Approximating Data in Constraint Databases*

Rui Chen, Min Ouyang, Peter Revesz

Department of Computer Science and Engineering
University of Nebraska-Lincoln, Lincoln, NE 68588, USA

Abstract. Approximate representation of any spatio-temporal variable, by some interpolation function, is necessary when it is measured only sporadically. This paper argues that the approximate representation can be captured by a constraint database. Since constraint databases can be queried via standard query languages – such as relational algebra, SQL and Datalog – this provides an immediate benefit for flexible querying of the data. We propose a concrete system that implements a version of this approach. We also add beyond the standard queries new ones like cartogram similarity queries and an advanced graphical user interface with 3-D animation of GIS-based data.

1 Introduction

Many databases contain (spatio)temporal data that change continuously with time but are measured and recorded only sporadically. For example, population and various other census data in the United States is recorded only every ten years. Different weather and environmental stations throughout the world may be measuring and reporting data like air temperature, precipitation, wind direction, wind speed and levels of different air or water pollutants with different frequencies and regularities.

It is obvious that all these spatio-temporal data cannot be available for all locations at all times. If we are interested in the value of a spatio-temporal variable at a particular time, then we have to somehow approximate that value based on some interpolation from the available data.

The interpolation could be done at two different levels. One approach is to represent the measured data in a standard relational database. Then the relational database can be embedded in a high-level computer program that retrieves the measurements, interpolates them and does other calculations. This approach may be a workable one for some scientists who are advanced programmer or who have such help readily available. It is not feasible for average users.

An alternative approach, that we advocate in this paper, is to perform the interpolation at the time of the data entry, that is, the data should be stored as a constraint database [7, 11, 16], where the constraints are parametric functions of time that interpolate the data. This approach is advantageous because it is possible to build powerful database systems (for example, CCUBE [1], DEDALE [6]

* This work was supported in part by NSF grants IRI-9625055 and IRI-9632871 and by a Gallup Research Professorship.

and MLPQ [15]) that can be queried by standard relational database query languages, such as relational algebra, SQL and Datalog. Also, the enhanced MLPQ [14] has the ability to display the results with color bands according to the associated attribute values. This enables a potentially much wider range of users to use the database.

Applications of constraint database systems were until now severely limited to a few well-understood areas of constraint representation, for example, GIS where convex polygonal areas were represented as conjunctions of linear inequality, i.e., half-plane intersection, constraints. Our work on interpolation functions as a natural source of constraint data opens up a range of uses of constraint databases beside these narrow focus applications.

It is very important to present the data to a user in a form that is easily understandable. Many current constraint database systems have a poor graphical user interface. Probably MLPQ/GIS [8] has the most advanced user interface that allows a number of iconic queries, including the option to ask the system to show an animation of a 2-D object (a moving polygon).

In this paper, we describe an advanced GIS-oriented user interface that can animate in 3-D various spatio-temporal variables (distributed over spatial cells, for example, the U.S. states). Such an animation has a potential to reveal many interesting features to a user that would be hard or impossible to notice otherwise. The user interface also allows a number of new queries. For example, we define similarity queries over cartograms. A similarity query could be for instance the following: given a precipitation map of the United States for March 2000, find among all the other monthly precipitation maps in the past 40 years those where the precipitation was most similar to the given map.

The rest of the paper is structured as follows. Section 2 describes the algorithms for data input and transformation. Section 3 presents the algorithms for the update on piecewise linear functions. Section 4 introduces several kinds of algebraic operators and queries. Section 5 discusses 3-D animation. Finally, Section 6 concludes with some possible directions for future works.

2 Data Input and Transformation

In this section we describe how the input data of measurements can be transformed into a constraint database representation by using various interpolation functions. In particular, we present a transformation method in Section 2.1 based on a linear interpolation function.

We also analyse the correlation between the interpolation function obtained by this method and the original data using as a test case the data obtained from the National Climatic Data Center. The correlation depends on the values of two parameters: the average error threshold and the maximum error threshold, denoted by Φ and Ψ, respectively. In general, the lower Φ and Ψ are the better the correlation is with the original data but the piecewise linear interpolation function will need more pieces. Hence these parameters allow the user to control the trade-off between the accuracy of the approximation and the required storage space.

2.1 The Piecewise Linear Function Transformation Method

Given a set of spatiotemporal data, where the third dimension (called z later) could stand for any property associated with that point, this section will show how to transform a sequence of z values into a piecewise linear function for each spatial point.

Example 1. Suppose there are four weather stations 1 to 4 located in $(10, 20)$, $(20, 40)$, $(50, 25)$ and $(30, 10)$ respectively as shown in Table 1, where SN stands for station number and X, Y the coordinates of the location. Each station has a group of temperature data at five corresponding different moments as shown in Table 2, where t_1 to t_5 columns are the temperatures at the time t_1 to t_5 respectively.

SN	**X**	**Y**
1	10	20
2	20	40
3	50	25
4	30	10

Table 1. The locations of four weather stations

SN	$\mathbf{t_1}$	$\mathbf{t_2}$	$\mathbf{t_3}$	$\mathbf{t_4}$	$\mathbf{t_5}$
1	75	77	86	87	90
2	70	72	75	80	85
3	80	86	81	80	78
4	85	83	81	78	76

Table 2. The temperatures of the four weather stations

Table 1 stores the spatial information while Table 2 stores the z values related to time. In Example 1 z represents the temperature. For simplicity, we name the data in Table 1 *spatial data set* and in Table 2 *temporal data set.*

A piecewise linear function is a set of linear functions with only one parameter, the time t. For each linear function, the domain of t is constrained within a definite extent, which is non-overlapping with the extents of other linear functions. The following description expresses the idea of the transformation: try to include as many points as possible into one piecewise linear function without exceeding the prescribed approximation error thresholds.

Suppose there are $n > 1$ time-value pairs for a given point:

$$(t_1, z_1), (t_2, z_2), ..., (t_n, z_n),$$

where $t_1, t_2, ..., t_n$ stand for the points of time, $z_1, z_2, ..., z_n$ the corresponding z values, and $t_1, t_2, ..., t_n$ are all distinct and in an increasing order.

For any two pairs (t_b, z_b) and (t_e, z_e) the linear function can be expressed as Formula (1).

Definition 1. A piece linear function $z_{b,e}$ is the function:

$$z_{b,e}(t) = \frac{z_e - z_b}{t_e - t_b}(t - t_b) + z_b \tag{1}$$

where t_b and t_e are the lower and the upper bounds of the time interval adapted to this function, i.e. $t_b \leq t \leq t_e$.

We use two parameters to restrict the two kinds of interpolation errors resulted from Formula (1).

1. **Average Error Threshold Φ for Each Piece:** We use the average error threshold (denoted by Φ) to control the average approximation error for each piece of the piecewise linear function.
   The average error for each piece (denoted by ϕ) is defined in Formula (2).

$$\phi_{b,e} = \left(\Sigma_{i=b+1}^{e-1} |z_{b,e}(t_i) - z_i|\right) / (t_e - t_b) \tag{2}$$

2. **Maximum Error Threshold Ψ for Each Time Point:** We use the maximum error threshold (denoted by Ψ) to control the approximation error for each time point.
   The error for each time point (denoted by ψ) is defined in Formula (3).

$$\psi_{b,e}(t_i) = |z_{b,e}(t_i) - z_i| \tag{3}$$

This leads to the following simple transformation algorithm with the use of the two interpolation error thresholds Φ and Ψ.

PIECEWISE LINEAR INTERPOLATION ALGORITHM:

Input: A temporal data set with n time-value pairs $(t_1, z_1), \ldots, (t_n, z_n)$.
Φ the average error threshold, and
Ψ the maximum error threshold in the approximation.
Output: A piecewise linear function.
Local Vars: The b and e are integer variables that denote if the piecewise linear interpolation function should interpolate by one piece the sequence of temporal data $(t_b, z_b), \ldots, (t_e, z_e)$.
The *one_error*, *total_error*, and *max_error* correspond to the

approximation error for a time point, the total approximation error for one piece and the maximum approximation error in one piece, respectively.

```
Initialize $b := 1$, $e := 2$
while $e \leq n$ do
    Initialize $one\_error := 0$, $total\_error := 0$, and $max\_error := 0$
    repeat
        for $i := b + 1$ to $e - 1$ do
            $one\_error := \psi_{b,e}(t_i)$
            $total\_error := total\_error + one\_error$
            $max\_error := max(max\_error, one\_error)$
        end-for
        $e := e + 1$
    until $\frac{total\_error}{t_e - t_b} > \Phi$ or $max\_error > \Psi$ or $e > n$

    Add to the piecewise linear function the piece $z_{b,e-1}$ defined by Formula (1)
    with the current values of $b$ and $e - 1$ with the time interval from $t_b$ to $t_{e-1}$.
    If $b = 1$, set the left boundary of the time interval $-\infty$; if $e - 1 = n$, set the
    right boundary of the time interval $+\infty$.

    $b := e - 1$
end-while
```

Lemma 1 The piecewise linear interpolation algorithm transforms the given data into a piecewise linear function within the specified average approximation error threshold Φ for each piece.

Proof: When $\frac{total\_error}{t_e - t_b} > \Phi$, i.e., the average approximation error is greater than the average error threshold, the *repeat-until* loop exits, and one piece $z_{b,e-1}$ is generated. Therefore, there is no piece whose average approximation error will be greater than Φ. $\square$

Lemma 2 The piecewise linear interpolation algorithm transforms the given data into a piecewise linear function within the specified maximum approximation error threshold Ψ for each data point.

Proof: When $max\_error > \Psi$, i.e., the maximum approximation error for all points in one piece is greater than the maximum error threshold, the *repeat-until* loop exits, and one piece $z_{b,e-1}$ is generated. Therefore, there is no point in one piece whose approximation error will be greater than Ψ. $\square$

We can call the transformation algorithm for each location separately to obtain a piecewise linear approximation of the input data. The output of our transformation algorithm is such that the average approximation error for each piece is less than Φ, and the approximation error for each time point is less than Ψ.

Example 2. Given the temporal data set in Table 2 and the average error threshold $\Phi = 2$ and the maximum error threshold $\Psi = 3$, and assuming that $t_1 = 0, t_2 = 1, t_3 = 2, t_4 = 3, t_5 = 4$, the transformation algorithm will be executed as follows for the weather station 1.

First initialize b to 1, and e to 2. Then do *while* loop. After initializing the local variables one_error, $total\_error$ and max_error, enter into *repeat-unitl* loop. This time *for* loop is skipped because $b + 1 > e - 1$. Then e increases by 1, i.e., $e = 3$. Since the *until* condition is not satisfied, it continues to execute *repeat-until* loop again. After executing *for* loop from 2 to 2, the $one\_error = 3.5$, $total\_error = 3.5$, and $max\_error = 3.5$. So, $\frac{total\_error}{t_e - t_b} = \frac{3.5}{2-0} = 1.75 < \Phi$, and $max\_error = 3.5 > \Psi$. So, the *until* condition is satisfied and hence exit the *repeat-unitl* loop. Then generate one piece $z_{1,2}$, i.e., $75 + 2.00t$ with the time interval from $-\infty$ to 1.

Next, $b = 2$, $e = 3$, do the *while* loop again and again, create the piece $z_{2,3}$ and $z_{3,5}$, unitl $e > n$ exit the *while* loop. At last, a piecewise linear function is generated. The output of the transformation algorithm will be the relation *Temperature* composed of piecewise linear functions as shown in Table 3.

SN	**Temp(t)**	**t**
1	$75 + 2.00t$	$-\infty < t \leq 1$
1	$68 + 9.00t$	$1 < t \leq 2$
1	$82 + 2.00t$	$2 < t < +\infty$
2	$70 + 3.75t$	$-\infty < t < +\infty$
3	$80 + 6.00t$	$-\infty < t \leq 1$
3	$88.67 - 2.67t$	$1 < t < +\infty$
4	$85 - 2.25t$	$-\infty < t < +\infty$

Table 3. The Temperature Relation

We obtain a piecewise linear function over that is applicable at any time for each of the four locations. The resulting piecewise linear function for each location approximates the temperatures by one or several linear pieces.

2.2 Transformation Accuracy Analysis

Note that the interpolation will not be uniformly good on the entire time interval. It will be more accurate in general when the time t we are interested in is close to one of the original data points. The interpolations allow us to look a little backward and forward in time with steadily decreasing reliability as we approach $-\infty$ and $+\infty$. However, for many applications which use reasonable

t values the interpolations seem good enough. They can be improved by using more sophisticated interpolation techniques that use high-degree polynomial functions.

We used the temporal data containing $96 \times 6,726$ temporal data points, that is 96 monthly precipitation data between the year 1990 and 1997 from $6,726$ weather stations throughout the continental United States [9]. The precipitation values ranged between 0 and $4,957$ with an average value of 295.91 and a standard deviation of 269.95.

We tested the transformation accuracy of our algorithm with different values of Φ between 10 and 640, and different values of Ψ between 10 and 640. After the piecewise linear interpolation function was found, we checked the differences between the value of the interpolation function and the original values. We ran separately for each weather station the transformation algorithm and made the correlation tests.

Table 4, Table 5, and Table 6 show the average number of generated pieces of the piecewise linear function and the transformation accuracy for different values of Φ (assume that $\Psi = +\infty$), different values of Ψ (assume that $\Phi = +\infty$) and different values of Φ and Ψ ($\Phi = \Psi$), respectively.

Average Error Threshold	**10**	**20**	**40**	**80**	**160**	**320**	**640**
Average number of linear pieces	84.18	75.79	62.56	44.51	23.48	6.00	1.57
Correlation coefficient	0.9999	0.9993	0.9955	0.9747	0.8800	0.6459	0.4798

Table 4. The statistics for different average error thresholds

Maximum Error Threshold	**10**	**20**	**40**	**80**	**160**	**320**	**640**
Average number of linear pieces	89.03	84.29	76.09	63.22	45.72	25.30	7.84
Correlation coefficient	0.9999	0.9999	0.9993	0.9956	0.9748	0.8775	0.6424

Table 5. The statistics for different maximum error thresholds

Avg. & Max. Thresholds	**10**	**20**	**40**	**80**	**160**	**320**	**640**
Average number of linear pieces	89.03	84.29	76.09	63.22	45.72	25.30	7.84
Correlation coefficient	0.9999	0.9999	0.9993	0.9956	0.9748	0.8775	0.6424

Table 6. The statistics for different average and maximum approximation thresholds

The results of the correlation coefficients show that the transformation is highly accurate when the Φ or Ψ are lower than the average value of the data. The number of pieces in the piecewise linear functions decreases as Φ or Ψ increase. We also combine the two approximation thresholds to test the transformation accuracy.

The maximum number of generated linear pieces for n data points is $n-1$. The relationships between the percent of the number of linear pieces over $n-1$ and the correlation coefficient are shown in Figure 1 and Figure 2 when Φ varies from 10 to 640 and $\Psi = +\infty$, and Ψ varies from 10 to 640 and $\Phi = +\infty$, respectively.

From the Table 6, we can see that the maximum error threshold dominates the transformation accuracy when both of the two thresholds have the same value. This result shows that the maximum error threshold is more restrict to control the transformation than the average error threshold.

Also, we can see that the piecewise linear function transformation has very high correlation with few number of linear pieces. We believe that this holds for any reasonable data set. Form the point view of the storage space, this property shows that the linear function transformation provides a certain ability of data compression.

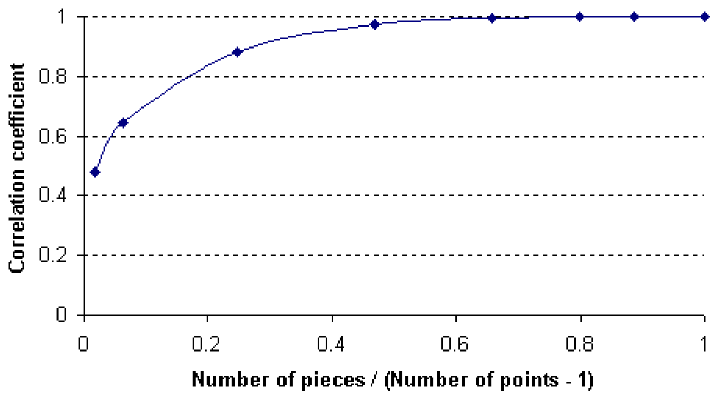

Fig. 1. Φ varies, $\Psi = +\infty$

2.3 Other Interpolation Methods

Given a set of spatiotemporal points, we can apply several numerical analysis algorithms [4, **?**] to construct curves which pass through the given temporal data points. In general, for any n temporal data points, there is always a polynomial

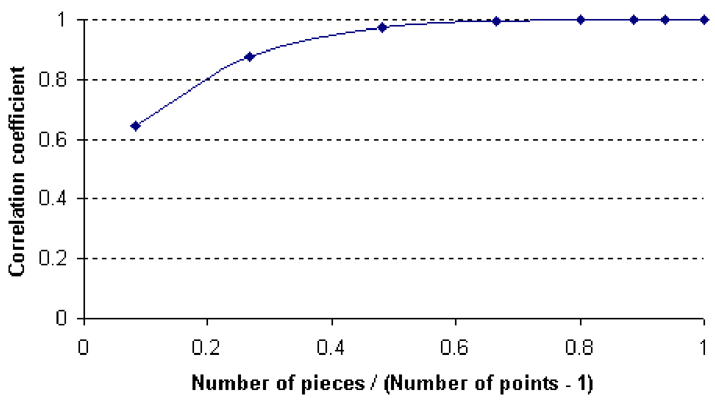

Fig. 2. Ψ varies, $\Phi = +\infty$

function with $n-1$ degree that passes through all of the n points by using interpolation algorithms of Lagrange, Gauss, Bessel, etc [13].

3 The Update on Piecewise Linear Functions

There are two kinds of update operations: insert a new time-value pair into or delete a time-value pair from a piecewise linear function. This section presents *insertion* and *deletion* algorithms of updating the piecewise linear functions.

Insert Operation: From the original data set, we transform the time-value pair for each location into a piecewise linear function. The following algorithm shows how to insert a new time-value pair (t_α, z_α).

INSERTION ALGORITHM:

```
Input:   A piecewise linear function for the data set $(t_1, z_1), \ldots, (t_n, z_n)$.
         $\Psi$ the maximum error threshold in the approximation.
Output:  A new piecewise linear function.

if $t_\alpha < t_1$ then
  Add one piece $z_{\alpha,1}$ into the piecewise linear function
else if $t_\alpha > t_n$ then
  Add one piece $z_{n,\alpha}$ into the piecewise linear function
else
  Using binary search to find the time interval $[t_b, t_e]$ such that $t_b \leq t_\alpha \leq t_e$
  **if** $\psi_{b,e}(t_\alpha)$ is between $\frac{1}{2}\Psi$ and $\frac{3}{2}\Psi$ **then**
    $t_\beta = t_\alpha$
    **if** $z_\alpha < z_{b,e}(t_\alpha)$ **then**
      $z_\beta = z_{b,e}(t_\beta) - \frac{1}{2}\Psi$
    **else**
      $z_\beta = z_{b,e}(t_\beta) + \frac{1}{2}\Psi$
    **end-if**
  **end-if**
  Split the piece $z_{b,e}$ into two pieces, $z_{b,\beta}$ and $z_{\beta,e}$
**end-if**
```

Theorem 1 The insertion algorithm satisfies the condition, such that the approximation errors of the inserted point and all of original points are within the extent Ψ in the new piecewise linear function after inserting the point which is within the extent $\frac{3}{2}\Psi$ from the approximated values in the original piecewise linear function.

Proof: Let us consider the two cases shown in Figure 3.

1. If the approximation error for the point (t_α, z_α) is not greater than half of the maximum approximation error threshold, i.e. $\frac{1}{2}\Psi$, the original piece is not changed. Therefore, in this case the insertion satisfies the condition for the inserted point and all of the original points. This corresponds to the situation of inserting the point u in Figure 3.
2. If the approximation error for the point (t_α, z_α) is greater than $\frac{1}{2}\Psi$ and less than or equal to $\frac{3}{2}\Psi$ in the original piecewise linear function, two new linear pieces are generated after insertion operation. The possible largest approximation error of the inserted point in the new piecewise linear function is equal to the original approximation error minus $\frac{1}{2}\Psi$, hence less than Ψ. This corresponds to the situation of the point v in Figure 3. For all of original points, the maximum approximation error happens at the point v', which is Ψ.

Therefore, this insertion algorithm satisfies the above specified condition. $\square$

Delete Operation: We use the following algorithm to delete a time-value pair (t_α, z_α). We assume that the time points are distributed uniformly between t_1 and t_n. If the point to be deleted is a boundary point for two pieces, say $z_{b-1,b}$ and $z_{b,b+1}$, we approximate the last second point (t_β, z_β) in the piece $z_{b-1,b}$ and the second point (t_γ, z_γ) in the piece $z_{b,b+1}$. Then shrink those two pieces to $z_{b-1,\beta}$ and $z_{\gamma,b+1}$, and insert one new piece $z_{\beta,\gamma}$ into the piecewise linear function. For other cases, the piecewise linear function need not be changed. The deletion diagram is shown in Figure 4.

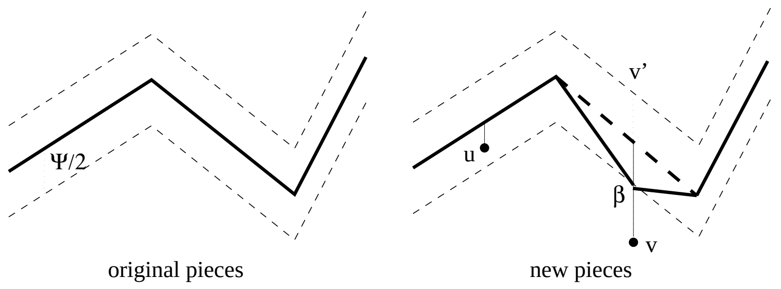

Fig. 3. The insertion operation

DELETION ALGORITHM:

Input: A piecewise linear function for the data set $(t_1, z_1), \ldots, (t_n, z_n)$.
Output: A new piecewise linear function.

```
Using binary search to find the time interval where the point $(t_\alpha, z_\alpha)$ locates
**if** $(t_\alpha, z_\alpha)$ is a boundary point for two pieces, say $z_{b-1,b}$ and $z_{b,b+1}$ **then**
  $t_\beta := t_b - \frac{t_n - t_1}{n}$
  $z_\beta := z_{b-1,b}(t_\beta)$
  $t_\gamma := t_b + \frac{t_n - t_1}{n}$
  $z_\gamma := z_{b,b+1}(t_\gamma)$
  change the piece $z_{b-1,b}$ to $z_{b-1,\beta}$
  change the piece $z_{b,b+1}$ to $z_{\gamma,b+1}$
  insert one new piece $z_{\beta,\gamma}$
**end-if**
```

Remark on Modify Operation: For modify operation, we can do it by executing delete-then-insert operations. First delete the specified data point from the piecewise linear function, then insert the data point with the new value. By doing so, the value of the data point to be modified is changed to its new value.

The Comparisons of the Interpolation Methods: The linear parametric constraint transformation method outperforms other interpolation methods in some important aspects.

First, other interpolation methods needs much more computational time compared with the piecewise linear interpolation transformation since they use higher polynomial functions in their interpolation algorithms.

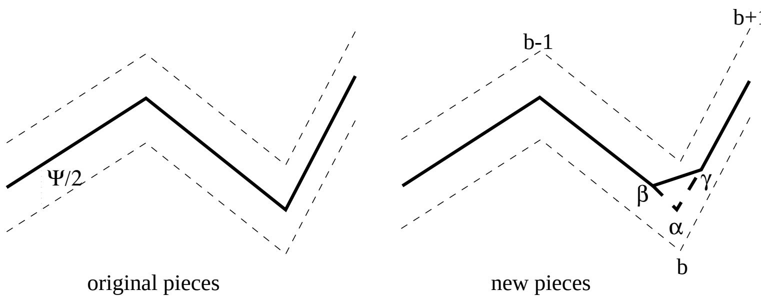

Fig. 4. The deletion operation

Second, the cost of the update operation on other methods is much more than that on the piecewise linear interpolation transformation. Actually, other interpolation methods need reconstruct the whole polynomial function. But using the method we propose before, the update operation only needs $\log(n)$ time to insert one time point.

4 Algebraic Operators and Queries

We implemented the algorithm for transforming a set of temporal data into a piecewise linear constraint database. We also implemented a prototype constraint database system called TAQS, (pronounced: *tax*), which is short for Three-dimensional Animation and Query System. This section describes the capabilities of this system.

We define algebraic operators on the data model introduced in Section 2. Our TAQS system provides users the standard relational algebra operations, such as Project and Join, as well as some standard aggregate operators, such as Min and Max. The following will describe these operations briefly.

Generally, the constraint tuples has the form of $R(a_1, \ldots, a_m, z_1, \ldots, z_n)$, where $a_1, \ldots, a_m$ are m general attributes, and $z_1, \ldots, z_n$ are n geographically distributed attributes represented by piecewise linear functions of t (called *functional attributes* later).

Select: The select operator will return the tuples which satisfy the select condition.

Example 3. For the relation $Temperature(SN, Temp(t))$ in Table 3, the following algebraic query will find the temperature at time $t = 1.5$:

$$\sigma_{t=1.5} Temperature$$

The result of this query will be the temperatures at time $t = 1.5$ shown in Table 7. The result is also a relation (called Temperature_1.5 relation) which can be used by other queries.

SN	Temp(1.5)
1	81.500
2	75.625
3	84.667
4	81.625

Table 7. The select result

Project: This operator is used to reorder the columns of a relation or to eliminate some columns of a relation. It creates a new relation which contains the specified columns of the original relation.

Example 4. For the relation $Temperature\_1.5$ in Example 3, the following query will only return the temperature values.

$$\Pi_{Temp} Temperature\_1.5$$

The result is shown in Table 8.

Temp(1.5)
81.500
75.625
84.667
81.625

Table 8. The project result

Add/Subtract: The *add* or *substract* operation is adapted to functional attributes of relations. The result of add/substract two relations R_1 and R_2 will create a new relation R, where the values of those corresponding functional attributes in R are the piecewise linear functions such that the values at any time instance are the same as the addition/substraction of the values at that time of R_1 and R_2.

Example 5. Suppose there are another relation $Temperature2(SN, Temp(t))$ which has the same number of weather stations as shown in Table 9.

Executing the following query:

$$Temperature + Temperature2$$

will create a new relation $Temperature\_addition$ shown in Table 10.

Intersection: This operation returns the intersection points of two relations to the user. The two relations should have the same attribute names and types.

SN	Temp(t)	t
1	$70 + 3.00t$	$-\infty < t \leq 1$
1	$65 + 8.00t$	$1 < t + \infty$
2	$75 + 3.75t$	$-\infty < t < +\infty$
3	$80 + 5.00t$	$-\infty < t \leq 3$
3	$104 - 3.00t$	$3 < t < +\infty$
4	$80 - 2.25t$	$-\infty < t < +\infty$

Table 9. The Temperature2 relation

SN	Temp(t)	t
1	$145 + 5.00t$	$-\infty < t \leq 1$
1	$133 + 17.00t$	$1 < t \leq 2$
1	$147 + 10.00t$	$2 < t + \infty$
2	$145 + 7.50t$	$-\infty < t < +\infty$
3	$160 + 11.00t$	$-\infty < t \leq 1$
3	$168.67 + 2.33t$	$1 < t \leq 3$
3	$192.67 - 5.67t$	$3 < t < +\infty$
4	$165 - 4.50t$	$-\infty < t < +\infty$

Table 10. The add result

Example 6. Given the relations Temperature and Temperature2, the following query:

$$Temperature \cap Temperature2$$

will return the tuples whose temperatures are the same. The result is shown in Table 11.

SN	Temp(t)	t
1	87.66	$t = 2.83$
3	80.00	$t = 0.00$
3	85.65	$t = 1.13$
3	-35.36	$t = 46.45$

Table 11. The intersection result

Join: This operator executes the natural join operation for two relations A and B which have some attributes in common. It will match these same attributes, then returns the tuples whose projection onto the attributes of A belong to A and whose projection onto the attributes of B belong to B.

Example 7. Suppose there are two relations $Temperature(SN, Temp(t))$ and $Precipitation(SN, Prep(t))$ defined in Table 12. The natural join of these two relations:

$$Temperature \bowtie Precipitation$$

will create a new relation, which includes three attributes SN, $Temp$, and $Prep$. The result is shown in Table 13.

SN	**Prep(t)**	**t**
1	$1050 + 50.00t$	$-\infty < t < +\infty$
2	$980 + 35.00t$	$-\infty < t \leq 5$
2	$1230 - 15.00t$	$5 < t + \infty$
3	$1040 - 20.00t$	$-\infty < t < +\infty$

Table 12. The Precipitation relation

SN	**Temp(t)**	**Prep(t)**	**t**
1	$75 + 2.00t$	$1050 + 50.00t$	$-\infty < t \leq 1$
1	$68 + 9.00t$	$1050 + 50.00t$	$1 < t \leq 2$
1	$82 + 2.00t$	$1050 + 50.00t$	$2 < t < +\infty$
2	$70 + 3.75t$	$980 + 35.00t$	$-\infty < t \leq 5$
2	$70 + 3.75t$	$1230 - 15.00t$	$5 < t < +\infty$
3	$80 + 6.00t$	$1040 - 20.00t$	$-\infty < t \leq 1$
3	$88.67 - 2.67t$	$1040 - 20.00t$	$1 < t < +\infty$

Table 13. The join result

Min/Max: The Min/Max operator will return the minimum/maximum value within a specified time interval.

Example 8. For the relation Temperature in Table 3, the following query will find the minimum temperature during the time interval $1 \leq t \leq 2$:

$$\min(\sigma_{1 \leq t \leq 2} Temperature)$$

The result of this query will be the minimum temperature during that time interval as shown in Table 14.

5 Animation

For spatiotemporal databases, an animation can reveal more information than could be learned by looking at tables of numbers. For the geographical distributed data, such as the population or precipitation distribution in states or

SN	Temp(t)	t
1	77	$t = 1$
2	73.75	$t = 1$
3	83.33	$t = 2$
4	80.5	$t = 2$

Table 14. The minimum result

counties of a state. The areas of states or counties (x, y values) can be represented by polygons. The z values (population, precipitation, temperature, etc.) can be represented by piecewise linear functions. These data can be animated by 3-D animation or cartogram animations.

3-D Animation: In 3-D animation, each constraint tuple in the constraint database can be expressed by a 3-D object. In 3-D animation, at each time instance t, the "height" of the object represents the z value of the constraint tuple at that time t. Besides of using the "height" to represent the z value, we can also give each height value a different color or gray scale to make the z values more clear.

Figure 5 and Figure 6 are two examples of the 3-D animation snapshots for daily mean temperature in the continental U.S. during winter and summer. Note that the higher the "height" of an object is, the lighter its color. This make it more clear, for example, that the mean temperature in Texas is higher than that in North Dakota during summer.

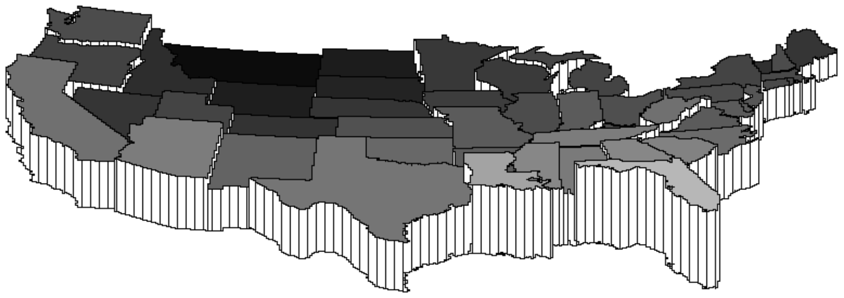

Fig. 5. A Snapshot for 3-D animation of Daily Mean Temperature During Winter

Value-by-Area Cartogram Animation: Besides of 3-D animations, another possible way to display constraint tuples is to use value-by-area cartogram animation [10]. In value-by-area cartogram [5], instead of giving a "height" to each area, each area is enlarged or shrunk proportionally to its z value. Figure 7 is

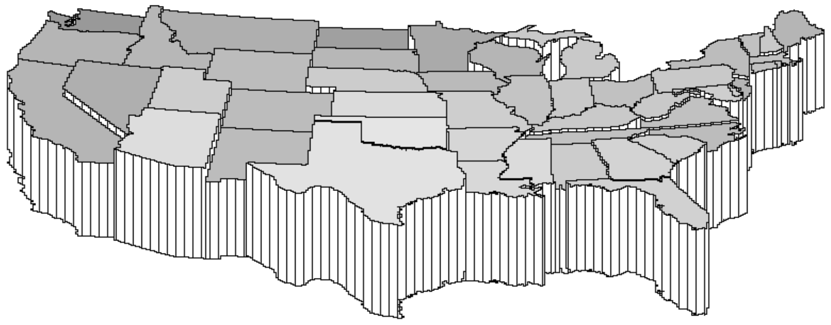

Fig. 6. A Snapshot for 3-D animation of Daily Mean Temperature During Summer

a value-by-area cartogram for the U.S. population in 1990. Value-by-area cartogram animation can be done by displaying the cartogram snapshots consecutively [10].

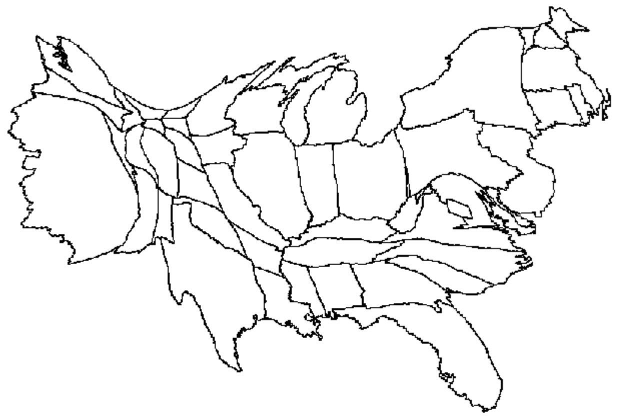

Fig. 7. A Value-by-area Cartogram for the U.S. Population in 1990

Value-by-area cartogram animation is a 2-D animation for 3-D data. Hence, it avoids a problem that could occur in the 3-D animation, namely, the problem when a high-value but nearer object obstructs the view of some low-value and further object. However, the price is the distortion of the objects. For some data distribution, the distortion may be so much that it may be difficult to construct the cartogram and may be also difficult to recognize the objects in the animation.

In contrast, the 3-D animation does not distort the object, and it seems more natural for people to view the "value" as a "height" of an object.

By now we have only discussed the simple case such that the x, y values do not change by time. Hence the x, y values can be represented by points or polygons. It is not always the case. For example, sometimes we may want to animate the population growth for cities in a state, both the city area and the city population change by time. In this case, it may be convenient to represent the city area of x, y by constraint tuples.

If an x, y area is represented by constraint tuples, for example, by conjunction of linear inequalities. It can not be immediately displayed on the computer screen, but has to be converted to some explicit boundary representation (for example, vertices of a polygon). Such a conversion is relatively time-consuming. [2] gives a model, called *Parametric Spaghetti Data Model* to convert and then efficiently animate 2-D animation for moving objects. However, the algorithms in [2] can not be used for general 3-D spatiotemporal objects. For the case that the change of x, y is independent to z, it is possible to apply algorithms as in [2] to compute the boundary of x, y, while getting z value from piecewise functions to have an efficient 3-D animation.

It is also possible to combine the 3-D animations and the value-by-area animations to animate some more complex cases. It is possible to compute the natural join for relations $R_1(x, y, z_1)$ and $R_2(x, y, z_2)$ in which R_1 is the relation for gross state revenue and R_2 is the state per-capita revenue. We may want to view the animation for this join result. In this animation, for each animation snapshot, we may use a value-by-area cartogram to represent the gross revenue and the "height" of each area to represent the per-capita revenue.

Similarity Queries: It is possible to query the similarity for the animations. In practice, people may want to know which map in a set of maps is similar to a given map. This can be done by using similarity queries.

The definition of the similarity depends on the actual situation. There is no universal formula to define the similarity. Our system allows user to change the evaluation rule for similarity measure.

Example 9. Suppose there are the relation R_{Prec} for precipitation. The user wants to find in which year the average precipitation of the U.S. is the closest to that in the year 1997.

To query this information, the user may define the similarity measure rule on cartograms for precipitation. The similarity of two cartograms A and B with n states and each state with area a_i and precipitation values $A.z_i$ and $B.z_i$, respectively, for $1 \leq i \leq n$ may be defined as follows:

$$sim(A, B) = \sum_{i=1}^{n} a_i \mid A.z_i - B.z_i \mid$$

6 Conclusion and Future Works

In this paper, we proposed a linear constraint database system that supports the interpolation and update of the input data, algebraic queries and 3-D animation. Using appropriate queries, the system can support predictions by available data in the databases. The animation provides an expressive visualization that is capable of revealing more information than viewing numbers and tables.

For the future work, one extension for the 3-D animation is to support the rotation, scaling and zooming of the objects. Such abilities will make the animation more expressive.

For queries, an important feature is to support the similarity queries more efficiently. We are planning to explore indexing methods that support efficient similarity queries.

References

1. A. Brodsky, V. Segal, J. Chen and P. Exarkhopoulo. The CCUBE Constraint Object-Oriented Database System. In: *Proc. ACM SIGMOD*, pp. 577-579, 1999.
2. J. Chomicki, Y. Liu and P. Revesz. Animating Spatiotemporal Constraint Databases. In: *Proc. Workshop on Spatiotemporal Database Management*, Edinburgh, Scotland, September 1999.
3. J. Chomicki and P. Revesz. Constraint-Based Interoperability of Spatiotemporal Databases. In: *Proc. 5th International Symposium on Spatial Databases*, Berlin, Germany, pp. 142-161, July 1997.
4. P. J. Davis. *Interpolation and Approximation*, Dover Publications, 1975.
5. B. Dent. *Cartography Thematic Map Design*, McGraw-Hill, 1999.
6. S. Grumbach, P. Rigaux and L. Segoufin. The DEDALE System for Complex Spatial Queries. In: *Proc. ACM SIGMOD Conference*, Estes Park, Colorado, pp. 49-58, August 1997.
7. P. C. Kanellakis, G. M. Kuper and P. Revesz. Constraint Query Languages. *Journal of Computer and System Sciences*, vol. 51, no. 1, pp. 26-52, August 1995.
8. P. Kanjamala, P. Revesz and Y. Wang. MLPQ/GIS: A GIS using linear constraint databases. In: *Proc. the 9th COMAD International Conference on Management of Data*, Tata McGraw Hill, pp. 389-393, 1998.
9. National Climatic Data Center, (NCDC). Monthly Precipitation Data for U.S. Cooperative & NWS Sites. In: *http://www.ncdc.noaa.gov/pub/data/coop-precip/*.
10. M. Ouyang, and P. Revesz. Algorithms for Cartogram Animations. In: *Proc. 4th International Database Engineering and Applications Symposium 2000*, to appear.
11. J. Paredaens. Spatial Databases, The Final Frontier. In: *Proc. International Conference on Database*, Springer-Verlag, pp. 14-32, 1995.
12. F. Preparate and M. Shamos. *Computational Geometry*, Springer-Verlag, 1985.
13. W. Press. *Numerical recipes in C : the art of scientific computing*, Cambridge; New York : Cambridge University Press, 1988.

14. P. Revesz, R. Chen, and et al. The MLPQ/GIS Constraint Database System. In: *Proc. ACM SIGMOD*, 2000.
15. P. Revesz and Y. Li. MLPQ: A Linear Constraint Database System with Aggregate Operations. In: *Proc. 1st International Database Engineering and Applications Symposium*, 1997.
16. L. Vandeurzen, M. Gyssens and D. Van Gucht. On the Desirability and Limitations of Linear Spatial Database Models. In: *Proc. International Symposium on Large Spatial Databases*, Springer-Verlag, pp. 14-28, 1995.