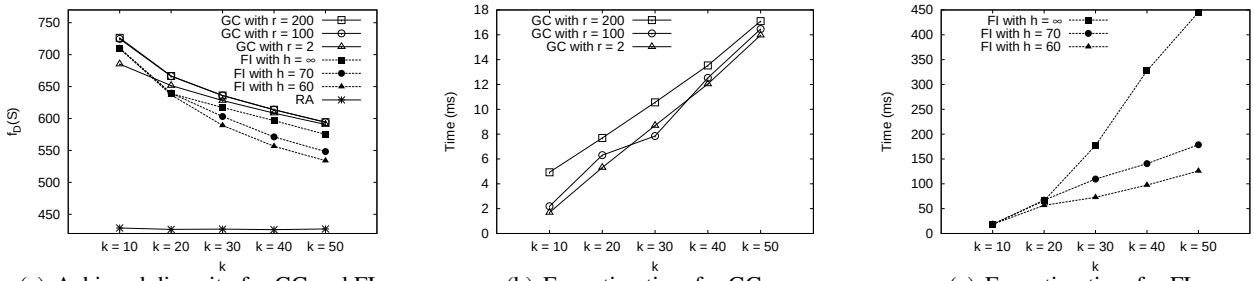
1: Set S to be a random solution
2: **while** less than h iterations have been performed **do**
3: find p_1, p_2, s.t. $d(p_1, p_2) = \min\{d(p_1, p_2) : p_1, p_2 \in S, p_1 \neq p_2\}$
4: **for all** $p_i \in P \backslash S$ **do**
5: $S' \leftarrow \{S \backslash \{p_1\}\} \cup \{p_i\}$
6: $S'' \leftarrow \{S \backslash \{p_2\}\} \cup \{p_i\}$
7: **if** $f(S') > f(S)$ and $f(S') \geq f(S'')$ **then**
8: $S \leftarrow S'$; **break**
9: **end if**
10: **if** $f(S'') > f(S)$ and $f(S'') > f(S')$ **then**
11: $S \leftarrow S''$; **break**
12: **end if**
13: **end for**
14: **end while**
15: **return** S

The complexity of GC is $O(n^2)$. However, this is due to the first step of the algorithm where the two furthest apart items have to be located. The rest of the algorithm takes $O(k^2 n)$ time. Therefore, to reduce the distance computations required by GC, we opt to initialize S by selecting the two furthest apart items from a randomly selected subset of P with size equal to r, $r < n$, instead of using the whole set.

Interchange heuristics are initialized with a random solution S and then iteratively attempt to improve that solution by interchanging an item in the solution with another item that is not in the solution. The item that is eliminated from the solution at each iteration is one of the two closest items in it. Again, there are two main variations. The *First Pairwise Interchange* heuristic (denoted FI) performs at each iteration the first interchange that improves the solution, while the *Best Pairwise Interchange* heuristic (denoted BI) considers all possible interchanges and performs the one that improves the solution the most.

None of the two algorithms clearly outperforms the other, while their worst case complexity is $O(n^k)$. Each distinct iteration of FI is on average faster than an iteration of BI. However, FI is more likely to perform more iterations. Even though there is no clear winner in terms of execution time, FI usually locates better solutions [9, 6]. This is the reason we will focus on this variation (Algorithm 3) in the rest of this work. There are two ways to limit the iterations performed by FI. We can either set a bound, say h, on the maximum number of possible interchanges to be performed or allow interchanges to continue as long as the solution improves by a given threshold. In this paper, we choose to directly control the number of iterations, so that we can get reasonable execution times.

Next, we provide experimental results to evaluate the performance of these heuristics, both in terms of efficiency (execution time) as well as effectiveness (achieved diversity). Our goal is not to provide a comprehensive evaluation of the two heuristics but to present a couple of experiments to provide some insight about their performance. We create synthetic datasets consisting of 200 data points in the euclidean space, where the values of each dimension are in $[0, 1000]$ and use as distance d the euclidean distance. The heuristics were implemented in JDK 6 and run on a Windows XP PC with a 2.4 GHz Intel Core 2 Processor and 2.00 GB of RAM. In the following, we show results averaged over 500 runs for 5-dimensional data items drawn from a normal distribution. Figure 1 shows results for various values of k for the two heuristics. For GC, we use $r = 200$ (i.e. initializing the heuristic with the whole set P), $r = 100$ and $r = 2$ (i.e. initializing the heuristic with two random items), while for FI, we either allow a certain number of iterations ($h = 60$ or $h = 70$) or allow the algorithm to run until



(a) Achieved diversity for GC and FI. (b) Execution time for GC. (c) Execution time for FI.

Figure 1: Average achieved diversity and execution time for GC and FI.

convergence ($h = \infty$). We also report the average diversity of k randomly selected items (denoted RA). Both heuristics achieve comparable diversity with GC slightly outperforming FI. GC is much faster that FI. Also, note that reducing the initial items r considered by GC does not affect the achieved diversity considerably.

3 Content Diversity in Publish/Subscribe Systems

Publish/subscribe systems provide an alternative way of receiving data of interest. In such systems, users express their interests in items through subscriptions. New items (or events) published by an information source (or publisher) are matched against the subscriptions and those items that match subscriptions are delivered to the corresponding users. Some examples of publish/subscribe systems or proactive delivery include news alerts, RSS feeds and notification services in social networks. As with web search, user subscriptions are often exploratory in nature, in the sense that users do not really know what they want and may not be precise on expressing their information needs. Thus, recent works have suggested that event matching should also be best effort [13, 19]. For this reason, to further enhance user satisfaction, we consider the case where not all matching items are forwarded to the subscribers but just the k most diverse of them.

This is an instance of continuous data delivery. Since events are published and matched in a continuous manner, we need to define over which subsets of data we apply diversification. In our previous work [7], we have considered three fundamental models for item delivery: (i) *periodic*, (ii) *sliding-window* and (iii) *history-based filtering* delivery. Given a budget of k, with periodic delivery, the k most diverse items are computed over disjoint periods of length T and are forwarded to the subscribers at the end of the period. With sliding-window delivery, the k most diverse items are computed over sliding windows of length w, so that, an item is forwarded, if and only if, it is part of the k most diverse items in any of the windows it belongs to. Finally, history-based filtering forwards new items as they are matched, if and only if, they are dissimilar enough to the k most diverse items recently delivered. The lengths T and w can be defined either in time units (e.g. as "the 10 most diverse items matched per hour" and, respectively, "the 10 most diverse items matched in the last hour") or in number of items (e.g. as "the 10 most diverse items per 100 matched ones" and, respectively, "the 10 most diverse items among the 100 most recently matched ones").

Here, we allow windows not only to slide but also to "jump", i.e. move forward more than one position in the stream of matching items each time. We call these windows *jumping windows* (Figure 2). Assuming windows of length w and a jump step of length j, with $j < w$, consequent windows overlap and share $w - j$ common items. Note that, for $w = 1$, we get sliding-window delivery, while for $w = T$, we get periodic delivery. We denote the i^{th} window as W_i and write $W_i = \{p_1, \ldots, p_w\}$, where $p_1, \ldots, p_w$ are the items that belong to W_i. Next, we define the k-diversity problem over continuous data:

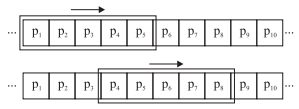


Figure 2: A jumping window with $w = 5$ and $j = 3$.

Definition 2: Let $\mathbb{W}$ be a stream of items and consider a window sliding over $\mathbb{W}$. The sliding-window k-diversity problem is the following: In each window W_i of $\mathbb{W}$, locate and forward to the user a set S_i^*, such that:

$$S_i^* = \underset{\substack{S_i \subseteq W_i \\ |S_i|=k}}{\operatorname{argmax}} f_D(S_i) \tag{16}$$

One difficulty of computing diverse items over continuous data is that it cannot be done incrementally. Let S and S' be two sets that differ at only one item. Then, the k most diverse items of S are in general completely different than the k most diverse items of S'. As a simple example, consider 2-dimensional points in the euclidean space and the sets $S = \{(0,0),(3,3),(5,6),(1,7)\}$ and $S' = \{(3,3),(5,6),(1,7),(4,4)\}$. Then, the 2 most diverse items of S are $(0,0)$ and $(5,6)$, while the 2 most diverse items of S' are $(1,7)$ and $(3,3)$.

The straightforward solution to the problem is to re-compute the k most diverse items at each window. To do this, we will consider the GC heuristic, since, as we showed in Section 3, it consistently outperforms FI, especially in terms of execution time, which is crucial in real-time systems such as publish/subscribe. We also consider the following variation. After each window jump, some of the previous k most diverse items (say m of them) leave the window. Therefore, we initialize the GC algorithm with the $k-m$ remaining most diverse items from the previous window and then let the algorithm add m items from the new window based on their item-set distances. We denote this variation as SGC.

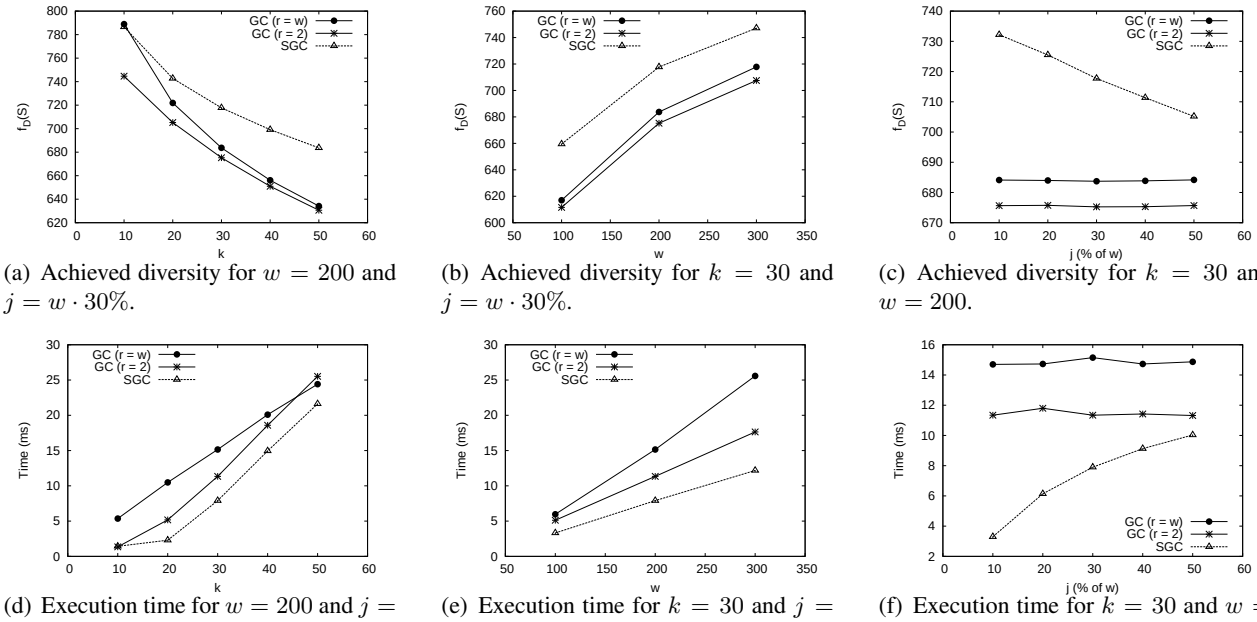
Next, we evaluate the performance of GC versus SGC. We examine GC for $r = w$ and $r = 2$. We use a stream of 10000 5-dimensional data points drawn from a normal distribution and vary k, w and j. In Figure 3, we report the average diversity achieved in each window of the stream and the corresponding average execution time. We observe that SGC constantly behaves better than both versions of GC, both in terms of diversity and time. The achieved diversity of both heuristics decreases for larger values of k and smaller values of w, as expected. SGC performs better as k increases (or w decreases), since there are more diverse items from the previous window that still remain in the new one. For the same reason, a large jump step causes the performance of SGC to degrade. The behavior of GC does not depend on the length of the jump, as the k most diverse items are recomputed at every window.

4 Content Diversity and Relevance

Often, diversity is just one of the quality aspects in information delivery. In general, items are also ranked based on their relevance to a user query or subscription. Besides relevance, the ranking of an item may be an indicator of its importance associated, for example, with the authority of the publisher or with the specific preferences or interests of the user. We assume that this ranking is expressed using a numeric value (or score) associated with each item p_i through some ranking function $f_R(p_i)$ that takes values in [0,1]. Then, the score of a set of items is the average score of its members.

This creates a multi-criteria problem, since we want to deliver the set of k items that are both highly preferred and diverse. There are basically two approaches to combine the two criteria, i.e. diversity and relevance: (i) employing a weighted combination of the two criteria through a weight σ and (ii) a threshold variation where we seek to maximize diversity given a minimum required score for the selected set of items (or, the dual, maximize the score of the selected items given a minimum required diversity). Placing a threshold on diversity however may be hard, since it requires an estimation of the achievable diversity.

In this work, we focus on a weighted combination of relevance and diversity. Assuming that items of interest take values from a non-infinite domain, we use M to denote the maximum possible distance between any two items. Then, we redefine the k-diversity problem as locating a subset S^* of P, such that:



(a) Achieved diversity for $w = 200$ and $j = w \cdot 30\%$.

(b) Achieved diversity for $k = 30$ and $j = w \cdot 30\%$.

(c) Achieved diversity for $k = 30$ and $w = 200$.

(d) Execution time for $w = 200$ and $j = w \cdot 30\%$.

(e) Execution time for $k = 30$ and $j = w \cdot 30\%$.

(f) Execution time for $k = 30$ and $w = 200$.

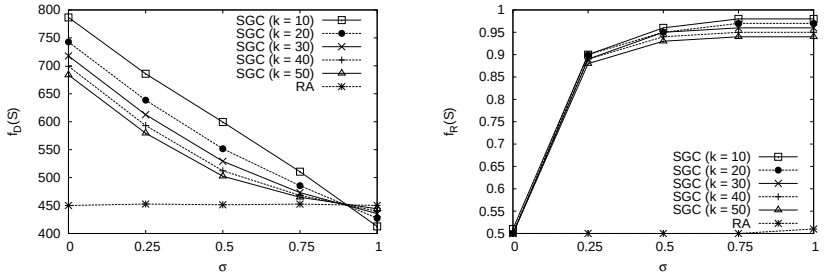
Figure 3: Average achieved diversity and execution time for the GC and SGC heuristics.

$$S^* = \underset{\substack{S \subseteq P \\ |S|=k}}{\operatorname{argmax}} g(S), \text{ where } g(S) = \sigma \cdot \frac{1}{|S|} \sum_{p_i \in S} f_R(p_i) + (1-\sigma) \cdot \frac{1}{M} f_D(S), \text{ with } \sigma \in [0,1] \qquad (17)$$

Many interesting questions arise concerning the best way to combine the two measures, such as, what is the best value for the weight and whether we can adjust it dynamically based on the dataset. Here, we assume that items are associated with uniformly distributed scores. Also, the item-set distance $setdist(p_i, S)$ between an item p_i and a set of items S (Equation 15) used by SCG is re-defined accordingly to Equation 17 to consider both measures. In Figure 4, we show the average archived diversity and relevance for the windows of our stream of 10000 data items when varying σ for the SGC heuristic and for the various values of k. Figure 4 also shows the corresponding results when k random items are chosen in each window (denoted RA). Naturally, by varying the value of σ, we can tune the trade-off between the achieved diversity and relevance. We notice that when choosing items based solely on relevance ($\sigma = 1.00$), the achieved diversity drops below that of the random case (RA). In this setting, we observe a great increase in relevance as σ increases, even for $\sigma = 0.25$, while the reduction of diversity is more gradual. In this case, there is no reason to use $\sigma > 0.50$ since this results in further reduction of diversity without an analogous gain of relevance.

5 Related Work

Although the combination of diversity and relevance has attracted considerable attention recently, it is not a new concept. [4] proposed a linear combination of the two measures back in 1998, similar to the approaches more recently followed in [18, 7]. In [18], a method for topic diversification is proposed for recommendations based on the intra-list similarity, a permutation-insensitive metric introduced to assess the topical diversity of a given recommendation list. The proposed algorithm also considers the relevance of the candidate items when creating recommendation lists. [7] applies the concept of ranking based on both relevance and diversity in the context of publish/subscribe systems. The notion of diversity is also explored in database systems. Motivated



(a) Achieved diversity. (b) Achieved relevance.

Figure 4: Average achieved diversity and relevance for various values of σ.

by the fact that some database relation attributes are more important to the user, [14] proposes a method where recommendation lists consisting of database tuples are diversified by first varying the values of higher priority attributes before varying the values of lower priority ones. When the tuples are associated with scores, a scored variation of the method picks more relevant to the query tuples first. [12] tackles the problem of automatically extracting a set of features from the items of interest that is capable of differentiating the items from each other. [16] formulates the k-diversity problem as an optimization problem. Given the $n \times n$ distance matrix of the candidate items, the goal is to find a binary vector of size n that represents which items belong to the most diverse subset of size k. This is a binary problem which has to be relaxed to a real-valued problem to be solved. Recently, [10] defined a set of natural, intuitive axioms that a diversification system is expected to satisfy. The authors prove that all axioms cannot hold simultaneously and show which of them are satisfied by three main diversification methods.

Besides using content diversity, there is also related work based on different definitions of diversity. [15] proposes computing diversity based on the notion of explanations. The explanation of a given item for a user is the set of similar items the user has rated in the past. Distances between two items are measured based on their corresponding explanations. [11] proposes algorithms to capture diverse concepts in text documents, aiming at retrieving diverse sentences that can be used as snippets from search engines. These algorithms are based on coverage, i.e. how many terms of the document appear in the selected sentences, and orthogonality, i.e. how much the terms appearing in such a sentence differ from those appearing in the rest. Coverage is also considered in [3], which aims at locating diverse documents to answer a user query. Given a taxonomy of topics and a probability distribution for topics relevant to the query, the k-diversity problem is formulated as the location of a set of documents with cardinality k that maximizes the probability of as many topics as possible being represented by that set. In [5], a distinction is made between novelty, i.e. avoiding redundancy, and diversity, i.e. resolving ambiguity. Both user queries and documents are broken down into small pieces of information, called nuggets. Then, diverse documents are retrieved based on a probabilistic model of nuggets being contained in the various queries and documents.

6 Conclusions

Deriving efficient and effective algorithms for content diversity, let alone continuous content diversity, is still an open problem; we believe as broad as clustering. This paper attempts to serve as one step towards exploring some of the issues involved. To this end, we have focused on two intuitive heuristic solutions to the problem, namely a greedy and an interchange one, and their variants. Information filtering as in publish/subscribe is an instance where the need for continuous diversity arises. Publish/subscribe delivery also raises issues related to distributed data management, since both the publishers of items and the subscribers are distributed. In addition, for performance and reliability, the matching of subscriptions and publications is often done distributed. We

have implemented diversity and ranking based on user preferences in our prototype, called PrefSIENA [1], that extends SIENA [2], a popular publish/subscribe system. We are currently working on distribution issues.

References

[1] PrefSIENA. http://www.cs.uoi.gr/∼mdrosou/PrefSIENA.

[2] SIENA. http://serl.cs.colorado.edu/∼serl/dot/siena.html.

[3] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM*, 2009.

[4] J. G. Carbonell and J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, 1998.

[5] C. L. A. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon. Novelty and diversity in information retrieval evaluation. In *SIGIR*, 2008.

[6] M. Drosou and E. Pitoura. Comparing diversity heuristics, Technical Report 2009-05. Computer Science Department, University of Ioannina, 2009.

[7] M. Drosou, K. Stefanidis, and E. Pitoura. Preference-aware publish/subscribe delivery with diversity. In *DEBS*, 2009.

[8] E. Erkut. The discrete *p*-dispersion problem. *European Journal of Operational Research*, 46(1), 1990.

[9] E. Erkut, Y. Ülküsal, and O. Yeniçerioglu. A comparison of *p*-dispersion heuristics. *Computers & OR*, 21(10), 1994.

[10] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, 2009.

[11] K. Liu, E. Terzi, and T. Grandison. Highlighting diverse concepts in documents. In *SDM*, 2009.

[12] Z. Liu, P. Sun, and Y. Chen. Structured search result differentiation. *PVLDB*, 2(1), 2009.

[13] A. Machanavajjhala, E. Vee, M. N. Garofalakis, and J. Shanmugasundaram. Scalable ranked publish/subscribe. *PVLDB*, 1(1), 2008.

[14] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, and S. Amer-Yahia. Efficient computation of diverse query results. In *ICDE*, 2008.

[15] C. Yu, L. V. S. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *EDBT*, 2009.

[16] M. Zhang and N. Hurley. Avoiding monotony: improving the diversity of recommendation lists. In *RecSys*, 2008.

[17] Y. Zhang, J. P. Callan, and T. P. Minka. Novelty and redundancy detection in adaptive filtering. In *SIGIR*, 2002.

[18] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW*, 2005.

[19] C. Zimmer, C. Tryfonopoulos, and G. Weikum. MinervaDL: An architecture for information retrieval and filtering in distributed digital libraries. In *ECDL*, 2007.

Efficient Computation of Diverse Query Results*

Erik Vee, Jayavel Shanmugasundaram, Sihem Amer-Yahia

Yahoo! Research
Sunnyvale, CA, USA
{erikvee,jaishan,sihem}@yahoo-inc.com

Abstract

We study the problem of efficiently computing diverse query results in online shopping applications, where users specify queries through a form interface that allows a mix of structured and content-based selection conditions. Intuitively, the goal of diverse query answering is to return a representative *set of top-k answers from all the tuples that satisfy the user selection condition. For example, if a user is searching for cars and we can only display five results, we wish to return cars from five different models, as opposed to returning cars from only one or two models. A key contribution of this paper is to formally define the notion of diversity, and to show that existing score based techniques commonly used in web applications are not sufficient to guarantee diversity. Another contribution of this paper is to develop novel and efficient query processing techniques that guarantee diversity. Our experimental results using Yahoo! Autos data show that our proposed techniques are scalable and efficient.*

1 Introduction

Online shopping is increasing in popularity due to the large inventory of listings available on the Web. Users can issue a search query through a combination of fielded forms and keywords, and only the most relevant search results are shown due to the limited "real-estate" on a Web page. An important but lesser-known concern in such applications is the ability to return a *diverse* set of results which best reflects the inventory of available listings. As an illustration, consider a user searching for used 2009 MotoPed scooters. If we only have space to show five results, we would rather show five different MotoPed models (e.g., MotoPed Zoom, MotoPed Putt, MotoPed Bang, MotoPed Zip and MotoPed Vroom) instead of showing cars from just one or two models. Similarly, if the user searches for 2009 MotoPed Zoom scooters, we would rather show 2009 MotoPed Zoom scooters in different colors rather than simply showing scooters of the same color. Other applications such as online auction sites and electronic stores also have similar requirements (e.g., showing diverse auction listings, cameras, etc.).

While there are several existing solutions to this problem, they are either inefficient or do not work in all situations. For instance, the simplest solution is to obtain all the query results and then pick a diverse subset from these results. A more scalable variant of this method is commonly used in web search engines: in order to show k results to the user, first retrieve $c \times k$ results (for some $c > 1$) and then pick a diverse subset from these

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*An earlier version of this work appeared in *Proceedings of the 24th International Conference on Data Engineering*, ICDE 2008, April 7-12, 2008, Cancun, Mexico, pp. 228-236.

results [3, 11, 12]. However, while this method works well in web search where there are few duplicate or near-duplicate documents, it does not work as well for structured listings since there are many more duplicates. For instance, it is not uncommon to have hundreds of cars of a given model in a regional dealership, or thousands of cameras of a given model in a large online store. Thus, c would have to be of the order of 1000s or 10000s, which is clearly inefficient and furthermore, does not guarantee diverse results.

Another commonly used method is to issue multiple queries to obtain diverse results. For instance, if a user searches for purple MotoPed scooters, this method would issue a query for purple MotoPed Zooms, another for purple MotoPed Putts, and so on. While this method guarantees diverse results, it is inefficient for two reasons: it issues multiple queries, which hurts performance, and many of these queries may return empty results (e.g., if there are no purple MotoPed Zooms)

A final method that is sometimes used is to retrieve only a sample of the query results (e.g., using techniques proposed in [9]) and then pick a diverse subset from the sample. However, this method often misses rare but important listings that are missed in the sample.

To address the above limitations, we initiate a formal study of the diversity problem in search of methods that are scalable, efficient and guaranteed to produce diverse results. Towards this goal, we first present a formal definition of diversity, including both unscored and scored variants, that can be used to evaluate the correctness of various methods. We then explore whether we can use "off-the-shelf" technology to implement diversity efficiently and correctly. Specifically, we explore whether we can use optimized Information Retrieval (IR) engines with score-based pruning to implement diversity, by viewing diversity as a form of score. Unfortunately, it turns out that the answer is no — we prove that no possible assignment of static or query-dependent scores to items can be used to implement diversity in an off-the-shelf IR engine (although there is an open conjecture as to whether we can implement diversity using a combination of static and query-dependent scores).

We thus devise evaluation algorithms that implement diversity *inside the database/IR engine*. Our algorithms use an inverted list index that contains item ids encoded using Dewey identifiers [6]. The Dewey encoding captures the notion of *distinct values* from which we need a representative subset in the final query result. We first develop a one-pass algorithm that produces k diverse answers with a single scan over the inverted lists. The key idea of our algorithm is to explore a bounded number of answers within the same distinct value and use B+-trees to skip over similar answers. Although this algorithm is optimal when we are allowed only a single pass over the data, it can be improved when we are allowed to make a small number of probes into the data. We present an improved algorithm that is allowed to probe the set of answers within the same distinct value iteratively. The algorithm uses just a small number of probes — at most $2k$. Our algorithms are provably correct, they can support both unscored and score versions of diversity, and they can also support query relaxation Our experiments show that they are scalable and efficient. In summary, the main contributions of this paper are

- A formal definition of diversity and a proof that "off-the-shelf" IR engines cannot be used to implement diversity (Section 2)
- Efficient one-pass and probing algorithms for implementing diversity (Section 3)
- Experimental evaluation using Yahoo! Autos data (Section 4)

2 Diversity Definition and Impossibility Results

We formally define the notion of diversity and present some impossibility results for providing diversity using off-the-shelf IR systems.

Data and Query Model. We assume that the queried items are stored as tuples in a relation R. A query Q on a relation R is defined as a conjunction or disjunction of two kinds of predicates: *scalar predicates* of the form `att` $=$ `value` and *keyword predicates* of the form `att` $\ni$ `keywords` where `att` is an attribute of R and $\ni$ stands

for keyword containment. Given a relation R and a query Q, we use the notation $\text{RES}(R, Q)$ to denote the set of tuples in R that satisfy Q.

In many online applications, it is also often useful to allow tuples to have scores. One natural case is in the presence of keyword search queries, e.g., using scoring techniques such as TF-IDF [10]. Another case is in the context of "soft matches," where we give a weight to tuples so long as they satisfy *some* of the predicates in a given query (e.g., see [2]). We use the notation $score(t, Q)$ to denote the score of a tuple t that is produced as a result of evaluating a query Q.

Id	Make	Model	Color	Year	Description
1	MotoPed	Zoom	Green	2009	Low miles
2	MotoPed	Zoom	Blue	2009	Low miles
3	MotoPed	Zoom	Red	2009	Low miles
4	MotoPed	Zoom	Black	2009	Low miles
5	MotoPed	Zoom	Black	2008	Low price
6	MotoPed	Putt	Blue	2009	Best price
7	MotoPed	Putt	Red	2008	Good miles
8	MotoPed	Bang	Green	2009	Rare
9	MotoPed	Bang	Green	2008	Good miles
10	MotoPed	Zip	Red	2009	Fun car
11	MotoPed	Zip	Orange	2008	Good miles
12	Skoot	Hawk	Tan	2009	Low miles
13	Skoot	Raptor	Black	2009	Low miles
14	Skoot	Falcon	Blue	2009	Low miles
15	Skoot	Eagle	Blue	2009	Low miles

(a)

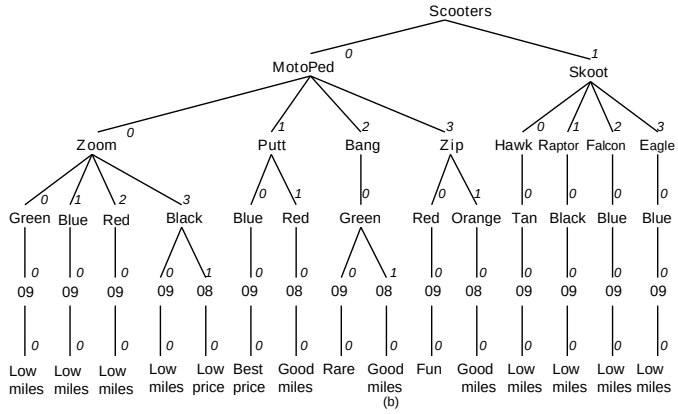


Figure 1: Example Database and Dewey Tree Representation

Diversity Definition. Consider the database of Figure 1(a). If the user issues a query for all cars and we have room for 3 results, then clearly, we should show at least one Skoot scooter and one MotoPed. If the user issues a query `Make = Skoot`, then we would show 3 different models of Skoot scooters. In general, there is a priority ordering of attributes: Make is more important than model, which is more important than say, color. This ordering is domain specific, and can be modified to suit the context. We define it below.

Definition 1: Diversity Ordering. A diversity ordering of a relation R with attributes A, denoted by $\prec_R$, is a total ordering of the attributes in A.

In our example, `Make` $\prec$ `Model` $\prec$ `Color` $\prec$ `Year` $\prec$ `Description` $\prec$ `Id`. (We ignore the suffix in $\prec_R$ when it is clear from the context.)

Given a diversity ordering, we can define a similarity measure between pairs of items, denoted $\text{SIM}(x, y)$, with the goal of finding a result set S whose items are least similar to each other (and hence most diverse); i.e., we wish to find a result set that minimizes $\sum_{x,y \in S} \text{SIM}(x, y)$.

With an eye toward our ultimate goal, let us take a very simple similarity function: $\text{SIM}(x, y) = 1$ if x and y agree on the highest priority attribute, and 0 otherwise. It is not hard to check that by using this similarity measure, minimizing the all-pairs sum of similarities in Skoots and MotoPeds (within one), so long as there are enough Skoots and MotoPeds to display.

However, we need to diversify not just on the top level, but on lower levels as well. If ρ is a prefix and S is a set, then denote $S_\rho = \{x \in S : \ \rho \text{ is a prefix of } x\ \}$. Then, if ρ is a prefix of length ℓ, define $\text{SIM}_\rho(x, y) = 1$ if x, y agree on their $(\ell + 1)$st attribute, and 0 otherwise. Again thinking of our example database, notice that if ρ = `MotoPed Zoom`, then minimizing $\sum_{x,y \in S_\rho} \text{SIM}_\rho(x, y)$ guarantees that we will not display two `Black` Zooms before displaying the `Green`, `Blue`, and `Red` ones, so long as they all satisfy a given query.

We are now almost ready for our main definition. Let $\mathcal{R}_k(R, Q)$ denote the set of all subsets of $\text{RES}(R, Q)$ of size k. For convenience, we will suppress the R, Q when it is clear from context.

Definition 2: Diversity. Given a relation R, a diversity ordering $\prec_R$, and a query Q, let $\mathcal{R}_k$ be defined as above. Let ρ be a prefix consistent with $\prec_R$. We say set $S \in \mathcal{R}_k$ is *diverse with respect to* ρ if $\sum_{x,y \in S_\rho} \text{SIM}_\rho(x, y)$ is minimized, over all sets $T \in \mathcal{R}_k$ such that $|T_\rho| = |S_\rho|$.

We say set $S \in \mathcal{R}_k$ is a *diverse result set (for* $\prec_R$*)* if S is diverse with respect to every prefix (for $\prec_R$).

Our definition for scored diversity is analogous. We define $\mathcal{R}_k^{score}$ to be the collection of all result sets of size k that have the largest score possible. Then scored diversity is defined as in Definition 2, with $\mathcal{R}_k$ replaced by $\mathcal{R}_k^{score}$. It can be shown that a diverse set of size k always exists, for both scored and unscored diversity.

Impossibility Results. We define Inverted-List Based IR Systems as follows: each unique attribute value/keyword contains the list of items that contain that attribute value/keyword. Each item in a list also has a score, which can either be a global score (e.g., PageRank) or a value/keyword -dependent score (e.g., TF-IDF). The items in each list are ordered by their score (typically done so that top-k queries can be handled efficiently). Given a query Q, we find the lists corresponding to the attribute values/keywords in Q, and aggregate the lists to find a set of k top-scored results. The score of an item that appears in multiple lists is aggregated from the per-list scores using a monotone aggregation function (efficient algorithms such as the Threshold Algorithm [5] require this). We have the following impossibility result.

Theorem 3: There is a database such that no Inverted-List Based IR System always produces an unscored diverse result set, even if we only consider non-null queries.

In fact, Figure 1(a) is such a database. Though strong, this result does not rule out every conceivable IR system. If we allow f to consider both static and value/keyword -dependent scores, then for every database, there is a set of scores and a monotonic function f such that the top-k list for every query is a diverse result set. However, the construction produces an f that essentially acts as a look-up table into the database, clearly an inefficient and infeasible solution. (It takes $O(n^2)$ space, where n is the number of items, to just write down this f.) We leave open the question of whether there is a "reasonable" aggregation function f that produces diverse result sets.

3 Algorithms

Data Structures and Processing. Given a diversity ordering, items can be arranged in a tree (which we refer to as a *Dewey tree*, as illustrated in Figure 1(b). Each item is then assigned a unique id, reminiscent of the Dewey encoding as done in XML query processing [6]. Each leaf value is obtained by traversing the tree top down and assigning a distinct integer to siblings. Since we only need to distinguish between siblings in the tree, we can re-initialize the numbering to 0 at each level. So, for example, the `MotoPed Bang Green 08 'Good miles'` has Dewey id 0.2.0.1.0.

Our basic functionality for processing lists follows standard IR systems. We use a WAND-like algorithm, which supports the calls `next` and `prev`. Since items are identified with and sorted by their Dewey ids, we can think of `next` as moving left-to-right through the leaves of the Dewey tree, returning the first item satisfying a given query. It also supports skipping to a new Dewey id; we will think of this as skipping to the beginning of a new subtree/branch. So for example, we could skip to find the first matching result in the `MotoPed Zip` subtree. Note that if no such matches exist in that subtree, `next` will continue on through the leaves until a result was found. (So, e.g., a result from the `Skoot Falcon` branch might be returned.) The method `prev` works in the same way, but moving *right-to-left*. For scored results, we modify `next` and `prev` to return the first matching item with some minimum score.

One-Pass Unscored Algorithm. In the one-pass algorithm, we make calls only to `next`, potentially skipping items, but never going back. Conceptually, this means that we encounter results from our query in left-to-right order of our tree nodes, as pictured in Figure 1(b). We maintain a *tentative result set*. This tentative result set is a valid result set, and further, is the most diverse result set possible, given the items we have encountered so far.

Consider a query Q requesting k items. Our tentative result set starts as the first k items matching our query. We then skip to the next dewey id that could possibly improve the diversity of our tentative result set. We repeat this until we have no more items matching the query.

The key steps of this tentative set maintenance are (1) deciding how far to skip, and (2) deciding which element to remove from the tentative result set to make room for the latest item found. We first describe (2). Let the *branch weight* of a branch in a tree to be the total number of leaves in the subtree rooted at (either) endpoint of the branch.

The algorithm for removal works as follows: Imagine the $k + 1$ items in the tentative result set together with the latest found item, arranged in a tree. Working from the root, always follow a branch with the highest branch weight (breaking ties by, say, choosing the rightmost among highest-weight branches). Eventually, we reach a leaf of this tree; remove the corresponding item from our set. Notice that the most diverse set will have "balanced" branches. The removal algorithm always seeks to remove a leaf from the most unbalanced branch.

The intuition behind the algorithm for (1) is to skip over any item that would result in immediate removal from the tentative result set We again imagine the tentative result set arranged in a tree. Working from the root, always examine the rightmost branch, i.e., the branch leading to the leaf corresponding to the most recently added item. If this branch is ever the highest-weight branch, then stop; skip to the first dewey id past this branch. On the other hand, if this branch ever has weight more than one less than the heaviest-weight branch, then stop; we cannot skip at all. Otherwise, when the branch has weight exactly one less than the heaviest-weight branch, continue down toward the leaf.

The key savings in this algorithm come from knowing when it is acceptable to skip ahead (i.e. jumping to a new branch in the Dewey tree). In the worst case, at most $k \ln^d(3k)$ calls to `next` are made to compute a top-k list with a Dewey tree of depth d. When the list of matching items is large, this can be a significant saving over the naive algorithm.

One-pass Scored Algorithm. The main difference with the unscored algorithm lies is what parts of the tree we can skip over. Each time we skip in the scored version, we can only skip to the *smaller* of the original skip Dewey id and the next item whose score is greater than or equal to the minimum score among items in our current tentative result set. Likewise, we can only remove items from the tentative result set whose scores are minimum. In all other respects, the algorithm is the same.

Unscored Probing Algorithm. The probing algorithm simply probes the Dewey tree, searching the branch that will be most beneficial in generating a diverse result set. Imagine the set of items arranged in a tree, as in Figure 1(b). The first probe searches the first branch of the tree, sweeping left-to-right. Note that since we are using a WAND-like implementation, this is accomplished by calling `next` on the all-zeros dewey id. We next wish to probe a new branch of the tree, in order to get the maximum diversity. However, rather than calling `next` on a new branch, we instead sweep *right-to-left* by calling `prev` on the all-infinity dewey id. (Thus, we start at the rightmost leaf and move left until finding a result.)

We continue in this manner, alternating between calling `next` on the branch *after* the branch of the most recently found item from a `next` call, and calling `prev` on the branch *before* the branch of the most recently found item from a `prev` call. If we continue in this way for k calls without returning two items from the same branch, we are done. However, if one of our calls returns an item from a previously used branch, then we have more work to do.

In this case, suppose we have found k' items so far. We divide $k - k'$ as evenly as possible among the

branches with found items. We then recurse on each subtree. For example, suppose $k = 6$ and we have found one MotoPed and one Skoot; searching more will repeat a branch, so we recurse. Here, we would assign 2 more items to the MotoPed branch and 2 more items to the Skoot branch. We continue our algorithm on each subtree. In our example, we would first process the subtree rooted at the MotoPed branch. Since the last call to the MotoPed branch was a `next` call, we begin by making a `prev` call, finding the rightmost item in the MotoPed subtree satisfying the query. If we are able to find enough items in the MotoPed subtree, then we proceed to the next subtree; otherwise, we redistribute the necessary items to other subtrees. For instance, suppose that we were unable to find any additional items in the MotoPed subtree. Then we would search for 4 additional matches in the Skoot subtree. (In our actual implementation, we ensure even distribution by recursing on each subtree is round-robin fashion.)

By making calls using both `next` and `prev`, we ensure that do not make many unnecessary probes. In fact, it is possible to show that we use at most $2k$ probes to find a k-item diverse result set.

Scored Probing Algorithm. The scored probing algorithm works in much the same way as the unscored version. First, we call WAND to obtain a top-k scored list. Let θ be the smallest score in the list. By the definition of scored diversity, all items with score greater than θ must be kept. We arrange these items into a tree, based on their dewey ids, ignoring items with score exactly θ. We now use this tree to make decisions on where to probe, in much the same way as in the unscored case. Each probe returns the next (or previous) item matching our query and having score θ. The details are omitted here.

4 Experiments

We compared the performance of five algorithms in the unscored and the scored cases. `MultQ` is based on rewriting the input query to multiple queries and merging their result to produce a diverse set. `Naive` evaluates a query and returns all its results. We do not include the time this algorithm takes to choose a diverse set of size k from its result. `Basic` returns the k first answers it finds without guaranteeing diversity. `OnePass` performs a single scan over the inverted lists (Section 3). Finally, `Probe` is the probing version (Section 3). We prefix each algorithm with a "U" for the unscored case. and with an "S" for their scored counterparts. Scoring is achieved with additional keyword predicates in the query. Recall that all of our diversity algorithms are *exact*. Hence, all results they return are maximally diverse.

4.1 Experimental Setup

We ran our experiments on an Intel machine with 2GB RAM. We used a real dataset containing car listings from Yahoo! Autos. The size of the original cars relation was varied from 100K to 1M rows with a default value set to 100K. Queries were synthetically generated using the following parameters: Number of cars $\in [10K - 100K]$ (default $50K$); Number of predicates/query $\in [1, 5]$ (default none); predicate selectivity $\in [0, 1]$ (default 0.5); $k \in [1, 100]$ (default 10). Query predicates are on car attributes and are picked at random. We report the total time for running a workload of 5000 different queries. In our implementation, the cars listings were stored in a main-memory table. We built an index generation module which generates an in-memory Dewey tree which stores the Dewey of each tuple in the base table. Index generation is done offline and is very fast (less than 5 minutes for 100K listings).

Varying Data Size: Figure 2(a) reports the response time of `UNaive`, `UBasic`, `UOnePass` and `UProbe`. `UOnePass` and `UProbe` have similar performance and are insensitive to increasing number of listings.

Varying Query Parameters: Figure 2(b) reports the response time of our unscored algorithms. Once again, `UOnePass` and `UProbe` have similar performance. The main two observations here are: (i) all our algorithms

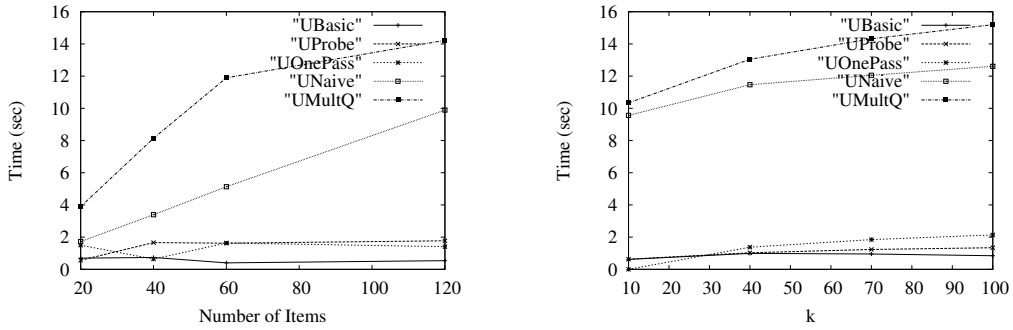


Figure 2: (a) Varying Data Size (Unscored), and (b) Varying k (Unscored)

outperforms the naive case which evaluates the full query and (ii) diversity incurs negligible overhead (over non-diverse `UBasic`) even for large values of k.

Figure 3(a) shows the response time of our unscored algorithms for different query selectivities. We grouped queries according to selectivity, measuring the average response time for each. `UOnePass` and `UProbe` remain stable with increasing selectivity, while `UNaive` is very sensitive since it retrieves all query results.

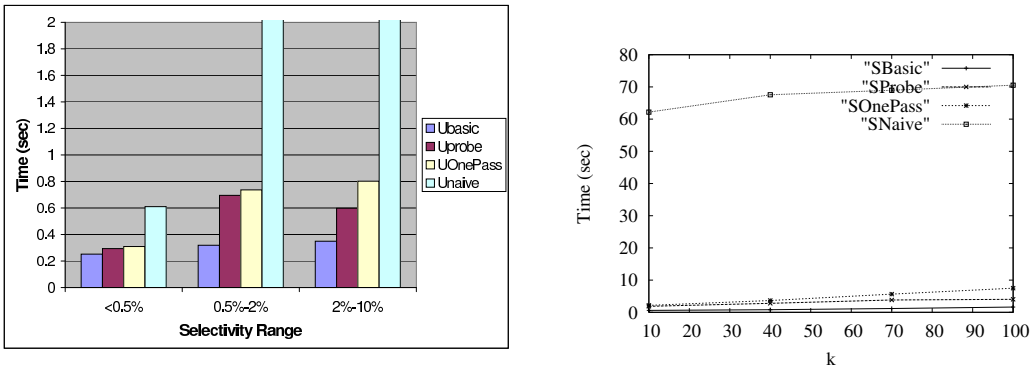


Figure 3: (a) Varying Q's Selectivity, and (b) Varying Data Size (scored)

Varying Query Parameters (scored): Figure 3(b) shows the response time of the scored algorithms as the number of results requested is varied. With increasing k, more listings have to be examined to return the k best ones. Thus, the response time of both `SOnePass` and `SProbe` increases linearly with k but as observed in the unscored case, the naive approach is outperformed. We note that varying query selectivity and data size is similar to the unscored case.

Experiments Summary: The naive approaches, `MultQ`, `UNaive`, `SNaive` are orders of magnitude slower than the other approaches. The most important finding is that returning diverse results using probing algorithms incurs negligible overhead (in the unscored case) and incurs very little overhead (in the scored case). Specifically, UProbe matches the performance of UBasic and SProbe comes very close to the performance of SBasic.

5 Related Work

The notion of diversity has been considered in many different contexts. Web search engines often enforce diversity over (unstructured) data results as a post-processing step [3, 11, 12]. Chen and Li [4] propose a notion of diversity over structured results which are post-processed and organized in a decision tree to help

users navigate them. In [8], the authors define the Précis of a query as a generalization of traditional query results. For example, if the query is "Jim Gray", its précis would be not just tuples containing these words, but also additional information related to it, such as publications, and colleagues. The précis is diverse enough to represent all information related to the query keywords. In this paper, we study a variant of diversity on structured data and combine it with top-k processing and efficient response times (no post-processing.)

In some online aggregation [7], aggregated group are displayed as they are computed and are updated at the same rate by index striding on different grouping column values. This idea is similar to our notion of equal representation for different values. However, in addition to considering scoring and top-k processing, we have a hierarchical notion of diversity, e.g., we first want diversity on `Make`, then on `Model`. In contrast, Index Striding is more "flat" in that it will simply consider (`Make`, `Model`) as a composite key, and list all possible (make, model) pairs, instead of showing only a few cars for each make.

6 Conclusion

We formalized diversity in structured search and proposed inverted-list algorithms. Our experiments showed that the algorithms are scalable and efficient. In particular, diversity can be implemented with little additional overhead when compared to traditional approaches.

A natural extension to our definition of diversity is producing weighted results by assigning weights to different attribute values. For instance, we may assign higher weights to MotoPeds and Skoots when compared to other scooter brands, so that the diverse results have more MotoPeds and Skoots. Another extension is exploring an alternative definition of diversity that provides a more symmetric treatment of diversity and score thereby ensuring diversity across different scores.

References

[1] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, J. Y. Zien. Efficient Query Evaluation Using a Two-Level Retrieval Process. CIKM 2003.

[2] N. Bruno, S. Chaudhuri, L. Gravano. Top-K Selection Queries Over Relational Databases: Mapping Strategies and Performance Evaluation. ACM Transactions on Database Systems (TODS), 27(2), 2002.

[3] J. Carbonell and J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. SIGIR 98.

[4] Z. Chen, T. Li. Addressing Diverse User Preferences in SQL-Query-Result Navigation. SIGMOD 2007.

[5] R. Fagin. Combining Fuzzy Information from Multiple Systems. PODS 1996.

[6] L. Guo, F. Shao, C. Botev, J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. SIGMOD 2003.

[7] J. M. Hellerstein, P. J. Haas, H. J. Wang. Online Aggregation. SIGMOD 1997.

[8] G. Koutrika, A. Simitsis, Y. Ioannidis. Précis: The Essence of a Query Answer. ICDE 2006.

[9] F. Olken. Random Sampling from Databases. PhD thesis, UC Berkely, 1993.

[10] G. Salton and M. McGill. Introduction to Modern Information Retrieval. McGraw-Hill, 1983.

[11] D. Xin, H. Cheng, X. Yan, J. Han. Extracting Redundancy-Aware Top-k Patterns. KDD 2006.

[12] C-N Ziegler, S.M. McNee, J.A. Konstan, and G. Lausen. Improving Recommendation Lists Through Topic Diversification. WWW 2005.

Diversity in Skylines

Yufei Tao
Department of Computer Science and Engineering
Chinese University of Hong Kong
Sha Tin, New Territories, Hong Kong
taoyf@cse.cuhk.edu.hk

Abstract

Given an integer k, a diverse skyline *contains the k skyline points that best describe the tradeoffs (among different dimensions) offered by the full skyline. This paper gives an overview of the latest results on this topic. Specifically, we first describe the state-of-the-art formulation of diverse skylines. Then, we explain several algorithms for finding a diverse skyline, where the objective is to save cost by avoiding the computation of the entire skyline. In particular, we will discuss a polynomial-time algorithm in 2D space that returns the exact result, the NP-hardness of the problem in dimensionality at least 3, and an approximate solution with good quality guarantees.*

1 Introduction

Given a set $\mathcal{D}$ of multidimensional points, the *skyline* [2] consists of the points that are not dominated by any other point. Specifically, a point p *dominates* another p' if the coordinate of p is smaller than or equal to that of p' on all dimensions, and strictly smaller on at least one dimension. Figure 1 shows a classical example with a set $\mathcal{D}$ of 13 points, each capturing two properties of a hotel: its distance to the beach (the horizontal coordinate), and price (the vertical coordinate). The skyline has 8 points p_1, p_2, ..., p_8.

Skyline retrieval has received considerable attention from the database community, resulting in a large number of interesting results as surveyed in Section 5. These research efforts reflect the crucial importance of skylines in practice. In particular, it is well-known [18] that there exists an inherent connection between skylines and top-1 queries. Specifically, given a preference function $f(p)$ which calculates a *score* for each point p, a *top*-1 query returns the data point with the lowest score. As long as function $f(\cdot)$ is *monotone*[1], the top-1 result is definitely in the skyline. Conversely, every skyline point is guaranteed to be the top-1 result for at least one preference function $f(\cdot)$.

The skyline operator is particularly useful in scenarios of multi-criteria optimization where it is difficult, or even impossible, to formulate a good preference function. For example, consider a tourist that wants to choose from the hotels in Figure 1 a good one offering a nice tradeoff between price and distance. S/he may not be sure about the relatively weighting of the two dimensions, or in general, whether the quality of a hotel should be assessed through a linear, quadratic, or other types of preference functions. In this case, it would be reasonable

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1]Namely, $f(p)$ grows as long as the coordinate of p along any dimension increases.

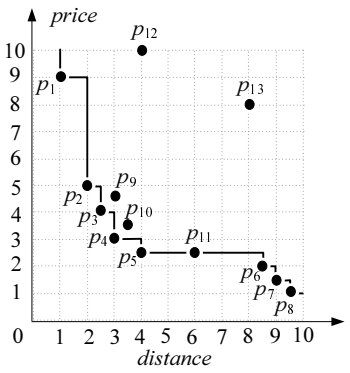


Figure 1: A skyline example

to return the skyline, so that the tourist can directly compare the tradeoffs offered by different skyline points. For example, as far as tradeoffs are concerned, the skyline points in Figure 1 can be divided into three subsets S_1, S_2, S_3:

- $S_1 = \{p_1\}$, which includes a hotel that is very close to the beach, but rather expensive;
- $S_2 = \{p_2, p_3, p_4, p_5\}$, where the hotels are farther away from the beach, but cheaper;
- $S_3 = \{p_6, p_7, p_8\}$, where the hotels are the cheapest, but far from the beach.

In this article, we discuss the problem of retrieving *diverse skylines*, which includes a small number k of skyline points that best describe the *representative* tradeoffs in the full skyline. For example, given $k = 3$, the representatives would be p_1, p_4, and p_7, each of which comes from a distinct subset illustrated above, providing a unique tradeoff. Diverse skylines are especially helpful in web-based recommendation systems such as the one in our previous hotel example. Skyline computation can be rather costly, particularly in high dimensional spaces. This necessitates a long waiting period before the entire skyline is delivered to the user, which may potentially incur negative user experience. A better approach is to return a few early skyline points representing the *contour* of the final skyline, and then, progressively refine the contour by reporting more skyline points. In this way, a user can understand the possible tradeoffs s/he may eventually get, well before the query finishes. Moreover, given such valuable information, the user may also notify the web server to stop fetching more skyline points offering uninteresting tradeoffs, thus significantly reducing the processing time. The importance of diverse skylines is further discussed in [16, 24].

Skyline diversity can be formulated in several ways [16, 24]. The state-of-the-art definition [24] is designed based on the intuition that, for every non-representative skyline point, there should be a nearby representative. In this paper, we will focus on that definition, and the corresponding computation algorithms. In 2D space, the problem can be settled in polynomial time by a dynamic programming algorithm. For dimensionality at least 3, the problem is NP-hard, but fortunately we show that there is a 2-approximate polynomial algorithm. Utilizing a multidimensional access method, our algorithm can quickly identify the k representatives without extracting the entire skyline. Furthermore, the algorithm is progressive, and does not require the user to specify the value of k. Instead, it continuously returns representatives that are guaranteed to be a 2-approximate solution *at any moment*, until either manually terminated or eventually producing the full skyline.

The rest of the paper is organized as follows. Section 2 clarifies the formulation of diverse skylines. Section 3 presents an algorithm for finding the optimal diverse skyline in 2D space. Section 4 tackles the problem of retrieving diverse skylines in dimensionality at least 3. Section 5 surveys the previous literature on skyline. Finally, Section 6 concludes the paper with a summary.

2 Formulation of Diverse Skylines

Let $\mathcal{D}$ be a set of d-dimensional points. We use $\mathcal{S}$ to denote the full skyline of $\mathcal{D}$. The quality of a diverse skyline can be evaluated with the concept of *representation error* of $\mathcal{K}$, denoted as $Er(\mathcal{K}, \mathcal{S})$. Intuitively, a Diverse skyline $\mathcal{K}$ is good if, for every non-representative skyline point $p \in \mathcal{S} - \mathcal{K}$, there is a representative in $\mathcal{K}$ close to p. Hence, $Er(\mathcal{K}, \mathcal{S})$ quantifies the representation quality as the maximum distance between a non-representative skyline point in $\mathcal{S} - \mathcal{K}$ and its nearest representative in $\mathcal{K}$, under Euclidean distance, or formally:

$$Er(\mathcal{K}, \mathcal{S}) = \max_{p \in \mathcal{S} - \mathcal{K}} \{\min_{p' \in \mathcal{K}} \|p, p'\|\}. \tag{18}$$

In the sequel, when the second parameter of function $Er(\cdot, \cdot)$ is the full skyline $\mathcal{S}$ of $\mathcal{D}$, we often abbreviate $Er(\mathcal{K}, \mathcal{S})$ as $Er(\mathcal{K})$. For example, in Figure 1, when $\mathcal{K} = \{p_3, p_4, p_5\}$, $Er(\mathcal{K}) = \|p_5, p_8\|$, i.e., p_8 is the point that is worst represented by $\mathcal{K}$. The following definition is the state-of-the-art formulation of diverse skylines [24]:

Definition 1: Let $\mathcal{D}$ be a multidimensional dataset and $\mathcal{S}$ its skyline. Given an integer k, the *diverse skyline* of $\mathcal{D}$ is a set $\mathcal{K}$ of k skyline points in $\mathcal{S}$ that minimizes $Er(\mathcal{K}, \mathcal{S})$ as calculated by Equation 18. Each point in $\mathcal{S}$ is a called a *representative*. □

In other words, the diverse skyline consists of k skyline points that achieve the lowest representation error. For example, in Figure 1, with $k = 3$ the diverse skyline is $\mathcal{K} = \{p_1, p_4, p_7\}$, whose $Er(\mathcal{K})$ equals $\|p_4, p_2\|$.

The diverse skyline is essentially the optimal solution of the *k-center problem* [10] on the full skyline $\mathcal{S}$. As a result, the diverse skyline also shares the properties of k-center results. One, particularly, is that the result is not sensitive to the densities of clusters. This is very important for capturing the *contour* of the skyline. Specifically, we do not want to allocate many representatives to a cluster simply because it has a large density. Instead, we would like to distribute the representatives evenly along the skyline, regardless of the densities of the underlying clusters. This is why Equation 18 is better than its sum-counterpart $\sum_{p \in \mathcal{S} - \mathcal{K}} \{\min_{p' \in \mathcal{K}} \|p, p'\|\}$. The latter tends to give more representatives to a dense cluster, because doing so may reduce the distances of a huge number of points to their nearest representatives, which may outweight the benefit of trying to reduce such distances of points in a faraway sparse cluster.

From now on, we will use the term *sub-skyline* to refer to any subset $\mathcal{K}$ of the full skyline $\mathcal{S}$. If $\mathcal{K}$ has k points, we say that it is *size-k*. The problem we study in this paper can be defined as:

Problem 1: Given an integer k, find an optimal size-k sub-skyline $\mathcal{K}$ that has the smallest representation error $Er(\mathcal{K}, \mathcal{S})$ given in Equation 18. □

Note that the optimal sub-skyline is the diverse skyline in Definition 1. Sometimes it is computationally intractable to find the optimal solution. In this case, we instead aim at computing a sub-skyline whose representation error is as low as possible.

3 The Two-dimensional Case

In this section, we give an algorithm for solving Problem 1 optimally in 2D space. We consider that the skyline $\mathcal{S}$ of dataset $\mathcal{D}$ has already been computed using an existing algorithm. Let m be the size of $\mathcal{S}$. Denote the skyline points in $\mathcal{S}$ as p_1, p_2, ..., p_m, sorted in ascending order of their x-coordinates.

We adopt the notation $\mathcal{S}_i$ to represent $\{p_1, p_2, ..., p_i\}$ where $i \leq m$. Specially, define $\mathcal{S}_0 = \emptyset$. Introduce a function $opt(i, t)$ to be the optimal size-t diverse skyline of $\mathcal{S}_i$, where $t \leq i$. Hence, the optimal size-k diverse

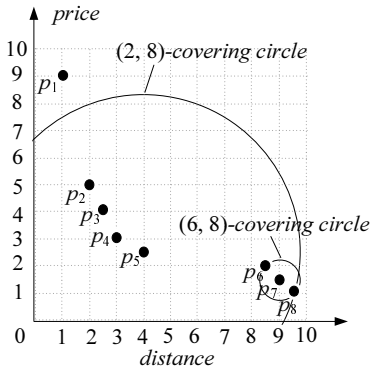


Figure 2: Covering circles

Algorithm *2D-opt* $(\mathcal{S}, k)$
Input: the skyline $\mathcal{S}$ of dataset $\mathcal{D}$ and an integer k
Output: the diverse skyline of $\mathcal{D}$
1. for each pair of (i, j) such that $1 \leq i \leq j \leq m$, derive $radius(i, j)$ and $center(i, j)$.
2. set $opt(i, 1) = \{center(1, i)\}$ and $optEr(i, 1) = radius(1, i)$ for each $1 \leq i \leq m$
3. for $t = 2$ to $k - 1$
4. for $i = t$ to m
5. compute $optEr(i, t)$ by Equation 19
6. compute $opt(i, t)$ by Equation 20
7. compute $optEr(k, m)$ and $opt(k, m)$ by Equations 19 and 20
8. return $opt(k, m)$

Figure 3: An optimal algorithm for computing 2D diverse skylines

skyline of $\mathcal{S}$ is essentially $opt(m, k)$. Let function $optEr(i, t)$ be the representation error of $opt(i, t)$ with respect to $\mathcal{S}_i$, or formally, $optEr(i, t) = Er(opt(i, t), \mathcal{S}_i)$, where $Er(\cdot, \cdot)$ is given in Equation 18.

For any $1 \leq i \leq j \leq m$, we use $radius(i, j)$ to denote the radius of the smallest circle that (i) covers points p_i, p_{i+1}, ..., p_j, and (ii) centers at one of these $j - i + 1$ points. Call the circle the (i, j)-*covering circle*, and denote its center as $center(i, j)$. Figure 2 shows the $(2, 8)$- and $(6, 8)$-covering circles, whose centers are p_5 and p_7, respectively. Hence, $center(2, 8) = p_5$ and $center(6, 8) = p_7$. There exists a recursive equation about $optEr(i, t)$ when $t \geq 2$:

$$optEr(i, t) = \min_{j=t}^{i-1} \{\max\{optEr(j - 1, t - 1), radius(j, i)\}\} \tag{19}$$

The above equation is based on the following rationale. Assume, without loss of generality, that the optimal size-t diverse skyline of $\mathcal{S}_i$ is $\{p_{j_1}, p_{j_2}, ..., p_{j_t}\}$ with $1 \leq j_1 < j_2 < ... < j_t \leq i$, i.e., p_{j_1},..., p_{j_t} are in ascending order of their x-coordinates. Let p_j be the first point (in ascending order of x-coordinates) in $\mathcal{S}_i$ that has p_{j_t} as its nearest representative. Then, $\{p_{j_1}, ..., p_{j_{t-1}}\}$ must be the optimal size-$(t - 1)$ diverse skyline of $\mathcal{S}_{j-1}$, and p_{j_t} must be $center(j, i)$.

Let v be the value of j where Equation 19 reaches its minimum; we have:

$$opt(i, t) = opt(v - 1, t - 1) \cup \{center(v, i)\} \tag{20}$$

Equations 19 and 20 point to a dynamic programming algorithm *2D-opt* in Figure 3 for computing $opt(k, m)$, i.e., the size-k diverse skyline of $\mathcal{D}$.

Time complexity. As explained shortly, Line 1 of *2D-opt* can be implemented in $O(m^2)$ time, where m is the size of the full skyline $\mathcal{S}$ of $\mathcal{D}$. Line 2 obviously requires $O(m)$ time. Lines 3-6 perform $k - 2$ iterations.

Each iteration evaluates Equations 19 and 20 m times respectively. Regardless of i and t, every evaluation of Equation 19 can be completed in $O(m)$ time, and that of Equation 20 in $O(k)$ time. Hence, Lines 3-6 altogether incur $O(m^2(k-2))$ cost. Finally, Line 7 requires $O(m)$ time. Therefore, the overall complexity of *2D-opt* is $O(m^2(k-2)+m)$. Note that this is much lower than the complexity of $O(|\mathcal{D}| \cdot \log m + m^2 \cdot k)$ (mentioned in Section 3) of computing the optimal 2D max-dominance skyline.

Computing covering circles. Next, we give an $O(m^2)$-time algorithm to find all the covering circles, i.e., $radius(i,j)$ and $center(i,j)$ for all $1 \leq i \leq j \leq m$. Note that one cannot hope to do any better because there are $\Omega(m^2)$ circles to decide. First, it is easy to see that

$$radius(i,j) = \min_{u=i}^{j} \{\max\{\|p_i, p_u\|, \|p_u, p_j\|\}\}. \tag{21}$$

Let $p_u.radius(i,j) = \max\{\|p_i, p_u\|, \|p_u, p_j\|\}$. Equation 21 can be re-written as:

$$radius(i,j) = \min_{u=i}^{j} p_u.radius(i,j), \tag{22}$$

Thus, $center(i,j)$ equals the p_u where the above equation reaches its minimum.

As u moves from i to j, the value of $p_u.radius(i,j)$ initially decreases and then increases, exhibiting a V-shape. The V-shape property offers an easy way, called *simple scan*, of finding $radius(i,j)$ and $center(i,j)$ as follows. We only need to inspect p_i, p_{i+1}, ..., p_j in this order, and stop once $p_u.radius(i,j)$ starts to increase, where u is the point being inspected. At this moment, we have just passed the minimum of Equation 22. Hence, we know $center(i,j) = p_{u-1}$ and $radius(i,j) = p_{u-1}.radius(i,j)$. A simple scan needs $O(j-i)$ time to decide a $radius(i,j)$. This, however, results in totally $O(m^3)$ time in determining all the covering circles, which makes the time complexity of our algorithm *2D-opt* $O(m^3)$ as well.

The time can be brought down to $O(m^2)$ with a method called *collective pass*. The main idea which obtains the (i,i)-, $(i,i+1)$-, ..., (i,m)-covering circles collectively in *one* scan from p_i to p_m in $O(m-i)$ time. Since a collective pass is needed for every $1 \leq i \leq m$, overall we spend $O(m^2)$ time. The details can be found in [24].

4 The Higher-dimensional Case

We proceed to study Problem 1 in dimensionality $d \geq 3$. As proved in [24], the problem is NP-hard for $d \geq 3$. Section 4.1 describes an algorithm for finding 2-approximate solution. Then, Section 4.2 discusses how to improve the efficiency of the approximate algorithm.

4.1 2-approximation

A 2-approximate solution $\mathcal{K}$ is a sub-skyline with k points whose representation error is at most twice that of $\mathcal{K}^*$. Namely, if the optimal diverse skyline is $\mathcal{K}^*$, then $Er(\mathcal{K}, \mathcal{S}) \leq 2 \cdot Er(\mathcal{K}^*, \mathcal{S})$, where $Er(\cdot,\cdot)$ is given in Equation 18.

Such a $\mathcal{K}$ can be found by a standard greedy algorithm [9] for the k-center problem. Specifically, first we retrieve the skyline $\mathcal{S}$ of $\mathcal{D}$ using any existing skyline algorithm, and initiate a $\mathcal{K}$ containing an arbitrary point in $\mathcal{S}$. Given a point p, define its *representative distance rep-dist*$(p, \mathcal{K})$ as the distance between p and its closest representative, or formally:

$$\textit{rep-dist}(p, \mathcal{K}) = \min_{p' \in \mathcal{K}} \|p, p'\|. \tag{23}$$

Then, we repeat the following $k-1$ times to create the final $\mathcal{K}$: add to $\mathcal{K}$ the point in $\mathcal{S} - \mathcal{K}$ with the largest representative distance. Note that as the content of $\mathcal{K}$ expands, the representative distance of a point may vary

as well, because its nearest representative may change to the one most recently added. We refer to this solution as *naive-greedy*. It guarantees a 2-approximate solution, as is established directly by the analysis of [9].

As an example, consider the dataset in Figure 1, where $\mathcal{S} = \{p_1, p_2, ..., p_8\}$. Assume $k = 3$ and that p_4 is the first point inserted to $\mathcal{K}$. Among the other skyline points, p_8 is farthest from p_4, and thus, is the second point added to $\mathcal{K}$. Now, p_1 has the greatest representative distance in $\mathcal{S} - \mathcal{K}$, and hence, enters $\mathcal{K}$. Thus, the final result is $\mathcal{K} = \{p_4, p_8, p_1\}$.

Naive-greedy has several drawbacks. First, it incurs large I/O overhead because it requires retrieving the entire skyline $\mathcal{S}$. Since we aim at returning only $k \ll |\mathcal{S}|$ points, ideally we should be able to do so by accessing only a fraction of $\mathcal{S}$, thus saving considerable cost. Second, it lacks progressiveness, because no result can be output until the full skyline has been computed. In the next section, we outline an alternative algorithm called *I-greedy* which overcomes both drawbacks of *naive-greedy*.

4.2 I-greedy

I-greedy assumes a multidimensional index (such as an R-tree [1]) on the dataset $\mathcal{D}$. It can be regarded as an efficient implementation of the *naive-greedy* algorithm explained in the previous subsection. Specifically, it returns the same set $\mathcal{K}$ of representatives as *naive-greedy*. Therefore, *I-greedy* also has the same approximation ratio as *naive-greedy*.

Recall that, after the first representative, *naive-greedy* repetitively adds to $\mathcal{K}$ the point in $\mathcal{S} - \mathcal{K}$ with the maximum representative distance given by Equation 23. Finding this point is analogous to *farthest neighbor search*, using Equation 23 as the distance function. However, remember that not every point in dataset $\mathcal{D}$ can be considered as a candidate result. Instead, we consider only $\mathcal{S} - \mathcal{K}$, i.e., the set of skyline points still outside $\mathcal{K}$.

The *best-first* algorithm [11] is a well-known efficient algorithm for farthest neighbor search[2]. To apply *best-first*, we must define the notion of *max-rep-dist*. Specifically, given an MBR R in the R-tree, its max-rep-dist, *max-rep-dist*$(R, \mathcal{K})$, is a value which upper bounds the representative distance *rep-dist*$(p, \mathcal{K})$ of any potential skyline point p in the subtree of R. *max-rep-dist*$(R, \mathcal{K})$ can be easily computed, as shown in [24]. Let us refer to both *max-rep-dist*$(R, \mathcal{K})$ and *rep-dist*$(p, \mathcal{K})$ as the *key* of R and p, respectively. *Best-first* visits the intermediate and leaf entries of the whole R-tree in descending order of their keys. Hence, the first leaf entry visited is guaranteed to be the point in $\mathcal{D}$ with the largest representative distance.

Let p be the first data point returned by *best-first*. We cannot report p as a representative, unless we are sure that it is a skyline point. Whether p is a skyline point can be resolved using an *empty test*. Such a test checks if there is any data point inside the *anti-dominant region* of p, which is the rectangle having p and the origin of the data space as two opposite corners. If the test returns "empty", p is a skyline point; otherwise, it is not. In any case, we continue the execution of *best-first* to retrieve the point with the next largest max-rep-dist, and repeat the above process, until enough representatives have been reported.

Best-first may still entail expensive I/O cost, as it performs numerous empty tests, each of which may need to visit many nodes whose MBRs intersect the anti-dominant region of a point. A better algorithm should therefore avoid empty tests as much as possible. *I-greedy* achieves this goal with two main ideas. First, it maintains a *conservative skyline* based on the intermediate and leaf entries already encountered. Second, it adopts an access order different from *best-first*, which *totally* eliminates empty tests. The details can be found in [24].

5 Related Work

The first work on skylines in the database area is due to Borzsonyi et al. [2]. Since then, many algorithms have been developed for computing skylines efficiently, for example, *Bitmap* [23], *NN* [14], *BBS* [18], *Lattice* [17],

[2]Precisely speaking, *best-first* is originally designed for nearest neighbor search [11]. However, its adaptation to farthest neighbor search is trivial.

to name just a few. These algorithms focus on the original data space, while considerable efforts have also been made to retrieve skylines in subspaces, such as *subsky* [25], *skycube* [20], and so on. Besides traditional centralized DBMS, skyline search has also been studied in distributed systems [7, 26], streams [21], p2p networks [27], partially-ordered domains [3], etc. The concept of skyline has numerous useful variations. One example is the *diverse skyline* studied in this paper, and this notion is originally introduced in [16]. Other examples include k*-dominant skyline* [4], *spatial skyline* [22], *probabilistic skyline* [19], *reverse skyline* [8, 15], *privacy skyline* [6], *approximately dominating representatives* [13], retrieving the points with the highest *subspace skyline frequencies* [5], and so on.

The *best-first* algorithm mentioned in Section 4.2 is proposed by Hjaltason and Samet [11] for solving nearest/farthest neighbor search. It is I/O optimal in the sense that, given the same R-tree, no other algorithm is able to answer the same query by accessing fewer nodes. The k*-center* problem, which underlines diverse skylines, is a classical problem that can be defined on any distance metric. Without knowing the metric, it is NP-hard to find a solution with approximation ratio $2-\varepsilon$ for any positive ε [12]. The greedy algorithm described in Section 4.1 provides a 2-approximate solution for any distance metric satisfying the triangle inequality [9].

6 Conclusions

The skyline of a dataset may have a large number of points. Returning all of them may make it difficult for a user to understand the possible tradeoffs offered by the skyline. A better approach is to present only a few representative points that reflect the contour of the entire skyline, so that the user may request only the skyline points in a specific part of the contour that looks interesting. In this article, we described the formulation of such a diverse skyline, and discussed its computation algorithms. In 2D space, the problem can be solved optimally in low-degree polynomial time. In higher dimensional spaces where computing the optimal solution is NP-hard, it is possible to return a 2-approximate solution efficiently.

References

[1] N. Beckmann, H. Kriegel, R. Schneider, B. Seeger, *The R***-Tree: An Efficient and Robust Access Method for Points and Rectangles*, in Proc. of ACM Management of Data (SIGMOD), 322–331, 1990.

[2] S. Borzsonyi, D. Kossmann, K. Stocker, *The Skyline Operator*, in Proc. of Int'l Conference on Data Engineering (ICDE), 421–430, 2001.

[3] C. Y. Chan, P.-K. Eng, K.-L. Tan, *Stratified Computation of Skylines with Partially-Ordered Domains*, in Proc. of ACM Management of Data (SIGMOD), 203–214, 2005.

[4] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, Z. Zhang, *Finding k-dominant skylines in high dimensional space*, in Proc. of ACM Management of Data (SIGMOD), 503–514, 2006.

[5] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, Z. Zhang, *On High Dimensional Skylines*, in Proc. of Extending Database Technology (EDBT), 478–495, 2006.

[6] B.-C. Chen, R. Ramakrishnan, K. LeFevre, *Privacy Skyline: Privacy with Multidimensional Adversarial Knowledge*, in Proc. of Very Large Data Bases (VLDB), 770–781, 2007.

[7] B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai, Y. Zhou, *Parallel Distributed Processing of Constrained Skyline Queries by Filtering*, in Proc. of Int'l Conference on Data Engineering (ICDE), 546–555, 2008.

[8] E. Dellis, B. Seeger, *Efficient Computation of Reverse Skyline Queries*, in Proc. of Very Large Data Bases (VLDB), 291–302, 2007.

[9] T. Feder, D. Greene, *Optimal algorithms for approximate clustering*, in Proc. of ACM Symposium on Theory of Computing (STOC), 434–444, 1988.

[10] T. F. Gonzalez, *Clustering to Minimize the Maximum Intercluster Distance*, in Theor. Comput. Sci., (38), 293–306, 1985.

[11] G. R. Hjaltason, H. Samet, *Distance Browsing in Spatial Databases.*, in ACM Transactions on Database Systems (TODS), (24):2, 265–318, 1999.

[12] W. L. Hsu, G. L. Nemhauser, *Easy and hard bottleneck location problems*, in Disc. Appl. Math. (1), 209–216, 1979.

[13] V. Koltun, C. H. Papadimitriou, *Approximately Dominating Representatives*, in Proc. of Int'l Conference on Database Theory (ICDT), 204–214, 2005.

[14] D. Kossmann, F. Ramsak, S. Rost, *Shooting Stars in the Sky: An Online Algorithm for Skyline Queries*, in Proc. of Very Large Data Bases (VLDB), 275–286, 2002.

[15] X. Lian, L. Chen, *Monochromatic and bichromatic reverse skyline search over uncertain databases*, in Proc. of ACM Management of Data (SIGMOD), 213–226, 2008.

[16] X. Lin, Y. Yuan, Q. Zhang, Y. Zhang, *Selecting Stars: The k Most Representative Skyline Operator*, in Proc. of Int'l Conference on Data Engineering (ICDE), 86–95, 2007.

[17] M. Morse, J. M. Patel, H. V. Jagadish, *Efficient Skyline Computation over Low-Cardinality Domains*, in Proc. of Very Large Data Bases (VLDB), 267–278, 2007.

[18] D. Papadias, Y. Tao, G. Fu, B. Seeger, *An Optimal and Progressive Algorithm for Skyline Queries*, in Proc. of ACM Management of Data (SIGMOD), 467–478, 2003.

[19] J. Pei, B. Jiang, X. Lin, Y. Yuan, *Probabilistic Skylines on Uncertain Data*, in Proc. of Very Large Data Bases (VLDB), 15–26, 2007.

[20] J. Pei, W. Jin, M. Ester, Y. Tao, *Catching the Best Views of Skyline: A Semantic Approach Based on Decisive Subspaces*, in Proc. of Very Large Data Bases (VLDB), 253–264, 2005.

[21] N. Sarkas, G. Das, N. Koudas, A. K. H. Tung, *Categorical skylines for streaming data*, in Proc. of ACM Management of Data (SIGMOD), 239–250, 2008.

[22] M. Sharifzadeh, C. Shahabi, *The Spatial Skyline Queries*, in Proc. of Very Large Data Bases (VLDB), 751–762, 2006.

[23] K.-L. Tan, P.-K. Eng, B. C. Ooi, *Efficient Progressive Skyline Computation*, in Proc. of Very Large Data Bases (VLDB), 301–310, 2001.

[24] Y. Tao, L. Ding, X. Lin, J. Pei, *Distance-Based Representative Skyline*, in Proc. of Int'l Conference on Data Engineering (ICDE), 892–903, 2009.

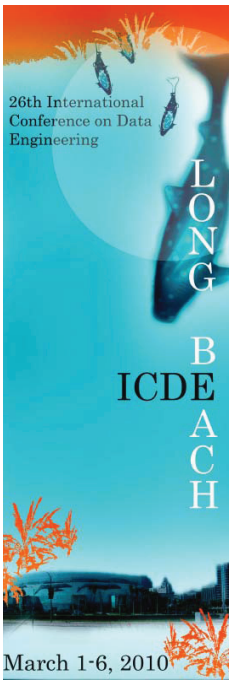
[25] Y. Tao, X. Xiao, J. Pei, *SUBSKY: Efficient Computation of Skylines in Subspaces*, in Proc. of Int'l Conference on Data Engineering (ICDE), 65–65, 2006.

[26] A. Vlachou, C. Doulkeridis, Y. Kotidis, *Angle-based space partitioning for efficient parallel skyline computation*, in Proc. of ACM Management of Data (SIGMOD), 227–238, 2008.

[27] S. Wang, B. C. Ooi, A. K. H. Tung, L. Xu, *Efficient Skyline Query Processing on Peer-to-Peer Networks*, in Proc. of Int'l Conference on Data Engineering (ICDE), 1126–1135, 2007.

26th IEEE International Conference on Data Engineering (ICDE 2010)

March 1 – 6, 2010, Long Beach, California, USA



CALL FOR PARTICIPATION

Data Engineering deals with the use of engineering techniques and methodologies in the design, development and assessment of information systems for different computing platforms and application environments. The **26th IEEE International Conference on Data Engineering** will continue in its tradition of being a premier forum for presentation of research results and advanced data-intensive applications and discussion of issues on data and knowledge engineering. The mission of the conference is to share research solutions to problems of today's information society and to identify new issues and directions for future research and development work.

Program Highlights:

- 3 ***Keynotes***
- 5 ***Advanced Technology Seminars***
- 8 ***Workshops***
- 69 ***Research Full Papers***
- 41 ***Research Short Papers***
- 9 ***Industrial Papers***
- 29 ***Demos***
- 2 ***Panels***

Keynotes:

- Donald Kossmann (Swiss Federal Institute of Technology, Zurich)
- Jeffrey Naughton (University of Wisconsin – Madison)
- Richard Winter (WinterCorp Consulting Services, Cambridge, MA)

Workshops:

- ***Ranking in Databases*** (DBRank)
- ***Information & Software as Services*** (WISS)
- ***Self Managing Database Systems*** (SMDB)
- ***Management and Mining of Uncertain Data*** (MOUND)
- ***New Trends in Information Integration*** (NTII)
- ***Data Engineering meets the Semantic Web*** (DeSWeb)
- ***Modeling, Managing and Mining of Evolving Social Networks*** (M3SN)
- ***PhD Workshop***

Advanced Technology Seminars:

- ***Anonymized Data: Generation, Models, Usage***
  Graham Cormode (AT&T Labs Research) and Divesh Srivastava (AT&T Labs Research)
- ***Database as a Service (DBaaS)***
  Wolfgang Lehner (TU Dresden) and Kai-Uwe Sattler (TU Ilmenau, Germany)
- ***Representation, Composition and Application of Preferences in Databases***
  Georgia Koutrika (Stanford University), Evi Pitoura and Kostas Stefanidis (Univ. of Ioannina, Greece)
- ***Privacy in Data Publishing***
  Johannes Gehrke (Cornell), Daniel Kifer (Penn State) and Ashwin Machanavajjhala (Yahoo! Research)
- ***Techniques for Efficiently Searching in Spatial, Temporal, Spatio-temporal, and Multimedia Databases***
  Matthias Renz, Hans-Peter Kriegel and Peer Kröger (Ludwig-Maximilians-University Munich, Germany)

Venue: Long Beach, CA

After many years, ICDE returns to the Los Angeles area, where it started in 1984. The conference will be held at the **Hyatt Regency Hotel** in Long Beach, California. Long Beach offers many attractions including the ***Queen Mary*** ocean liner and the ***Aquarium of the Pacific***. A large variety of sightseeing opportunities are in close proximity, ranging from ***The Getty Center*** to the ***Disney Concert Hall***, as well as attractions like ***Hollywood***, ***Universal Studios*** and ***Disneyland***.
The conference early registration deadline is **January 15th, 2010**. For more information, visit: **www.icde2010.org**

36th International Conference on Very Large Data Bases (VLDB2010)

13 to 17 September 2010, Grand Copthorne Waterfront Hotel, Singapore,

http://vldb2010.org

Submission Site: https://cmt.research.microsoft.com/VLDB2010/Default.aspx

VLDB 2010 calls for outstanding research papers as well as proposals for demonstrations. Tutorial and Panel proposals on all topics that will be of particular interest for the community are welcome. VLDB 2010 also strongly encourages the submission of workshop proposals on challenging topics in areas related to the VLDB focus. VLDB 2010 is organized into four tracks.

Core Database Technology Track

The Core Database Technology Track will evaluate papers on technologies intended to be incorporated within the database system itself. The topics of interest to this track include (but are not limited to):

Active Databases
Benchmarking, Performance & Evaluation
Concurrency Control, Recovery and Transaction Management
Data Models and Languages
Database Administration and Manageability
Database Indexing and Search
Databases on Modern Hardware
Embedded and Mobile Databases
Engine-based Views, Replication, and Caching
Fuzzy, Probabilistic, and Approximate Data
Native Semi-Structured Data and XML
Parallel, Distributed, and Grid Databases
Private and Secure Databases
Query Processing and Optimization
Real-Time Databases
Reliable and Robust Databases
Scientific Databases
Spatial, Temporal & Multimedia Databases
Stream Databases

Infrastructure for Information Systems Track

The Information Infrastructure Track covers all aspects of data management not implemented within a conventional database engine. The topics covered by this track include (but are not limited to):

Content Delivery Networks
Data Design, Evolution and Migration
Data Extraction
Data Management in Computational Science
Data Mining
Data Quality and Semantics
Heterogeneous and Federated DBMS (Interoperability)
Information Filtering and Dissemination
Information Integration and Retrieval
Meta-data Management
Middleware Platforms for Data Management
Mobile Data Management
Novel/Advanced Applications
On-Line Analytic Processing
P2P and Networked Data Management
Profile-based Data Management
Provenance Management
Scientific Databases
Sensor Networks
User Interfaces and Visualization
Web Replication and Caching
Web Services and Web Service Composition
XML Middleware Platforms
Social Systems and Recommendations

Industrial Applications and Experience Track

The Industrial, Applications, and Experience Track covers innovative commercial database implementations, novel applications of database technology, and experience in applying recent research advances to practical situations, in any of the following example areas (or, in other areas where data management is important):

Adapting DB Technology to Industrial Settings and Requirements
Application Areas (Government, Finance, Humanities, Telecommunications, Home and Personal Computing, ...)
Bio-Informatics/Life Sciences
Business Process Engineering and Execution Support
Data Management for Developing Countries
Digital Libraries/Document Management
Electronic Commerce
Engineering Information Systems
Enterprise Data Management
Enterprise Resource Planning
Environmental Management
Experiences in Using DB Technology
Geographic Information Systems
Industrial-Strength Systems based on DB Technology
Mobile Computing
Self-Managing Systems
System Design and Implementation using DB Technology

The Experiments and Analyses Track

Database management has been an active area of research for several decades. This special topic aims to meet needs for consolidation of a maturing research area by providing a prestigious forum for in-depth analytical or empirical studies and comparisons of existing techniques. The expected contribution of an Experiments and Analyses (E&A) paper is new, independent, comprehensive and reproducible evaluations and comparisons of existing data management techniques. Thus, the intended contribution of an E&A paper is not a new algorithm or technique but rather further insight into the state-of-the-art by means of careful, systematic, and scientific evaluation. Comparisons of algorithmic techniques must either use best-effort re-implementations based on the original papers, or use existing implementations from the original authors, if publicly available. Authors should discuss and validate substantial new results with authors of the original methods before submission.

IMPORTANT DATES

Research Papers
1 March 2010 (5:00pm GMT, 10:00am PDT) Abstracts submission
9 March 2010 (5:00pm GMT, 10:00am PDT) Paper submission
17 May - 20 May 2010 Author feedback period
12 June 2010 Notification
11 July 2010 Camera-ready paper due

PhD Workshop Papers
10 April 2010 (5:00pm GMT, 10:00am PDT) Paper submission
12 June 2010 Notification
11 July 2010 Camera-ready paper due

Demonstration Proposals
22 March 2010 (5:00pm GMT, 10:00am PDT) Proposal submission
12 June 2010 Notification
11 July 2010 Camera-ready paper due

Workshop Proposals
31 January 2010 (5:00pm GMT, 10:00am PDT) Proposal submission
31 March 2010 Notification

Tutorial proposals
2 April 2010 (5:00pm GMT, 10:00am PDT) Proposal submission
12 June 2010 Notification

Panel Proposals
2 April 2010 (5:00pm GMT, 10:00am PDT) Proposal submission
19 June 2010 Notification

Conference
13 and 17 September 2010 Workshops
14 to 16 September 2010 Main Conference

Call for Papers
First ACM Symposium on Cloud Computing (SOCC)
June 10 & 11, 2010, Indianapolis
https://research.microsoft.com/socc2010

The *ACM Symposium on Cloud Computing* 2010 (*ACM SOCC 2010*) is the first in a new series of symposia with the aim of bringing together researchers, developers, users, and practitioners interested in cloud computing. This series is co-sponsored by the ACM Special Interest Groups on Management of Data (ACM SIGMOD) and on Operating Systems (ACM SIGOPS). ACM SOCC will be held in conjunction with ACM SIGMOD and ACM SOSP Conferences in alternate years, starting with ACM SIGMOD in 2010.

The scope of SOCC Symposia will be broad and will encompass diverse systems topics such as software as a service, virtualization, and scalable cloud data services. Many facets of systems and data management issues will need to be revisited in the context of cloud computing. Suggested topics for paper submissions include but are not limited to:

- Administration and Manageability
- Data Privacy
- Data Services Architectures
- Distributed and Parallel Query Processing
- Energy Management
- Geographic Distribution
- Grid Computing
- High Availability and Reliability
- Infrastructure Technologies
- Large Scale Cloud Applications
- Multi-tenancy
- Provisioning and Metering
- Resource management and Performance
- Scientific Data Management
- Security of Services
- Service Level Agreements
- Storage Architectures
- Transactional Models
- Virtualization Technologies

Submission: Authors are invited to submit original papers that are not being considered for publication in any other forum. Manuscripts should be submitted in PDF format and formatted using the ACM camera-ready templates available at http://www.acm.org/sigs/pubs/proceed/template.html. The symposium website http://research.microsoft.com/socc provides addition details on the submission procedure. A submission to the symposium may be one of the following three types: (a) *Research papers:* We seek papers on original research work in the broad area of cloud computing. The length of research papers is limited to twelve pages. (b) *Industrial papers:* The symposium will also be a forum for high quality industrial presentations on innovative cloud computing platforms, applications and experiences on deployed systems. Submissions for industrial presentations can either be an extended abstract (1-2 pages) or an industrial paper up to 6 pages long. (c) *Position papers:* The purpose of a position paper is to expose a new problem or advocate a new approach to an old idea. Participants will be invited based on the submission's originality, technical merit, topical relevance, and likelihood of leading to insightful technical discussions at the symposium. A position paper can be no more than 6 pages long.

Important Dates

Paper Submission: Jan 15, 2010
Notification: Feb 22, 2010
Camera-Ready: Mar 22, 2010

Conference Officers

General Chair:
Joseph M. Hellerstein, U. C. Berkeley
Program Chairs:
Surajit Chaudhuri, Microsoft Research
Mendel Rosenblum, Stanford University
Steering Committee:
Phil Bernstein, Microsoft Research
Ken Birman, Cornell University
Joseph M. Hellerstein, U. C. Berkeley
John Ousterhout, Stanford University
Raghu Ramakrishnan, Yahoo! Research
Doug Terry, Microsoft Research
John Wilkes, Google
Publicity Chair:
Aman Kansal, Microsoft Research
Treasurer:
Brian Cooper, Yahoo! Research

Program Committee

Anastasia Ailamaki, EPFL
Brian Bershad, Google
Michael Carey, UC Irvine
Felipe Cabrera, Amazon
Jeff Chase, Duke
Dilma M da Silva, IBM
David Dewitt, Microsoft
Shel Finkelstein, SAP
Armando Fox, UC Berkeley
Tal Garfinkel, Stanford
Alon Halevy, Google
James Hamilton, Amazon
Jeff Hammerbacher, Cloudera
Joe Hellerstein, UC Berkeley
Alfons Kemper, Technische Universität München
Donald Kossman, ETH
Orran Krieger, Vmware
Jeffrey Naughton, University of Wisconsin, Madison
Hamid Pirahesh, IBM
Raghu Ramakrishnan, Yahoo!
Krithi Ramamritham, Indian Institute of Technology, Bombay
Donovan Schneider, Salesforce.com
Andy Warfield, University of British Columbia
Hakim Weatherspoon, Cornell

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398