What's Next? Sequence Queries

Raghu Ramakrishnan *
Michael Cheng † Miron Livny ‡ Praveen Seshadri §
Computer Sciences Department,
University of Wisconsin-Madison, WI 53706, U.S.A.
{raghu,mcheng,miron,praveen}@cs.wisc.edu.

Abstract

A large (and increasing) number of applications require the ability to manipulate sequences of data. While some of these applications use very specialized forms of sequences—e.g., video sequences—there is much to be gained by developing a general framework for managing and querying large sequence data sets. Important target domains include scientific modelling, stock market and financial analysis, analysis of trace data, temporal queries, and exploratory data analysis or 'data mining'. Current database systems cannot address the problems in these domains satisfactorily, and developing suitable systems and techniques is an important and interesting research problem.

In this paper, our goal is primarily to motivate the importance of sequence query processing, and to discuss why current systems are unsatisfactory. We illustrate some of the technical challenges in developing suitable systems through examples. We also outline some of the work that we are doing in this area in the SEQ project at Wisconsin. This includes a model of sequence data and a framework for the optimization of sequence queries based on several novel techniques, and a framework for *visual* querying of sequence data. These ideas are currently being implemented in the `SEQ` and `DEVISE` systems.

1 Introduction

Many real life applications manipulate data that is inherently sequential, and deal with such large volumes of data that the functionality of a database management system is needed. Such data is most naturally viewed and queried in terms of a sequence abstraction and is often physically stored as a sequence. It is therefore desirable that the underlying database system meet the following requirements:

- It should allow sequences to be queried in a *declarative*, non-operational manner, utilizing the ordered semantics of the data.
- It should take advantage of the opportunities for query optimization that arise due to sequentiality.

relational databases fall short on both counts. A fundamental limitation is that data collections are treated as sets, not sequences. This limitation is reflected in the design of query languages like SQL, and also in the optimizers and run-time engines for relational systems. Consequently, expressing sequence queries is tedious, and evaluating them is inefficient.

We therefore argue that extensions must be made to the relational model in each of the following areas in order to deal effectively with sequence data and queries:

- *The data model.* We must support the abstraction of a *sequence* of records, in addition to the abstraction of a relation as a *set* of records.
- *The query language.* We must provide language constructs that utilize notions like 'next' and 'previous' that are inherent in sequence queries to allow easy formulation of such queries. This can probably be accomplished by extending SQL in a straightforward way.

  An aspect of sequence querying that deserves special mention is the fact that very natural *visual* front-ends can be designed for browsing sequence data. Indeed, very effective use can be made of visualization techiques for sequences, especially for *exploratory data analysis*, where we typically look for interesting 'patterns' that are not fully understood *a-priori.*
- *The optimizer and run-time system.* The optimizer must explore query plans that utilize the sequentiality in queries effectively. The run-time system must

*Raghu Ramakrishnan's work was supported by a Packard Foundation Fellowship in Science and Engineering, a PYI Award with matching grants from DEC, HP, IBM, Tandem and Xerox, NSF grant IRI-9011563 and NASA grant NAGW-3921.

†Michael Cheng's work was supported by an IBM Fellowship.

‡Miron Livny's work was supported by NASA grant NAGW-3921.

§Praveen Seshadri's work was supported by IBM Research Grant 93-F153900-000 and an IBM Fellowship.

provide evaluation strategies that are well-suited for sequence queries.

In this paper, our goal is primarily to motivate the importance of sequence query processing, and to discuss why current systems are unsatisfactory. We illustrate some of the technical challenges in developing suitable systems through examples. We also outline some of the work that we are doing in this area in the SEQ project at Wisconsin. This includes a model of sequence data and a framework for the optimization of sequence queries based on several novel techniques, and a framework for *visual* querying of sequence data. These ideas are currently being implemented in the `SEQ` query processor and `DEVISE` visualization systems, which are being developed as part of the SEQ project. For further details, we refer the reader to [SLR95], [SLR94] and [CLR95].

We motivate sequence data management and queries in Section 2, present a sequence data model and discuss language issues in Section 3, consider sequence query optimization in Section 4, and discuss sequence data visualization in Section 5. We conclude with a brief discussion of related work.

2 Motivation for Sequence Queries

¿From a personal standpoint, our motivating problem domains are *scientific modelling*, *stock market analysis*, *trace data analysis*, and *data mining*. We discuss scientific modelling in some detail, and the others more briefly.

2.1 Scientific Modelling

We are working with scientists in the Earth Sciences to develop a package of query and visualization modules for primary sequence analysis combines remote observations from satellites with biophysical measurements on the ground. Data from a large, intensive field experiment (First ISLSCP Field Experiment, or FIFE) is presently available in database form at Goddard Space Flight Center and is being used to evaluate climate and weather models and explore ways to make ecological inferences. For example, our collaborator in Earth Sciences, Prof. John Norman, has developed a simulation model that is being used to relate remote observations of infrared surface temperature to energy exchanges at the Earth's surface. This same model is being used to combine remote shortwave and thermal measurements to the carbon budget of a native prairie. Recently another large field experiment has begun in the Boreal forest in Canada (BOReal Ecosystem Atmosphere Study, or BOREAS) and data should begin becoming available in early 1995. These analyses rely heavily upon sequence data generated through periodic observations of a number of variables such as soil characteristics, rainfall, cloud cover, and so on.

A data base query language capable of sequence processing would be extraordinarily useful for identifying, extracting and evaluating useful data from large Earth science experiments. First, sequence processing could identify missing or erroneous data sequences and interpolate in time or space with appropriate algorithms to fill in the data set, which must be done for model studies. Second, sequence processing could identify or extract data that meets a complex combination of coincident criteria, which may be required to test models or develop algorithms. Such sequence processing would be particularly valuable with FIFE or BOREAS data sets because data is collected and stored in time or space sequences. Third, sequence processing would permit efficient comparison of model predictions with remote measurements by permitting queries that would identify occasions when the correspondence between predictions and measurements either met or did not meet some appropriate criteria. Since model output is not incorporated into FIFE or BOREAS, some means for accomplishing this comparison on a second database system that contains the model output may be necessary.

The time spent finding and checking data in a data base system is wasted time from the point of view of a disciplinary scientist. A hundred or more scientists may be using a system like FIFE or BOREAS. Since we expect our work on sequence query analysis to be applicable to other sequence data sets and applications as well, it is clear that a more appropriate database query language for sequences, along with good support for sequence data visualization, could be very valuable.

2.1.1 Example Queries

We now present several examples of sequence queries that could arise in scientific modelling applications.

> *On months when a volcano erupts, show the mean sunspot activity over the last two solar cycles.*

This query involves multiple sequences. The volcano sequence has a granularity of a month. The sunspot activity sequence probably needs to be derived from sequences that record various levels of solar activity. The granularity of the aggregation calendar (for the mean) has a user-defined semantics (in this case, a period of 7 years).

Consider the following variant of the previous query:

> *On months when a volcano erupts, show the mean sunspot activity for each of the last two solar cycles.*

For the person querying the system, this appears to be hardly more complex than the previous query, and this should be reflected in the way the two queries are actually formulated. (This goal is surprisingly hard to achieve!)

Next, we show that there is a class of natural queries that are elegantly expressed recursively, and cannot be expressed without recursion. Recursive processing is beyond the capabilities of most existing database query languages. We believe, however, that this is fundamental to the evaluation of many interesting sequence queries. Consider the following example:

> *Determine the maximal decreasing runs (a subsequence of decreasing values) in the observed daily temperatures.*

Here, we do not *a priori* know the maximum sizes of the runs. This query cannot be expressed in SQL.

We conclude our set of examples with a more complex scenario that is of everyday concern to Prof. Norman's group. Extensive manual effort is currently required to extract useful data from the FIS or BORIS data bases. For example, to compare a prediction of infrared temperature from a soil-plant-atmosphere model (named Cupid) with aircraft and satellite observations, input data (soil, vegetation and weather data) required by the model must be obtained from the database subject to the constraint that the site was cloud free at the time of satellite and aircraft observation. Furthermore, when a cloud-free period is obtained all the model input data must be available (weather data is collected on half hour intervals, surface soil moisture measured daily, leaf area and deep soil moisture measured weekly with the time of observation varying from station to station) from the previous day of rain to the time of coincidence of clear sky and satellite overpass either by direct measurement or interpolation in time or space. Extracting the appropriate data from the data base is itself time consuming, but it does not require as much time as just finding an appropriate time sequence that meets the criteria, which can take months. We note that SQL is wholly inadequate for tasks such as this.

Missing or erroneous data are among the major complicating factors that make this selection process difficult. After the inputs required by the model have been extracted, the model is run; then model predictions of infrared temperatures are compared with measurements from aircraft or satellites, which also need to be extracted from the data system. Typically such comparisons involve time sequences of predictions and measurements. For example, predicted and measured infrared temperature would be plotted as a function of time; they may agree at some times and disagree at other times. On other occasions the two curves may just be shifted in time by some fixed phase. Alternatively they may agree at high wind speeds and disagree at low wind speeds.

2.2 Other Application Domains for Sequence Data Analysis

In our other Computer Science related research, we generate extensive trace and simulation data from experiments using a variety of software, including the Condor and PVL cluster computing packages, and the DeNet simulation package. Our everyday research involves detailed analysis of this trace data, and we are already using prototype versions of `SEQ` and, especially, `DEVISE` to assist in this work. Our experience with these tools is extremely encouraging. In previous work, we have built a tool for analysis of stock market data called Mimsy, on top of the Coral deductive database system. Examples of queries that can be posed using Mimsy include "Show the close of IBM on all days when it rose by 10%", "Show all increasing runs in IBM", "Show the close of IBM whenever IBM crosses DEC, and the weekly move of the Dow is over 10%". We believe that more powerful versions of such a tool can be developed using `SEQ`; this is an area for further research, as our implementation of `SEQ` becomes more mature.

A very interesting and currently popular area of database research is *data mining*, which has also been studied in various fields such as AI (where similar issues have been investigated by *machine learning* and *knowledge discovery* researchers) and statistics (where it comes under the subfield called *exploratory data analysis*). Many of the algorithms proposed for problems such as *mining for association rules* [AIS93] involve sequence data manipulation, and we believe that `SEQ` provides an excellent platform for implementing such algorithms. `SEQ` also offers a promising approach for implementing many of the functionalities offered by statistical data analysis packages such as S in an environment where the data is disk-resident.

In fact, the combination of `SEQ` and `DEVISE` offers a powerful platform for analysing large data sets for interesting *patterns*, where we do not know exactly what we are looking for in advance, or the patterns are too complex to specify algorithmically. We are currently investigating how `SEQ` and `DEVISE` can be combined and enhanced for such data mining applications.

Although we are not working directly upon temporal database queries, they represent an obvious application domain for sequence queries. Indeed, there is no lack of application domains that can benefit from increased support for sequence data and queries.

3 A Sequence Data Model

In this section, we present a sketch of a model for sequence data that is proposed in [SLR95]. There are two important kinds of sequences in the model:

- *Base sequences:* A base sequence is specified by an explicit set of records, each containing a *position* attribute. For simplicity, assume that a position is essentially a positive integer. All positions that do not explicitly map to a record are assumed to map to the Null record.

- *Derived sequences:* A derived sequence is defined by a *sequence operator*. Intuitively, this is similar to the definition of views in a relational database using relational operators. The next section describes operators that are used in the definition of such sequences.

We note that we also model constants as sequences so that the constructs of the model deal uniformly with sequences.

3.1 Sequence Operators

All operators in our model operate on one or more sequences and produce a single 'derived' sequence. The derived sequence defined by a sequence operator is called the *output sequence* of the operator, while the sequences used in this definition are called the *input sequences*. The *arity* of an operator is the number of input sequences. For an operator Op with output sequence S_{out} and input sequences S_1 through S_n, for all positions i, $S_{out}(i) = Op(S_1, S_2, ..., S_n, i)$. The collection of operators includes:

Simple Unary Operators: The output sequence S_{out} of a unary operator Op is defined by $S_{out}(i) = Op(S_{in}, i)$.

- A *Selection* applies a selection predicate σ to the input sequence record r at each position i. If $(\sigma(S_{in}(i), i))$, $Op(S_{in}, i) = S_{in}(i)$, else $Op(S_{in}, i) = Null$. Note that it does not matter if a predicate evaluates to true or false on a Null input record.

- A *Projection* π projects a subset of the attributes of the input sequence record r at each position i. Any projection of a Null record is a Null record. $Op(S_{in}, i) = \pi(S_{in}(i))$.

- A *Positional Offset* takes as a parameter an integer specifying an offset l. For each record r at position i in the input sequence, $Op(S_{in}, i) = S_{in}(i + l)$. Intuitively, this operator "shifts" the input sequence by the number of positions specified by the offset.

- A *Value Offset* takes as a parameter an integer specifying an offset l. For each record r at position i in the input sequence, $Op(S_{in}, i1) = S_{in}(i)$ iff the number of non-empty positions between i1 and i = l. If the input sequence has no empty positions, this is identical to the positional offset operator. Examples are the Previous operator that has an offset of -1, and the Next operator with an offset of +1.

Aggregate Unary Operators: The Aggregate Operators are unary operators that are defined by two functions. One function *agg_pos(i)* selects a set P of positions for each position i. The other function *agg_func* is an aggregate function over the records in $S_{in}(p)$ at the positions p in P. $Op(S_{in}, i) = agg\_func(\{S_{in}(p) \mid p \in agg\_pos(i)\})$. For example, the moving 3-position average of a sequence would have agg_func $\equiv$ Avg and $agg\_pos(i) \equiv \{p \mid i \geq p \geq i - 2\}$. A special case is where the agg_pos function is always true (thereby selecting all positions). The aggregate functions allowed are Avg, Count, Min, Max and Sum. For all these aggregates, Null records in the inputs are ignored if there is at least one non-Null record; else the output is a Null record.

Compose Operator: *Compose* is a binary operator that composes the records r1 and r2 of the two input sequences at each position i. $Op(S_{in1}, S_{in2}, i) = S_{in1}(i).S_{in2}(i)$. If either of the records is a Null record, the output is a Null record. A Compose operator is also referred to as a *positional join* operator. Note that in an implementation, the Compose operator would probably allow the specification of additional "join" predicates.

3.2 Sequence Queries

The sequence operators that we presented earlier can be regarded as a way to specify queries. Although they are at the level of relational Algebra, rather than SQL, they form a useful starting point for the design of a sequence query language.

A *sequence query* is an acyclic graph of operators such that the inputs of each operator are either the output of another operator or are base or constant sequences. The output of a query is the output sequence of the root operator of the query graph. As an example of the use of the operators described, we present a graphical representation of the query of Example 3.1 in Figure 1.

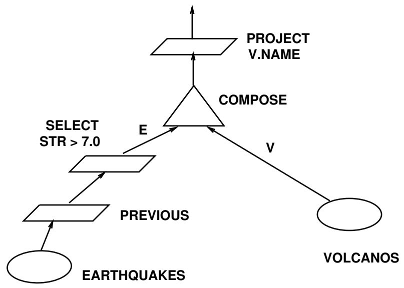

Figure 1: Motivating Example

3.3 Sequence Query Languages

Consider the following motivating example.

Example 3.1 *A weather monitoring system records information about various meteorological phenomena. There is sequentiality in the occurrence of these phenomena; the various meteorological events are sequenced by the time at which they are recorded. A scientist asks the query:* "For which volcano eruptions was the strength of the most recent earthquake greater than 7.0 on the Richter scale?". *It is a seemingly innocuous query, but it is difficult to even* express *in a relational query language like SQL. One possible attempt uses a nested sub-query as follows:*

```
SELECT V.name
FROM Volcanos V, Earthquakes E
WHERE E.strength > 7.0 AND
      E.time = (SELECT  max(E1.time)
                FROM  Earthquakes E1
                WHERE E1.time < V.time)
```

Other ways of expressing the query in SQL are not particularly simpler. Clearly, there is a need for more natural languages in which to express sequence queries. SQL is not good enough, and the algebra that we presented earlier is too low-level and operational for end users.

We believe that SQL can be adapted for sequence queries if we treat the underlying relations as *sequences*, i.e., as sets of records that are *ordered* according to some sequencing attribute (whose semantics is understood by the DBMS, and 'built-into' the language's idioms).

We sketch the idea through simple examples.

Example 3.2 *This query selects some observations from a sequence of observations called* AFS*:*

```
SELECT A.UserCPU, A.SystemCPU, A.Disk2
FROM AFS AS A
WHERE A.UserCPU > 1.0 AND A.Disk2 > 5.0
```

The result is also a sequence (with Nulls used for observation points that are not selected)!

Example 3.3 *This query illustrates how the sequence property of the underlying data can be utilized effectively for query specification:*

```
SELECT D.close, D.open
FROM OFFSET 2 (IBM) AS I, DEC AS D
WHERE I.close < D.close
```

IBM and DEC are two base sequences; IBM has (at least) attribute close, and DEC has (at least) attributes open and close, and each relation contains one tuple per day. Such data is available on a daily (or even higher frequency) basis for several thousand securities over the past three decades. OFFSET 2 is a sequence operator that takes the IBM sequence as input and returns the sequence shifted by two positions as the output. This is the positional offset *in the algebra of sequence operators that we presented earlier. The result of this query is the close and open values of DEC for each day when the close of DEC is greater than the close of IBM two days later. There is an* implicit join *on the date attribute (which is the sequencing attribute) between IBM and DEC; this is followed by the OFFSET operator and then the selection in the WHERE clause is applied.*

Example 3.4 *The motivating volcano query can be expressed as follows:*

```
SELECT V.id, E.id
FROM Volcanos AS V, PREVIOUS(Earthquakes) AS E
```

PREVIOUS is value offset *in the algebra of sequence operators that we presented earlier.*

4 Sequence Query Optimization

Our goal in this section is to motivate the need for new implementation techniques to deal with sequence queries.

Example 4.1 *Consider the volcano example again, expressed in SQL:*

```
SELECT V.name
FROM Volcanos V, Earthquakes E
WHERE E.strength > 7.0 AND
      E.time = (SELECT  max(E1.time)
                FROM  Earthquakes E1
                WHERE E1.time < V.time)
```

A conventional relational query optimizer as described in [SMALP79] would probably generate the following query evaluation plan. For every Volcano tuple in the outer query, the sub-query would be invoked to find the time of the most recent earthquake. Each such access to the subquery involves an aggregate over the entire Earthquake relation. The time of the most recent earthquake is used as a join condition to probe the Earthquake relation in the outer query. Finally, the selection condition to check that the strength is greater than 7.0 would be applied. Even the knowledge that the Earthquakes and Volcano relations are sorted by time would not significantly alter the query plan. A more efficient evaluation strategy does however exist; the two sequences can be scanned in lock step (similar to a sort merge join). The most recent earthquake record scanned can be stored in a temporary buffer. Whenever a volcano record is processed, the value of the most recent earthquake record stored in the buffer is checked to see if its strength was greater than 7.0, possibly generating an answer. This query can therefore be processed with a single scan of the two sequences, and using very little memory. The key to such optimization is the sequentiality of the data and the query.

Example 4.2 *Another query that is slightly more complex asks for those volcano eruptions for which the most recent Earthquake* in the same region *had a strength greater than 7.0. This could be expressed in SQL as follows:*

```
SELECT V.name
FROM Volcanos V, Earthquakes E
WHERE E.strength > 7.0 AND E.time =
    (SELECT  max(E1.time)
     FROM  Earthquakes E1
     WHERE E1.time < V.time AND E1.region = V.region)
```

It is more difficult to apply the same optimization technique to the second query, because for every volcano eruption, the most recent earthquake in the same region needs to be determined. Until the volcano record is retrieved, it is not known which earthquake record should be buffered. This is an example of a *correlation* in the nature of the query. These two examples bring out two distinct points:

1. Sequence queries present opportunities for optimization that require us to reason directly about the sequentiality.

2. Detecting when such opportunities exist (or do not exist) is not easy.

Thus, sequence queries present new challenges for query optimization. The goal is to effectively utilize a number of unique opportunities for query optimization that are not available in relational queries. As in a standard relational query optimizer, we can *transform* a query into an equivalent query, using equivalence rules for expressions involving sequence operators. Such optimizations are *independent* of the actual data in the input sequences. In addition, we can use meta-information about the input sequences to perform various query optimizations.

The goal of query optimization is to choose a query evaluation plan of low estimated cost. As in a typical SQL optimizer, a space of possible plans must be searched, using equivalence preserving transformations and heuristics to define the search space, and using statistics to estimate the cost of each plan. A good heuristic is to try and generate plans in which each base sequence is scanned just once, for example. We will not discuss optimization issues further; the reader is referred to [SLR94], where a detailed optimization framework is presented. This forms the basis for our ongoing implementation of the `SEQ` query processing system.

5 Visual Sequence Query Interfaces

An important component of the SEQ project is the development of a framework for *visual querying* of sequences, which is also being implemented in the `DEVISE` tool. A visualization interface helps scientists to understand their data through the use of graphics. This power is invaluable when a scientist is searching for complex patterns, which are often easier to detect visually than through a detailed specification. It is also likely to be very useful when dealing with missing or erroneous data, since such anomalies are typically easy to notice in a visual rendering of a sequence of values.

However, a simple plot of sequence data is much less useful than a presentation that faithfully captures the semantics of the sequence as understood by the scientist. In order to exploit the full power of graphics, therefore, the visualization interface has to enable the scientist to decide *how* the data is to be presented graphically, *what* portions should be viewed, and *where* the graphics should be placed on the display. Our model for achieving this is that a user-defined *graphics mapping* allows the interface to map the *source* data to *graphics* data, and that it is then the responsibility of the interface to provide efficient access to selected subsets of the graphical data and to display them on the screen. A visualization interface, therefore, can be viewed as a data management system that operates on the graphical presentation of the data.

To illustrate what we expect from a visualization interface assume that we have a data set that consists of two streams of sensor records, SD1 and SD2. The streams were generated by identical sensors at two different locations. Each sensor record is time-stamped and has a temperature, a pressure, and a rainfall attribute. A scientist who is interested in the relationship between these three attributes may choose to map each record with a rain fall of more than half an inch to a square graphical symbol and all other records to a triangle symbol. The color of the symbol is determined by the pressure attribute whereas the time and temperature determine the x and y co-ordinates of the symbol respectively. The color of the line connecting two adjacent records in the stream is determined by the temperature difference between the two records.

Given this graphics mapping, the scientist may ask to view all the red (high pressure) squares of SD1 in a given range of x (time) values and all the blue (low pressure) triangles of SD2 that are connected by green (small temperature change) lines in a given range of y (temperature) values. These are two *select* operations on the graphics data and it is up to the visualization interface to decide how to generate the requested visual presentation. Note in particular that the data to be displayed may well be too large to fit into available main memory. It must be paged in as needed, while updating the displayed data in an efficient way.

5.1 Our Visualization Framework

Our research in this area centers on providing a framework that can be used to efficiently display large amounts of data as one or more overlapping 2D plots. The framework provides the abstractions whereby programmers can specify how data is to be visualized, and how users interact with the plots (e.g., through operations such as *scroll* or *zoom*). However, the task of efficiently displaying the plots is handled by the visualization system, and is transparent to the user.

We view data visualization a 3-step process:

$$\text{TData} \xrightarrow{mapping} \text{GData} \xrightarrow{drawing} \text{Visual}$$

TData is the textual data provided by the user. It consists of attributes stored in records. Starting with the TData, the programmer specifies how it is to be mapped to its graphical representation, the GData. Each record of the TData is mapped to a record of the GData, where each GData record consists of graphical attributes location, size, shape, color, pattern, and orientation. The framework is responsible for rendering GData in a visual form, currently a bitmap.

¿From a user's standpoint, there is a first phase that is best thought of as *defining a visual query*. In this phase, the user lays out the screen by arranging one or more *windows*, each of which contains *views* of one or more sequences. Views can optionally be *linked*; all linked views are scrolled/zoomed synchronously when any one of them is scrolled/zoomed. There are several alternatives for customizing just how each sequence is to be viewed, or represented visually on the screen. This phase proceeds largely through choices made using a point-and-click interface. (Some programming and recompilation may be required to develop sophisticated visualizations that fully utilize the flexibility of the system, but an ongoing task is to minimize such effort, and the need to go outside the graphical query definition capability.)

Once a satisfactory 'visual query' is defined, it can be saved and given a name, much like one defines views in a relational DBMS. In the second phase, we can bring up a visual query and run it against a given data set of sequences. Just like relational views, the definition of a visual query is independent of the actual data instance. Unlike a relational query, a visual query is a dynamic object: we can 'browse' through the result of a visual query by scrolling and zooming, for instance.

6 Related Work

Since temporal sequences are among the most commonly occuring sequences in real life, there has been much research on temporal databases. We refer the reader to [Soo91] for a bibliography of recent work. Most of the research has concentrated on temporal models and languages[Gad86, CC87, SS88, NA89, WD92, Sno87]. Much of this research has focused on extending existing data models to support temporal data, or extending existing query languages to allow temporal queries to be expressed. There has been some work on query optimization based on such models[GS89, LM93, NG93]. Our approach to sequences presented in this paper takes a strongly 'positional' view of sequences, as opposed to the models mentioned above. In this aspect, it has been influenced by the model of Time Sequences in [SS87, SK86].

Research into sequence data in contexts other than temporal data include [GJS92], [Ric92], and [SP90]. Finally, a number of specialized access structures proposed for temporal data are relevant to sequence query processing (e.g., [EWK90, LS89, RS87]).

On the visualization side, software packages such as AVS, IDL, and Khoros are quite successful at visualizing data from physical phenomena such as data from weather patterns, terrain maps, or medical images. However, they are currently not suitable for visualizing sequence data originating from complex systems. Such data can be gathered from program traces, discrete event simulations, or remote sensors. What makes such data difficult to visualize is that it can be very large in size, and there can be many sets of data to be compared. The simplest visualization of such data is as a 2D plot. Systems such as AVS and Khoros have limitations on the number of points to be displayed, and the use of virtual memory results in excessive paging. Flexibility in arranging displays is also quite limited.

7 Conclusions

Sequence data management, querying and visualization have important applications, and present problems that current database and visualization packages cannot address satisfactorily. Developing techniques and systems to support sequence data applications is an important direction for database research. We are addressing this problem in the SEQ project at Wisconsin, and are implementing our ideas in two systems called `SEQ`, which is a sequence query processor, and `DEVISE`, which is a visualization tool. Taken together, they provide a powerful and flexible way to explore, display and analyze large and complex sequence data sets.

References

[AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules Between Sets of Items in Large Databases. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 207–216, Washington, DC, June 1993.

[CC87] James Clifford and Albert Croker. The Historical Data Model and Algebra Based on Lifespans. In *Proceedings of the International Conference on Data Engineering*, pages 528–537, 1987.

[CLR95] Michael Cheng, Miron Livny, and Raghu Ramakrishnan. Visual Analysis of Stream Data. In *Proceedings of the IS&T/SPIE Conference on Visual Data Exploration and Analysis, San Jose, CA*, February 1995.

[EWK90] Ramez Elmasri, Gene Wuu, and Yeung-Joon Kim. The Time Index : An Access Structure for Temporal Data. In *Proceedings of the International Conference on Very Large Databases*(*VLDB*), pages 1–12, 1990.

[Gad86] S.K. Gadia. Towards a Multihomogenous Model for a Temporal Database. In *Proceedings of the International Conference on Data Engineering*, pages 390–397, 1986.

[GJS92] N.H Gehani, H.V. Jagadish, and O. Shmueli. Composite Event Specification in Active Databases: Model and Implementation. In *Proceedings of the International Conference on Very Large Databases(VLDB)*, pages 327–338, 1992.

[GS89] Himawan Gunadhi and Arie Segev. Event-join Optimization in Temporal Relational Databases. In *Proceedings of the Fifteenth International Conference on Very Large Databases (VLDB), Amsterdam, Netherlands*, 1989.

[LM93] Cliff T.Y. Leung and Richard R. Muntz. *Stream Processing: Temporal Query Processing and Optimization*, chapter 14. Benjamin/Cummings, 1993.

[LS89] David Lomet and Betty Salzberg. Access Methods for Multiversion Data. In *Proceedings of ACM SIGMOD International COnference on Management of Data*, pages 315–324, 1989.

[NA89] S.B. Navathe and R. Ahmed. A Temporal Relational Model and a Query Language. *Information Sciences*, 49:147–175, 1989.

[NG93] Sunil S. Nair and Shashi K. Gadia. Algebraic Optimization in a Relational Model for Temporal Databases. In Richard Snodgrass, editor, *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, pages 390–397, Arlington, Texas, 1993.

[Ric92] Joel Richardson. Supporting Lists in a Data Model. In *Proceedings of the International Conference on Very Large Databases(VLDB)*, pages 127–138, 1992.

[RS87] Doron Rotem and Arie Segev. Physical Organization of Temporal Data. In *Proceedings of the International Conference on Data Engineering*, pages 547–553, 1987.

[SK86] Arie Shoshani and Kyoji Kawagoe. Temporal Data Management. In *Proceedings of the Twelfth International Conference on Very Large Databases (VLDB), Kyoto, Japan*, pages 79–88, 1986.

[SLR94] Praveen Seshadri, Miron Livny, and Raghu Ramakrishnan. Sequence Query Processing. In *Proceedings of ACM SIGMOD '94 International Conference on Management of Data, Minneapolis, MN*, May 1994.

[SLR95] Praveen Seshadri, Miron Livny, and Raghu Ramakrishnan. SEQ: A Model for Sequence Databases. march 1995.

[SMALP79] Patricia G. Selinger, D. Chamberlin M. Astrahan, Raymond Lorie, and T. Price. Access path selection in a relational database management system. In *Proceedings of ACM SIGMOD '79 International Conference on Management of Data*, pages 23–34, 1979.

[Sno87] Richard Snodgrass. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[Soo91] Michael D. Soo. Bibliography on Temporal Databases. *ACM SIGMOD Record*, 20(1):14–23, March 1991.

[SP90] D. Stott Parker. *Stream Data Analysis in Prolog*, chapter 8. MIT Press, 1990.

[SS87] Arie Segev and Arie Shoshani. Logical Modelling of Temporal Data. In *Proceedings of ACM SIGMOD '87 International Conference on Management of Data, San Francisco, CA*, pages 454–466, 1987.

[SS88] Arie Segev and Arie Shoshani. The Representation of a Temporal Data Model in the Relational Environment. In *Proceedings of the 4th Conference on Statistical and Scientific Database Management*, pages 39–61, June 1988.

[WD92] Gene Wuu and Umeshwar Dayal. A Uniform Model for Temporal Object-Oriented Databases. In *Proceedings of the International Conference on Data Engineering*, 1992.