Recent Progress on Selected Topics in Database Research
A Report from Nine Young Chinese Researchers Working in the United States

Zhiyuan Chen (Microsoft Research, zhchen@microsoft.com)
Chen Li (University of California, Irvine, chenli@ics.uci.edu)
Jian Pei (State University of New York at Buffalo, jianpei@cse.buffalo.edu)
Yufei Tao (Carnegie Mellon University, taoyf@cs.cmu.edu)
Haixun Wang (IBM T.J. Watson Research Center, haixun@us.ibm.com)
Wei Wang (University of North Carolina at Chapel Hill, weiwang@cs.unc.edu)
Jiong Yang (University of Illinois at Urbana Champaign, jioyang@cs.uiuc.edu)
Jun Yang (Duke University, junyang@cs.duke.edu)
Donghui Zhang (Northeastern University, donghui@ccs.neu.edu)

Abstract

The study on database technologies, or more generally, the technologies of data and information management, is an important and active research field. Recently, many exciting results have been reported. In this fast growing field, Chinese researchers play more and more active roles. Research papers from Chinese scholars, both in China and abroad, appear in prestigious academic forums.

In this paper, we, nine young Chinese researchers working in the United States, present concise surveys and report our recent progress on the selected fields that we are working on. Although the paper covers only a small number of topics and the selection of the topics is far from balanced, we hope that such an effort would attract more and more researchers, especially those in China, to enter the frontiers of database research and promote collaboration. For the obvious reason, the authors are listed alphabetically, while the sections are arranged in the order of the author list.

1 Automatic Database Tuning and Administration[1]

After decades of development, today's database systems all have numerous features, making it very difficult to choose these features toward the need of the specific applications using them. For example, building indexes and materialized views often dramatically improve the performance on a given query workload, but it is very difficult to select the necessary indexes and views because such decision depends on how these queries are executed. On the other hand, the cost of hardware has dropped dramatically. Thus the cost for human to tune and manage the database systems often dominates the cost of ownership. To reduce such cost, it is desirable to automate database tuning and administration.

Database tuning and administration includes physical database design and tuning system parameters. Physical database design includes selecting indexes, views, vertical partitioning and horizontal partitioning, parallel database design, etc. Tuning system parameters includes selecting the serializability level, locking granularity, placement of log files, buffer pool size, RAID levels, cache sizes and placement, etc.

There has been much work in the area of physical database design. Earlier work [7] uses a stand-alone cost model to evaluate the goodness of a configuration. However, all such work has the drawback that it is difficult to keep the stand-alone cost model consistent with the cost model used by the query optimizer.

More recent work proposed to use the query optimizer for cost estimation, and to use a search algorithm to enumerate possible configurations. Such work include [4, 5, 8] in the area of index selection, [2] in the area of

[1]This section is contributed by Dr. Zhiyuan Chen, Microsoft Research, zhchen@microsoft.com.

view selection, and [6] in the area of parallel database design.

In the area of system parameter tuning, there is plenty of work giving practical guidelines [3]. However, there is relatively fewer attempt automating this [1].

There are many research problems unsolved in this area. First, very little work has been done in automatically tuning system parameters, and it is challenging to predict the system performance after changing such parameters. Second, little is known on how to adjust the system to changes of the workload. Ideally, the database system shall be able to automatically adjust to such changes. Third, given the numerous features to tune, it remains challenging to identify the system bottleneck as well as to tune all these together.

2 Integrating Information from Heterogeneous Data Sources[2]

The purpose of data integration (a.k.a. information integration, data mediation) is to support seamless access to autonomous, heterogeneous information sources, such as legacy databases, corporate databases connected by intranets, and sources on the Web. Many research systems (e.g., [10, 11, 13]) have been developed to achieve this goal. These systems adopt a mediation architecture [19], in which a user poses a query to a *mediator* that retrieves data from underlying sources to answer the query. A *wrapper* on a source is used to perform data translation and local query processing.

Intensive research has been conducted on challenges that arise in data integration. The first challenge is how to support interoperability of sources, which have different data models (relational, XML, etc.), schemas, data representations, and querying interfaces. Wrapper techniques have been developed to solve these issues. The second challenge is how to model source contents and user queries, and two approaches have been widely adopted. In the local-as-view (LAV) approach, a collection of *global predicates* are used to describe source contents as views and formulate user queries. Given a user query, the mediation system decides how to answer the query by synthesizing source views, called *answering queries using views*. Many techniques have been developed to solve this problem [9, 12], and these techniques can also be used in other database applications such as data warehousing and query optimization. Another approach to data integration, called the global-as-view (GAV), assumes that user queries are posed directly on global views that are defined on source relations. In this approach, a query plan can be generated using a view-expansion process. Researchers mainly focus on efficient query processing in this case. The third challenge is how to process and optimize queries when sources have limited query capabilities. For instance, the Amazon.com source can be viewed as a database that provides book information. However, we cannot easily download all its books. Instead, we can query the source by filling out Web search forms and retrieving the results. Studies have been conducted on how model and compute source capabilities, how to generate plans for queries, and how to optimize queries in the presence of limited capabilities [14, 20].

Here we give three of the many open problems in data integration that need more research investigation. First, most previous mediation systems adopt a *centralized* architecture. Recently, database applications are seeing the emerging need to support data integration and sharing in *distributed*, *peer-based* environments. In such an environment, autonomous peers (sources) connected by a network are willing to exchange data and services with each other. Thus each peer is both a data source and a "mediator." Several projects have been proposed to study issues in this new architecture [17, 18]. Second, researchers are paying more attention on how to deal with source heterogeneity by cleansing data. One particular problem is called *data linkage*, i.e., identifying and linking duplicate records efficiently and effectively. Sources frequently contain approximately duplicate fields and records that refer to the same real-world entity, but are not identical, such as "Tom M. Franz" versus "Franz, T.", and "Barranca Blvd., Irvine, CA" versus "Barranca Boulevard, Irvine, California". Variations in representations may arise from typographical errors, misspellings, abbreviations, and other reasons. This problem is especially severe when data is automatically extracted from unstructured or semi-structured documents or Web pages. In order to integrate information from sources, we need to "clean" the data before doing high-level processing. Recently many results are developed on data cleansing and linkage [16, 15]. Third, the advent of XML introduces many new problems that need to be solved to support data integration.

[2]This section is contributed by Dr. Chen Li, University of California, Irvine, chenli@ics.uci.edu.

3 Mining Changes from Data Streams[3]

A growing number of emerging applications, such as sensor networks, networking flow analysis, and e-business and stock market online analysis, have to handle various data streams. It is demanding to conduct advanced analysis and data mining over fast and large data streams to capture the trends, patterns, and exceptions. Recently, some interesting results have been reported for modelling and handling data streams (see [21] for a comprehensive overview), such as monitoring statistics over streams and query answering (e.g., [23, 30, 24])). Furthermore, conventional OLAP and data mining models have been extended to tackle data streams, such as multi-dimensional analysis (e.g., [22]), clustering (e.g., [31]) and classification (e.g., [25, 32]).

While extending the existing data mining models to tackle data streams may provide valuable insights into the streaming data, it is high time we considered the following fundamental question: *Compared to the previous studies on mining various kinds of data, what are the distinct features/core problems of mining data streams?* In other words, *from mining data streams, do we expect something different than mining other kinds of data?*

Previous studies (e.g., [21, 29]) argue that mining data streams is challenging in the following two respects. On the one hand, random access to fast and large data streams may be impossible. Thus, multi-pass algorithms (i.e., ones that load data items into main memory multiple times) are often infeasible. On the other hand, the exact answers from data streams are often too expensive. Thus, approximate answers are acceptable. While the above two issues are critical, they are not unique to data streams. For example, online mining very large databases also requires ideally one-pass algorithms and may also accept approximations.

We argue that one of the keys to mining data streams is *online mining of changes*. For example, consider a stream of regular updates of various aircrafts' positions. An air traffic controller may be interested in the clusters of the aircrafts at each moment. However, instead of checking details for "normal" clusters, she/he may be more interested in those "abnormal" clusters, e.g., fast growing clusters indicating the forming of a traffic jam. In general, while the patterns in snapshots of data streams are important and interesting, the *changes to the patterns* may be more critical and informative. With data streams, people are often interested in mining queries like "*compared to the history, what are the distinct features of the current status?*" and "*what are the relatively stable factors over time?*" Clearly, to answer the above queries, we have to examine the changes.

Some previous works also involve change detection. For example, the emerging patterns [27] characterize the changes from one data set to the other. In [28], Ganti et al. propose methods to measure the differences of the induced models in data sets. Incremental mining studies how to update the models/patterns by factoring in the incremental part of data. However, mining data streams requires online and dynamic detection and summarization of interesting changes.

Interesting research problems on mining changes in data streams can be divided into three categories: modelling and representation of changes, mining methods, and interactive exploration of changes.

First, while the term "changes" sounds general and intuitive, it is far from trivial to define and describe changes in data streams. First, it is essential to propose concise query language constructs for describing the mining queries on changes in data streams. There can be many kinds of changes in data streams, and different users may be interested in different kinds. The user should be able to specify the changes she/he wants to see. Moreover, the system should be able to rank changes based on interestingness. The operations should be integrable into the existing data mining models and languages. An "algebra" for change mining may be essential. Second, methods of summarizing and representing the changes need to be developed. In particular, effective visualization of changes is very important. Third, while the model for mining "first order" changes is common and useful, the model for mining "higher order" changes can be an important kind of knowledge in some dynamic environments. For example, a stock market analyst may feel particularly interested in the changes in the ranges of price vibration, while the range of price vibration itself is a description of changes.

Second, efficient and scalable algorithms are needed for mining changes in data streams, at various levels. First, specific algorithms can be developed for specific change mining queries. While such query-specific approaches may not be systematic, it will provide valuable insights into the inherent properties, challenges and basic methods of change mining. Second, general evaluation methods for "change mining queries" should be developed based on the general model/query language/algebra. Third, facilities for various aspects of change

[3]This section is contributed by Dr. Jian Pei, State University of New York at Buffalo, jianpei@cse.buffalo.edu.

mining, such as quality management, should be considered. For example, algorithms should be able to meet user's specification on level/granularities/approximation error bound of change mining.

Third, the results from change mining per se form data streams, which can sometimes be large and fast. It is important to develop effective approaches to support user's interactive exploration of the changes. For example, a user may want to monitor the changes at an appropriate level. Once some interesting changes are detected, she/he can closely inspect the related details.

To the best of our knowledge, the above problems have not been researched systematically so far. By no means is the above list complete. We believe that thorough studies on these issues will bring about many challenges, opportunities, and benefits to stream data processing, management, and analysis. Our recent studies [33, 26] are concrete steps towards this direction.

4 Spatio-Temporal Indexing and Quering[4]

The *spatio-temporal database* (STDB) has received considerable attention during the past few years, due to the emergence of numerous applications (e.g., flight control systems, weather forecast, mobile computing, etc.) that demand efficient management of moving objects. These applications record objects' geographical locations (sometimes also shapes) at various timestamps, and support queries that explore their historical and future (predictive) behaviors. The STDB significantly extends the traditional *spatial database*, which deals with only stationary data and hence is inapplicable to moving objects, whose dynamic behavior requires re-investigation of numerous topics including data modeling, indexes, and the related query algorithms. In this section, we survey the existing solutions for these issues.

Depending on the temporal aspects of data, a STDB aims at either *historical* or *predictive* retrieval. Specifically, given a set of objects o_1, o_2, ..., o_N (where N is termed the *cardinality*), a historical STDB stores, for each object o_i ($1 \leq i \leq N$), its extent $o_i.E(t)$ at all the timestamps t in the history. Following the convention of spatial databases, each extent $o_i.E(t)$ can be a polygon describing the object's actual shape at time t (e.g., the contour of a moving typhoon). Specially, if the shape is not important (e.g., cars, flights, etc.), $o_i.E(t)$ degenerates to a point describing the location of o_i at time t. In practice, the extents of the same object at the successive timestamps can be compressed using various methods (e.g., if the object remains stationary at several continuous timestamps, its extent is stored only once during this period). A predictive STDB, on the other hand, stores, for each (usually point) object o_i, its most recent updated location $o_i.L(t_{upd})$ (where t_{upd} is the time of the object's last update), and the motion function describing its current movement. The most popular motion function is the linear function [54, 39, 40], because it (i) can approximate any trajectory, and (ii) requires the fewest number of parameters. Specifically, in addition to $o_i.L(t_{upd})$, the system only needs to record the object's velocity $o_i.vel$, such that the object's location at any future time $t > t_{upd}$ can be calculated as $o_i.L(t) = o_i.L(t_{upd}) + o_i.vel \cdot (t - t_{upd})$. Using such modeling, an object needs to issue an update to the database only if the parameters of its motion function (e.g., $o_i.vel$ for linear movement) change.

Since the spatial database can be regarded as a special type of STDB where all the objects have zero velocities, all the spatial query types naturally find their counterparts in STDB, except that they are augmented with additional temporal predicates. Specifically, a *window query* (WQ) specifies a query region q_R and time interval q_T (consisting of continuous timestamps), and finds all the objects whose extents (or locations for point data) intersect q_R during q_T. Particularly, the *selectivity* of a WQ equals the number of retrieved objects divided by the dataset cardinality, and its accurate estimation [38, 63, 42] is imperative to query optimization. A *k nearest neighbor* (kNN) query specifies a query point q_P and time interval q_T, and finds the k objects whose distances to q_P during q_T are the smallest. These problems become even more complex if query regions/points (in WQ/kNN) are also moving. While the above queries involve only one dataset, the *within-distance* join (WDJ), given two datasets S_1, S_2, reports all the object pairs (o_1,o_2) in the cartesian product $S_1 \times S_2$, such that the distance between o_1, o_2 during a query time interval q_T is smaller than certain threshold d. The *selectivity* of a join is the number of retrieved pairs divided by the size of $S_1 \times S_2$. Similarly, the *k closest pair* (kCP) query retrieves the k object pairs (o_1,o_2) such that the distance of o_1, o_2 during q_T is the smallest, among all the pairs

[4]This section is contributed by Dr. Yufei Tao, Carnegie Mellon University, taoyf@cs.cmu.edu.

in $S_1 \times S_2$. Note that the above queries can be defined in both historical and predictive STDB.

In addition to queries inherited from conventional spatial databases, the dynamic nature of STDB also leads to several novel query types. For historical databases, [48] introduces the *navigational* WQ, which specifies two query regions q_{R1}, q_{R2} and timestamps q_{T1}, q_{T2}, and retrieves all the objects that intersect q_{R1} at q_{T1}, and also intersect q_{R2} at q_{T2} (e.g., find all the vehicles that appeared in Harvard at 5pm yesterday and then appeared in MIT 10 minutes later). In predictive STDB, [58] points out that the results of traditional queries (i.e., WQ, kNN, WDJ, kCP) are usually inadequate because they may change (sometimes almost immediately) due to the movements of objects and/or queries (e.g., a user's nearest gas station may change as s/he drives on the highway). Motivated by this, [58] proposes the *time-parameterized* (TP) query, which applies to any traditional query, and returns, in additional to the result R, also (i) an expiry time T of R, and (ii) the change C of the result after T. An example of TPNN is to report (i) the nearest station s, (ii) when s will cease to be the nearest (given the user's moving direction and speed), and (iii) the new nearest station after the expiry of s. The concept of TP is extended to the *continuous query* in [55, 37, 60, 59], which is another general concept applicable to all traditional queries, and aims at continuously tracking the result changes until certain conditions are satisfied. A continuous WQ, for instance, may "return the aircrafts within 10 miles from flight UA183 now, and continuously update this information until its arrival". In TP and continuous processing, the moving direction of the query can be clearly specified, which is not true in some applications (e.g., a tourist wandering around casually). The concept useful in such scenarios is the *location-based* (LB) query [66], which applies to WQ and kNN, and finds the query result as well as its validity region such that, as long as the query is in this region, its result will remain the same. For example, a LB NN may return the nearest restaurant of a tourist, as well as a validity region in which the restaurant will remain the nearest.

Numerous access methods have been proposed for efficient spatio-temporal query processing. A straightforward approach to index historical STDB is to create a spatial index (the most common ones include the R-tree [41, 36]) at each timestamp in history, managing objects' extents at that timestamp. This is the idea behind the so-called *partially persistent structures* [52, 35], which, in order to reduce the space consumption, allow the R-trees at consecutive timestamps to share common nodes, if the objects in these nodes do not incur extent changes. The first partially persistent structure, the *historical R-tree* (HR-tree) [47], however, still involves considerable data redundancy as analyzed in [62], which led to the development of the *multi-version R-tree* (MVR-tree) [46], and its subsequent versions [57, 45, 43]. Besides the partially persistent methodology, historical STDB can also be indexed using a 3D R-tree by treating time just as an extra dimension (in addition to the two spatial dimensions). Specifically, each record in the 3D R-tree [65] represents a 3D box, whose spatial projection corresponds to the extent of a stationary object, and whose temporal projection denotes the time interval during which the object is stationary. Similar ideas are used in the *trajectory bundle tree* (TB-tree) [48], a structure optimized for navigational WQ queries. In practical STDB, [64] adapts the Quadtree [53] (a spatial index) for indexing the movements of 1D objects, while the *time-parameterized R-tree* [51] (TPR-tree) and its improved versions [50, 61] support objects of arbitrary dimensionality. Finally, indexing moving objects has also been studied in theory [44, 34, 56, 49], which develop numerous interesting structures with provable worst-case performance bounds. These bounds, however, usually involve large hidden constants, rendering these "theoretical" solutions to be outperformed by the "practical" solutions introduced earlier.

5 Indexing XML by Tree Structures[5]

With the growing importance of XML in data exchange, much research has been done in providing flexible query mechanisms to extract data from XML documents [69, 72, 71, 68, 67, 70]. We propose a novel index structure that addresses a wide range of challenges in indexing semi-structured data [73]. Unlike state-of-the-art approaches that disassemble a structured query into multiple sub-queries, and then *join* the results of these sub-queries to provide the final answers, our method uses tree structures as the basic unit of query to avoid expensive join operations.

XML provides a flexible way to define semi-structured data. Figure 1 shows an XML document representing a purchase record. Many state-of-the-art approaches create indexes on paths or nodes in DTD trees. However,

[5]This section is contributed by Dr. Haixun Wang, IBM T.J. Watson Research Center, haixun@us.ibm.com.

queries involving branching structures usually have to be disassembled into multiple sub-queries, each corresponding to a single path in the graph. The results of these sub-queries are then combined by expensive *join* operations to produce final answers. For the same reason, these methods are also inefficient in handling '*' or '//' queries, which too, correspond to multiple paths. Moreover, to retrieve semi-structured data efficiently, it is essential to have index on both structure and content of the XML data. Nevertheless, many algorithms index on structure only, or index on structure and content separately, which means attribute values are not used for filtering in the most effective way.

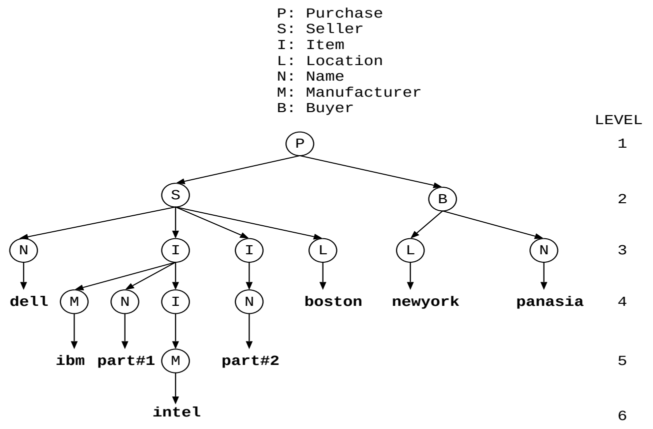

Figure 1: A Single Purchase Record

We transform XML data and queries into *structure-encoded sequences* by a depth-first traversal of their tree structure, and represent each node by (Symbol, Prefix) where symbol encodes the node and prefix is the path from the root node. We also convert XML queries to sequences. Most structural XML queries can be performed through (non-contiguous) subsequence matching. The only exception occurs when a branch has multiple identical child nodes [73]. The benefits of modeling XML queries through sequence matching is that structural queries can be processed as a whole instead of being broken down to smaller query units (paths or nodes of XML document trees). For example, Figure 2 is a sequential representation of the purchase record, where the underlining part matches query "find purchase records of Boston sellers and NY buyers".

$$\begin{array}{l}
\underline{(P,\epsilon)}, \underline{(S,P)}, (N,PS), (v_1,PSN), (I,PS), (M,PSI), \\
(v_2,PSIM), (N,PSI), (v_3,PSIN), (I,PSI), (M,PSII), \\
(v_4,PSIIM), (I,PS), (N,PSI), (v_5,PSIN), \underline{(L,PS)}, \\
\underline{(v_6,PSL)}, \underline{(B,P)}, \underline{(L,PB)}, \underline{(v_7,PBL)}, (N,PB), (v_8,PBN)
\end{array}$$

Figure 2: Sequential Representation of Figure 1.

The index structure is built in three steps: i) adding all structure-encoded sequences into a suffix tree; ii) labeling each node in the suffix tree by $\langle n, size\rangle$ through a preorder traversal where n is the preorder number of the node, and $size$ is the number of its descendents; and iii) for each node (Symbol, Prefix) labeled $\langle n, size\rangle$, inserting it to the `D-Ancestor B`$^+$`Tree` using (Symbol, Prefix) as the key, and then into the `S-Ancestor B`$^+$`Tree` using n as the key.

Suppose node x, labeled with $\langle n_x, size_x\rangle$, is one of the nodes matching a query prefix $q_1, \cdots, q_{i-1}$. To match the next element q_i in the query, we consult the `D-Ancestor B`$^+$`Tree` using q_i as a key. The `D-Ancestor B`$^+$`Tree` returns the root of an `S-Ancestor B`$^+$`Tree`. We then issue a range query $n_x < n \leq n_x + size_x$ on the `S-Ancestor B`$^+$`Tree` to find the descendants of x immediately. For each descendant, we use the same process to match symbol q_{i+1}, until we reach the last element of the query.

If node y is one of the nodes that matches the last element in the query, then the document IDs associated with y or any descendant node of y are answers to the query. Based on y's label, say $\langle n_y, size_y\rangle$, we know y's

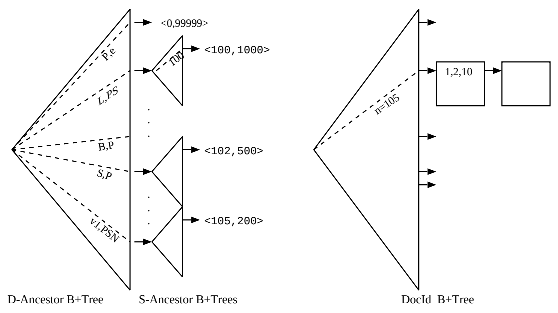

Figure 3: The index structure

descendants are in the range of $(n_y, n_y + size_y]$. Thus, we perform a range query $[n_y, n_y + size_y]$ on the `DocId` `B`$^+$`Tree` to retrieve all the document IDs for y and y's descendants.

Representing XML documents by sequences and performing XML queries by subsequence matching create many research issues. We are currently improving the index structure to support i) dynamic index, as the labeling scheme described above prevents the index structure from supporting dynamic insertion and deletion [73]; ii) better selectivity, as different (symbol,prefix) pairs have different selectivity, hence their order in the query sequence has an impact on the query performance; iii) approximate queries, where approximate tree matching can be simulated by approximate subsequence matching; and iv) top-K queries.

6 Clustering Biological Data[6]

Currently, bio-informatics has became one of the research areas that receive most of attention. In general, bio-informatics aims to solve complicated biological problems, e.g., gene regulatory network induction, motif discovery, etc, with computation algorithms. Many of the data is in the form of sequences, e.g., DNA sequences, protein sequences, etc. It is widely believed that the functionality of these biological sequences are highly depended on the its structures. Automatically extracting/analyzing the biological structures is an important step in better understanding their functionality.

Clustering has been widely recognized as a powerful data mining technique and has been studied extensively during recent years. The major goal of clustering is to create a partition of objects such that objects in each group have similar features. The result can potentially reveal unknown object groups/categories that may lead to a better understanding of the nature. In the context we address, each object is a symbol sequence and all potential features of a sequence are encoded (implicitly) in the specific symbol layout in the sequence. It is interesting to notice that these structural characteristics sometimes can uniquely determine the functional properties of the sequence and often plays a decisive role in meaningful clustering of the sequence.

Significant challenges exist on how to abstract and represent distinctive structural characteristics (of sequences of vastly different lengths) and to design an effective yet efficient similarity measure based on them. One possible approach to tackle this difficulty is to use the *edit distance* [74] to measure the distance between each pair of sequences. This, however, is not an ideal solution because, in addition to its inefficiency in calculation, the edit distance only captures the optimal global alignment between a pair of sequences, but ignores many other local alignments that often represent important features shared by the pair of sequences. These overlooked features may be very crucial to produce meaningful clusters.

Another approach that has been widely used in document clustering is the keyword-based method. Instead of being treated as a sequence, each text document is regarded as a set of keywords or phrases and is usually represented by a weighted word vector. The similarity between two documents is measured based on keywords and phrases shared by them and is often defined in some form of normalized dot-product. A direct extension of this method to generic symbol sequences is to use short segments of fixed length q (generated using a sliding

[6]This section is contributed by Dr. Wei Wang, University of North Carolina at Chapel Hill, weiwang@cs.unc.edu.

window through each sequence) as the set of "words" in the similarity measure. This method is also referred to as the q-gram based method in the literature. While the q-gram based approach enables significant segments (i.e., keywords/phrases/q-grams) to be identified and used to measure the similarity between sequences regardless of their relative positions in different sequences, valuable information may be lost as a result of ignoring sequential relationship (e.g., ordering, correlation, dependency, etc.) among these segments, which impacts the quality of clustering.

In [75], a new generative model is devised to measure the similarity between a cluster and a sequence. Statistics properties of sequence construction are used to assess the similarity. Sequences belonging to one cluster may subsume to the same probability distribution of symbols (conditioning on the preceding segment of a certain length), while different clusters may follow different underlying probability distributions. This feature, typically referred to as *short memory*, which is common to many applications, indicates that, for a certain sequence, the empirical probability distribution of the next symbol given the preceding segment can be accurately approximated by observing no more than the last L symbols in that segment. Significant features of such probability distribution can be very powerful in distinguishing different clusters. By extracting and maintaining significant patterns characterizing (potential) sequence clusters, one can easily determine whether a sequence should belong to a cluster by calculating the likelihood of (re)producing the sequence under the probability distribution that characterizes the given cluster.

In many applications, neither the number of clusters in the data set nor the percentage of outliers is known in advance. For instance, the biologists usually do not know the number of protein families for a set of newly mapped protein sequences. To solve this problem, a novel clustering algorithm (CLUSEQ) that produces a set of possibly overlapped clusters is proposed and is able to automatically adjust the number of clusters and the boundary to separate clustered sequences from outliers, benefiting from a unique combination of successive new cluster generation and cluster consolidation. CLUSEQ is an iterative algorithm. At each iteration, it first random generate a set of new seed clusters and add them to the set of existing clusters. Next, CLUSEQ reclusters all sequences according to the set of clusters. Finally, the thresholds are adjusted to fit the current clusters. The algorithm terminates when no improvement on the clustering is made in the current iteration.

CLUSEQ provides a sequence level generative model (by assuming the sequences belonging to a cluster are generated by a conditional probability distribution). However, many biological data are present in more complicated forms. For instance, the natural state of a protein is a folded 3D structure. This additional complexity poses great challenges to the clustering task and mandates further investigation and development of powerful methods.

7 Mining Sequential Patterns[7]

With the emergence of various applications, e.g., web log traces, system log traces, bio-informatics, etc., analyzing sequential data becomes one of the urgent problems with grate practical implications. Sequential patterns is an important model for analyzing the sequential data. In this section, we are surveying several techniques used for mining various sequential patterns.

A sequence may have several periodic patterns. For instance, an event may occur at 6th, 12th, 18th, ... position in a sequence. Periodic pattern can further our understanding of the behavior of the underlining system that generates the sequence. The periodic sequential patterns were introduced in [76]. In this work, a sequence is partitioned into non-overlap contiguous portions of segments. Each segment has the same length, l. A pattern of period l will occur once at each segment. Using this as a pruning condition, the search space for periodic patterns can be reduced.

However, in many situations, the pattern may not exhibit the periodicity in the entire sequence. The pattern may be only present during some portion of the sequence. For example, in a system trace log, the system behavior may change over time. Two parameters, namely *min_rep* and *max_dis*, are employed to qualify valid patterns and the event subsequence containing it, where this subsequence in turn can be viewed as a list of **valid segments** of perfect repetitions interleaved by disturbance. Each valid segment is required to be of at least

[7]This section is contributed by Dr. Jiong Yang, University of Illinois at Urbana Champaign, jioyang@cs.uiuc.edu.

min_rep contiguous repetitions of the pattern and the length of each piece of disturbance is allowed only up to *max_dis*. The intuition behind this is that a pattern needs to repeat itself at least a certain number of times to demonstrate its significance and periodicity. On the other hand, the disturbance between two valid segments has to be within some reasonable bound. Otherwise, it would be more appropriate to treat such disturbance as a signal of "change of system behavior" instead of random noise injected into some persistent behavior. The parameter *max_dis* acts as the boundary to separate these two phenomena. This type of pattern is called *asynchronous pattern* [77].

To efficiently mine the asynchronous patterns, a dynamic programming algorithm is devised. First, a distance-based pruning mechanism is proposed to discover all possible periods and the set of events that are likely to appear in some pattern of each possible period. In order to find the longest valid subsequence for all possible patterns, a level-wise approach is utilized. The Apriori property also holds on patterns of the same period. That is, a valid segment of a pattern is also a valid segment of any pattern with fewer events specified in the pattern. For example, a valid segment for $(d_1, d_2, *)$ will also be one for $(d_1, *, *)$. Then, for each likely period, all valid patterns with their longest supporting subsequences can be mined via an iterative process.

In many applications, users may be interested in not only the frequently occurred patterns, but also the surprising patterns (i.e., beyond prior expectation) as well. A large number of occurrences of an "expected" frequent pattern sometimes may not be as interesting as a few occurrences of an "expected" rare pattern. The support model is not ideal for these applications because, in the support model, the occurrence of a pattern carries the same weight (i.e., 1) towards its significance, regardless of its likelihood of occurrence. Intuitively, the assessment of significance of a pattern in a sequence should take into account the expectation of pattern occurrence (according to some prior knowledge). A new model is proposed in [78] to characterize the class of so-called *surprising* patterns (instead of frequent patterns).

The measure of *surprise* should have the following properties. (1) The surprise of a pattern occurrence is anti-monotonic with respect to the likelihood that the pattern may occur by chance (or by prior knowledge). (2) The metric should have some physical meaning, i.e., not arbitrary created. It is fortunate that the *information* metric which is widely studied and used in the communication field can fulfill both requirements. Intuitively, information is a measurement of how likely a pattern will occur or the amount of "surprise" when a pattern actually occurs. If a pattern is expected to occur frequently based on some prior knowledge or by chance, then an occurrence of that pattern carries less information. Thus, we use information to measure the surprise of an occurrence of a pattern. The *information gain* metric is introduced to represent the accumulated information of a pattern in an event sequence and is used to assess the degree of surprise of the pattern.

Many problems in sequential pattern mining still remain as open problems. When the amount of data is large, we only can afford to scan data once, e.g., the stream environment. In this type of application, it is not only important to discover the new sequential patterns, but also interesting to discover how they evolve over the time. For instance, at the beginning of a stream *abc* is a frequent pattern, but later it becomes *abd*. From this change, we may conjecture that some user/system behavior has changed.

8 Derived Data Maintenance[8]

The use of *derived data* to facilitate access to *base data* is a recurring technique in many areas of computer science. Used in hardware and software caches, derived data speeds up accesses to the base data. Used in replicated systems, it improves reliability and performance of applications in a wide-area network. Used as index structures, it provides fast alternative access paths to the base data. Used as materialized views [85] in databases or data warehouses [82], it improves the performance of complex queries over the base data. Used as synopses [84], it provides fast, approximate answer to queries or statistics necessary for cost-based query optimization. Derived data varies in complexity: It can be a simple copy of the base data, in the cases of caching and replication, or it can be the result of complex structural or content transformation of the base data, in the cases of indexes and materialized views. Derived data also varies in precision: Caches and materialized views are usually precise, while synopses are approximate.

[8]This section is contributed by Dr. Jun Yang, Duke University, junyang@cs.duke.edu.

Regardless of the form of the derived data, however, a fundamental problem is efficient maintenance of the derived data. If base data changes, we need to keep the derived data consistent with the new state of the base data. Derived data maintenance is becoming an increasingly important and difficult problem. First, more and more data is being generated at high speeds (e.g., sensor networks, network monitors, Web clickstreams, high-volume transactions from multiple sources), making it extremely expensive to compute queries from scratch. A solution is to store the query results as derived data, and incrementally maintain them as new data arrives, essentially creating a one-pass algorithm. Second, computing is becoming increasingly pervasive, and there are more and more clients accessing dynamically changing data simultaneously over a wide-area network. An approach is to place replicas of data at various locations in the network based on client access patterns. Maintenance of derived data in this case must scale up not only to the size, but also to the number of replicas. Third, as computing resources keep getting cheaper, human resources become more expensive. Traditionally, systems often rely on hand-tuning of derived data to achieve optimal performance, e.g., index tuning in databases. Nowadays, excessive hand-tuning is no longer an option. This requirement calls for more self-tuning approaches to derived data management.

At the first glance, derived data maintenance has been extensively studied and well understood in specific contexts, e.g., index updates and materialized view maintenance in databases, cache coherence and replication protocols in distributed systems. Although they share the same underlying theme, these techniques have been developed largely independently by different research communities. However, an increasing number of newer and more complex applications call for a creative combination of the traditionally separate ideas. A good example is *semantic caching* [83], which has received tremendous interests recently for its application in caching dynamic contents [86, 81, 80]. Semantic caching incorporates the idea of materialized views into a cache. Traditionally, caching is done at the object level, and hence only supports access to objects by identifiers. If the cache receives a declarative query, it is generally impossible to tell whether the cached data provides a complete answer to the query; therefore, base data must be queried. Semantic caching supports declarative queries by maintaining the cache as a pool of materialized views. From the semantic descriptions of materialized views, it is possible to determine whether the query answers are complete, so base data is only queried when necessary. This feature makes semantic caching a perfect solution for those applications that require declarative accesses to data.

With more thinking "outside the box," we can discover more techniques such as semantic caching that combine multiple flavors of derived data to provide better solutions to problems. For example, a lot of work in data warehousing focuses on storing auxiliary data to make warehouse data self-maintainable [89, 79, 90], without ever accessing base data. In many cases, e.g., materialized top-k and join views, we must store huge amounts of auxiliary data to guarantee self-maintenance. Borrowing the caching idea, we could remove the rigid requirement of self-maintenance, and manage auxiliary data as a semantic cache whose size is tunable by the application. The warehouse data is no longer guaranteed to be self-maintainable since the cache may occasionally "miss," i.e., the auxiliary data required for maintenance is not found in the cache. However, with a good cache management policy, we may be able to achieve a low miss rate that lowers the expected overall cost of maintenance. We have obtained some promising results for top-k view maintenance [92], and work is under way to generalize this technique to other types of views including joins.

As another exmaple, we can apply indexing techniques to materialized view maintenance. Traditionally, materialized views are simply stored as database tables (possibly with a B+-tree primary index). However, updating such views can be very expensive, as clearly illustrated by our earlier work on maintaining materialized temporal aggregate views [91]. For a temporal aggregate view, even a slight modification of the base table (e.g., on a single row) may affect a large number of rows in the view, making it prohibitively expensive to maintain incrementally. The solution is to materialize the view, not as a regular table or B+-tree, but instead as a novel index structure called the *SB-tree*, which supports efficient range updates to the aggregate view (in time logarithmic in the size of the view).

For yet another example, consider applying view maintenance techniques to data replication. Instead of propagating all updates immediately to each replica, we propagate them only when relevant views at the replica are affected. We could also throw in approximation techniques borrowed from synopses to improve performance further. Replicas can be approximations of the base data within the error bounds controlled by applications. Recent work on TRAPP [88, 87] represents a step toward this general direction.

9 Aggregation Query Processing[9]

In this section, we summarize some recent work on *selection-based aggregation.* The selection query has been one of the most widely used queries in databases. For instance, find the gas stations that are within a given spatial range. Such queries select a subset of records from a large collection of data based on a given selection condition. In contrast, the aggregation query aims not at finding the actual records themselves, but at some aggregate value of these records. For instance, find the total number of gas stations that are within a given range. Besides this COUNT query, some other interesting queries are SUM, AVERAGE, MIN, and MAX.

A straightforward approach to solve the selection-based aggregation problem is: first find all the records that satisfy the selection condition and then perform the aggregation on-the-fly. The problem with this approach is that the query performance is at least linear to the size of the selection result. If many records satisfy the selection condition, the performance is not satisfactory. A better approach is to build some specialized index which can help compute the aggregation result without scanning through the records. Below we summarize some major results on the latter approach.

The first problem we address is called *Range-Temporal Aggregation (RTA)*: "given a set of temporal records, each having a key, a time interval and a value, compute the total value of records whose keys are in a given range and whose intervals intersect with a given time interval". Previous work on temporal aggregation (the most efficient one being [95]) aggregate on the whole key space. To solve the RTA problem using the previous approaches, we would need to maintain a separate index for each possible key range, which is prohibitively expensive. [97] proposed a new index structure called *Multi-version SB-tree (MVSB-tree)* the solve the RTA problem in logarithmic time.

Another problem is the *Hierarchical Temporal Aggregation (HTA).* Here we are interested in computing temporal aggregates (both with and without the key-range predicate) with fixed storage space. Since historical data accumulates over time, while with fixed storage space we cannot keep all of them, we have to live with storing partial information. One approach is to keep just the most recent information. However, it leads to lose of the ability to answer queries for the past. In [96], we proposed to maintain the temporal aggregates under multiple granularities, with more recent information being aggregated at finer granularities.

When we consider the aggregation over spatial objects, a related problem is the *Box-Sum Aggregation*, which computes the total value of spatial objects whose regions intersect with a given region. One approach is to extend the R-tree index (state-of-art spatial index used for the selection query) by maintaining summary information in internal nodes of the tree. This approach was proposed by [93, 94] and is called the *aggregation R-tree (aR-tree).* However, the worst-case performance is still linear to the number of records. In [98] we proposed a new index structure called the BA-tree to this problem. The query performance is pretty much logarithmic to the number of records, which is a big improvement.

A variation of the previous problem is the *Functional Box-Sum Aggregation*: "given a set of objects, each having a box and a value function, and a query box q, compute the total value of all objects that intersect q, where the value contributed by an object r is the integral of the value function of r over the intersection between r and q". Compared with the original box-sum problem, in some cases the new problem more accurately captures the application needs. This is because an object contributes to the query result proportional to how large it intersects the query rectangle. In [98], we proposed techniques to solve the functional box-sum problem. Basically, we managed to reduce this problem to the original box-sum problem where no function was involved.

In summary, in this section we have summarized some major research results on the selection-based aggregation problem as reported in [97, 98, 96]. In all cases, the proposed specialized index structures have much better query performance than the existing non-specialized indices. An interesting future direction is how to extend the findings to the spatiotemporal aggregation problems. A simple case is to aggregate moving points. Each object is a weighted moving point whose location is a function of time. An aggregation query asks to compute the total weight of points which will move into a given area during a given time interval. A more complicated case is to aggregate moving objects with non-zero extents.

[9]This section is contributed by Dr. Donghui Zhang, Northeastern University, donghui@ccs.neu.edu.

References

[1] Sanjay Agrawal, Surajit Chaudhuri, Abhinandan Das, and Vivek Narasayya. Automating layout of relational databases. In *ICDE'2003*, 2003.

[2] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In *VLDB'2000*, pages 496–505, 2000.

[3] Philippe Bonnet and Dennis Elliott Shasha. *Database Tuning: Principles, Experiments, and Troubleshooting Techniques.* Morgan Kaufman, 2002.

[4] Surajit Chaudhuri and Vivek R. Narasayya. An efficient cost-driven index selection tool for microsoft SQL server. In *VLDB'1997*, pages 146–155, 1997.

[5] Surajit Chaudhuri and Vivek R. Narasayya. Index merging. In *ICDE'1999*, pages 296–303, 1999.

[6] Jun Rao, Chun Zhang, Nimrod Megiddo, and Guy M. Lohman. Automating physical database design in a parallel database. In *SIGMOD'2002*, pages 558–569, 2002.

[7] Steve Rozen and Dennis Shasha. A framework for automating physical database design. In *VLDB'1991*, pages 401–411, 1991.

[8] Gary Valentin, Michael Zuliani, Daniel C. Zilio, Guy M. Lohman, and Alan Skelley. DB2 advisor: An optimizer smart enough to recommend its own indexes. In *ICDE'2000*, pages 101–11, 2000.

[9] F. Afrati, C. Li, and J. D. Ullman. Generating efficient plans using views. In *SIGMOD*, pages 319–330, 2001.

[10] M. J. Carey et al. Towards heterogeneous multimedia information systems: The Garlic approach. *Proceedings RIDE-DOM'95*, pages 124–131, 1995.

[11] S. S. Chawathe et al. The TSIMMIS project: Integration of heterogeneous information sources. *IPSJ*, pages 7–18, 1994.

[12] A. Halevy. Answering queries using views: A survey. In *Very Large Database Journal*, 2001.

[13] A. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, pages 251–262, 1996.

[14] A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *PODS*, pages 105–112, 1995.

[15] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *The VLDB Journal*, pages 381–390, 2001.

[16] The Flamingo Project on Data Cleansing, University of California, Irvine.

[17] The Piaaza Project, University of Washington.

[18] The Raccoon Project on Distributed Data Integration and Sharing, University of California, Irvine.

[19] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.

[20] R. Yerneni, C. Li, H. Garcia-Molina, and J. D. Ullman. Computing capabilities of mediators. In *SIGMOD*, pages 443–454, 1999.

[21] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'02)*, Madison, WI, June 2002.

[22] Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-dimensional regression analysis of time-series data streams. In *Proc. 2002 Int. Conf. on Very Large Data Bases (VLDB'02)*, Hong Kong, China, Aug. 2002.

[23] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows (extended abstract). *citeseer.nj.nec.com/491746.html.*

[24] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In *Proc. 2002 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'02)*, Madison, Wisconsin, June 2002.

[25] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proc. 2000 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'00)*, pages 71–80, Boston, MA, Aug. 2000.

[26] G. Dong, J. Han, L.V.S. Lakshmanan, J. Pei, H. Wang, and P.S. Yu. Online mining of changes from data streams: Research problems and preliminary results. In *Proceedings of the 2003 ACM SIGMOD Workshop on Management and Processing of Data Streams*, San Diego, CA, June 2003.

[27] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD'99)*, pages 43–52, San Diego, CA, Aug. 1999.

[28] V. Ganti, J. Gehrke, and R. Ramakrishnan. A framework for measuring changes in data characteristics. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania*, pages 126–137. ACM Press, 1999.

[29] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: You only get one look. In *Proc. 2002 Int. Conf. on Very Large Data Bases (VLDB'02)*, Hong Kong, China, Aug. 2002.

[30] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continuous data streams. In *Proc. 2001 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'01)*, pages 13–24, Santa Barbara, CA, May 2001.

[31] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proc. IEEE Symposium on Foundations of Computer Science (FOCS'00)*, pages 359–366, Redondo Beach, CA, 2000.

[32] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proc. 2001 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'01)*, San Fransisco, CA, Aug. 2001.

[33] J. Pei, S.R. Ariwala, and D. Jiang. Online mining changes of clusters in data streams. In *Submitted for publication.*

[34] P. Agarwal, L. Arge, and J. Erickson. Indexing moving points. *PODS*, 2000.

[35] B. Becker, S. Gschwind, T. Ghler, B. Seeger, and P. Widmayer. An asymptotically optimal multiversion b-trees. *VLDB Journal*, 5(4):264–275, 1996.

[36] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robush access method for points and rectangles. *SIGMOD*, 1990.

[37] R. Benetis, C. Jensen, G. Karciauskas, and S. Saltenis. Nearest neighbor and reverse nearest neighbor queries for moving objects. *IDEAS*, 2002.

[38] Y. Choi and C. Chung. Selectivity estimation for spatio-temporal queries to moving objects. *SIGMOD*, 2002.

[39] L. Forlizzi, R. Guting, R. Nardelli, and M. Schneider. A data model and data structures for moving objects databases. *SIGMOD*, 2000.

[40] R. Gutting, M. Bohlen, M. Erwig, C. Jensen, N. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *TODS*, 25(1):1–42, 2000.

[41] A. Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD*, 1984.

[42] M. Hadjieleftheriou, G. Kollios, and V. Tsotras. Performance evaluation of spatio-temporal selectivity techniques. *SSDBM*, 2003.

[43] M. Hadjieleftheriou, G. Kollios, V. Tsotras, and D. Gunopulos. Efficient indexing of spatiotemporal objects. *EDBT*, 2002.

[44] G. Kollios, D. Gunopulos, and V. Tsotras. On indexing mobile objects. *PODS*, 1999.

[45] G. Kollios, D. Gunopulos, V. Tsotras, A. Delis, and M. Hadjieleftheriou. Indexing animated objects using spatiotemporal access methods. *TKDE*, 2001.

[46] A. Kumar, V. Tsotras, and C. Faloutsos. Designing access methods for bitemporal databases. *TKDE*, 10(1):1–20, 1998.

[47] M. Nascimento and J. Silva. Towards historical r-trees. *ACM Symposium on Applied Computing*, 1998.

[48] D. Pfoser, C. Jensen, and Y. Theodoridis. Novel approches to the indexing of moving object trajectories. *VLDB*, 2000.

[49] C. Procopiuc, P. Agarwal, and S. Har-Peled. Star-tree: An efficient self-adjusting index for moving points. *ALENEX*, 2000.

[50] S. Saltenis and C. Jensen. Indexing of moving objects for location-based services. *ICDE*, 2002.

[51] S. Saltenis, C. Jensen, S. Leutenegger, and M. Lopez. Indexing the positions of continuously moving objects. *SIGMOD*, 2000.

[52] B. Salzberg and V. Tsotras. A comparison of access methods for temporal data. *ACM Computing Survey*, 31(2):158–221, 1999.

[53] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley Publishing Company, 1990.

[54] A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. *ICDE*, 1997.

[55] Z. Song and N. Roussopoulos. K-nearest neighbor search for moving query point. *SSTD*, 2001.

[56] Y. Tao, N. Mamoulis, and D. Papadias. Validity information retrieval for spatio-temporal queries. *SSTD*, 2003.

[57] Y. Tao and D. Papadias. The mv3r-tree: A spatio-temporal access method for timestamp and interval queries. *VLDB*, 2001.

[58] Y. Tao and D. Papadias. Time-parameterized queries in spatio-temporal databases. *SIGMOD*, 2002.

[59] Y. Tao and D. Papadias. Spatial queries in dynamic enviornments. *To appear in TODS*, 2003.

[60] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. *VLDB*, 2002.

[61] Y. Tao, D. Papadias, and J. Sun. The tpr*-tree: An optimized spatio-temporal access method for predictive queries. *VLDB*, 2003.

[62] Y. Tao, D. Papadias, and J. Zhang. Cost models for overlapping and multi-version structures. *TODS*, 27(3):299–342, 2002.

[63] Y. Tao, J. Sun, and D. Papadias. Selectivity estimation for predictive spatio-temporal queries. *ICDE*, 2003.

[64] J. Tayeb, O. Ulusoy, and O. Wolfson. A quadtree-based dynamic attribute indexing method. *The Computer Journal*, 41(3):185–200, 1998.

[65] M. Vazirgiannis, Y. Theodoridis, and T. Sellis. Spatio-temporal composition and indexing for large multimedia applications. *Multimedia Systems*, 6(4):284–298, 1998.

[66] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. Lee. Location-based spatial queries. *SIGMOD*, 2003.

[67] C. Chung, J. Min, and K. Shim. APEX: An adaptive path index for XML data. In *ACM SIGMOD*, June 2002.

[68] B. F. Cooper, N. Sample, M. Franklin, G. Hjaltason, and M. Shadmon. A fast index for semistructured data. In *VLDB*, pages 341–350, September 2001.

[69] R. Goldman and J. Widom. DataGuides: Enable query formulation and optimization in semistructured databases. In *VLDB*, pages 436–445, August 1997.

[70] R. Kaushik, P. Bohannon, J. Naughton, and H. Korth. Covering indexes for branching path queries. In *ACM SIGMOD*, June 2002.

[71] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In *VLDB*, pages 361–370, September 2001.

[72] T. Milo and D. Suciu. Index structures for path expression. In *Proceedings of 7th International Conference on Database Theory (ICDT)*, pages 277–295, January 1999.

[73] Haixun Wang, Sanghyun Park, Wei Fan, and Philip S Yu. ViST: A dynamic index method for querying XML data by tree structures. In *SIGMOD*, 2003.

[74] D. Gusfield. Algorithms on strings, trees, and sequences. *Cambridge University Press*, 1997.

[75] J. Yang and W. Wang. CLUSEQ: efficient and efficient sequence clustering, *Proceedings of 19th IEEE Intern. Conf. on Data Engineering (ICDE)*, pp. 101-112, 2003.

[76] J. Han, G. Dong, and Y. Yin. Efficient mining partial periodic patterns in time series database. *Proc. Int. Conf. on Data Engineering*, 106-115, 1999.

[77] J. Yang, W. Wang, and P. Yu. Mining asynchronous periodic patterns in time series data. *Proc. of the 6th ACM SIGKDD Intern. Conf. on Knowledge Discovery and Data Mining*, 275-279, 2000.

[78] J. Yang, W. Wang, and P. Yu. Info-miner: mining surprising periodic patterns. *Proc. of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 395-400, 2001.

[79] M. O. Akinde, O. G. Jensen, and M. H. Böhlen. Minimizing detail data in data warehouses. In *Proc. of the 1998 Intl. Conf. on Extending Database Technology*, pages 293–307, 1998.

[80] K. Amiri, S. Park, R. Tewari, and S. Padmanabhan. DBProxy: A dynamic data cache for web applications. In *Proc. of the 2003 Intl. Conf. on Data Engineering*, pages 821–831, Bangalore, India, March 2003.

[81] K. S. Candan, W.-S. Li, Q. Luo, W.-P. Hsiung, and D. Agrawal. Enabling dynamic content caching for database-driven web sites. In *Proc. of the 2001 ACM SIGMOD Intl. Conf. on Management of Data*, Santa Barbara, California, USA, May 2001.

[82] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26(1):65–74, 1997.

[83] S. Dar, M. J. Franklin, B. Jónsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *Proc. of the 1996 Intl. Conf. on Very Large Data Bases*, pages 330–341, Bombay, India, September 1996.

[84] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science: Special Issue on External Memory Algorithms and Visualization*, A:39–70, 1999.

[85] Ashish Gupta and Inderpal Singh Mumick, editors. *Materialized Views: Techniques, Implementations and Applications*. MIT Press, June 1999.

[86] Q. Luo, J. F. Naughton, R. Krishnamurthy, P. Cao, and Y. Li. Active query caching for database web servers. In *Proc. of the 2000 Intl. Workshop on the Web and Databases*, pages 92–104, Dallas, Texas, USA, May 2000.

[87] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *Proc. of the 2003 ACM SIGMOD Intl. Conf. on Management of Data*, San Diego, California, USA, June 2003.

[88] C. Olston and J. Widom. Best-effort cache synchronization with source cooperation. In *Proc. of the 2002 ACM SIGMOD Intl. Conf. on Management of Data*, Madison, Wisconsin, USA, June 2002.

[89] D. Quass, A. Gupta, I. S. Mumick, and J. Widom. Making views self-maintainable for data warehousing. In *Proc. of the 1996 Intl. Conf. on Parallel and Distributed Information Systems*, pages 158–169, December 1996.

[90] J. Yang and J. Widom. Temporal view self-maintenance in a warehousing environment. In *Proc. of the 2000 Intl. Conf. on Extending Database Technology*, pages 395–412, Konstanz, Germany, March 2000.

[91] J. Yang and J. Widom. Incremental computation and maintenance of temporal aggregates. In *Proc. of the 2001 Intl. Conf. on Data Engineering*, Heidelberg, Germany, April 2001.

[92] K. Yi, H. Yu, J. Yang, G. Xia, and Y. Chen. Efficient maintenance of top-k views. In *Proc. of the 2003 Intl. Conf. on Data Engineering*, pages 189–200, Bangalore, India, March 2003.

[93] I. Lazaridis and S. Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *Proceedings of ACM/SIGMOD Annual Conference on Management of Data (SIGMOD)*, 2001.

[94] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient olap operations in spatial data warehouses. In *Proceedings of Symposium on Spatial and Temporal Databases (SSTD)*, 2001.

[95] J. Yang and J. Widom. Incremental computation and maintenance of temporal aggregates. In *Proceedings of International Conference on Data Engineering (ICDE)*, 2001.

[96] D. Zhang, D. Gunopulos, V. J. Tsotras, and B. Seeger. Temporal aggregation over data streams using multiple granularities. In *Proceedings of International Conference on Extending Database Technology (EDBT)*, 2002.

[97] D. Zhang, A. Markowetz, V. J. Tsotras, D. Gunopulos, and B. Seeger. Efficient computation of temporal aggregates with range predicates. In *ACM International Symposium on Principles of Database Systems (PODS)*, 2001.

[98] D. Zhang, V. J. Tsotras, and D. Gunopulos. Efficient aggregation over objects with extent. In *ACM International Symposium on Principles of Database Systems (PODS)*, 2002.