On Effective Data Clustering in Bitemporal Databases

Jong Soo Kim and Myoung Ho Kim

Department of Computer Science
Korea Advanced Institute of Science and Technology
373-1, KuSung-Dong, YuSung-Gu, Taejon, Korea, 305-701
E-mail : {jskim,mhkim}@dbserver.kaist.ac.kr

Abstract

Temporal databases provide built-in supports for efficient recording and querying of time-evolving data. In this paper, data clustering issues in temporal database environment are addressed. Data clustering is one of the most effective techniques that can improve performance of a database system. However, data clustering methods for conventional databases do not perform well in temporal databases because there exist crucial differences between their query patterns. We propose a data clustering measure, called Temporal Affinity, that can be used for the clustering of temporal data. The temporal affinity, which is based on the analysis of query patterns in temporal databases, reflects the closeness of temporal data objects in viewpoints of temporal query processing. We perform experiments to evaluate the proposed measure. The experimental results show that a data clustering method with the temporal affinity works better than other methods.

1 Introduction

Conventional databases represent only a snapshot of the real world, that is, the most recent state. Temporal databases, on the contrary, maintain past, present, and future data. Temporal databases provide users with built-in facilities for recording and querying time-evolving data efficiently. They can be applied to many applications where underlying data have temporal properties inherently, such as trend analysis in decision support systems, version control in computer-aided design, and medical record management.

Temporal databases manage a gigantic amount of data to maintain the history of stored data. Since query processing in temporal databases requires handling of huge data, efficient disk access is needed for high performance of temporal databases. Data clustering is an effective method that can improve the locality of data accesses by clustering related data together in one storage block. In general, data clustering methods improve performance of database systems by enabling them to prefetch and/or postwrite transfered blocks of data [1]. Therefore, it is essential for efficient temporal query processing to adopt a data clustering method designed for temporal database environment.

Data clustering methods for conventional databases do not perform well in temporal databases because there exist many differences between their query patterns. Queries in conventional databases are represented in forms of predicates composed of conditions on attributes of data objects. On the other hand, temporal queries include time conditions, that specify the temporal constraints of data objects on time attributes, as well as the conditions of conventional queries. Time conditions, which are the most effective filtering conditions in temporal queries, are expressed with special *temporal operators* on time attributes of temporal data. Clustering methods for temporal data objects should take these differences into consideration.

There has been much work on temporal databases. However, as far as we know, there is no full-dress proposal for temporal data clustering. [2] addressed the physical organization of temporal data and proposed a data partitioning scheme for range queries and aggregation. This work, however, only considered a single time dimension, which is not enough to be applied to general temporal databases. [3] proposed a partitioned storage structure for temporal databases and mentioned a data clustering issue under the proposed storage model briefly.

In this paper, we propose a data clustering measure, called *Temporal Affinity*, that can be used for the clustering of temporal data. The temporal affinity, which is based on the analysis of query patterns in temporal databases, reflects the closeness of two temporal data objects in viewpoints of temporal query processing. The closeness of temporal data objects is an effective data clustering criteria in temporal databases. It means the possibility that two data objects are selected together by a given temporal query. In our scheme,

0-8186-7937-9/97 $10.00 © 1997 IEEE

two data objects with a high affinity are clustered together in one storage block to reduce the number of disk accesses. Experimental results are also presented in order to evaluate the proposed temporal data clustering measure. Our performance study indicates that a data clustering method with the temporal affinity works better than other methods.

The paper is organized as follows. we describe some preliminary knowledge needed to understand our study, in Section 2. In Section 3, we address *canonical temporal queries*, which is a classification of temporal query patterns. We propose the temporal affinity for each canonical temporal query in Section 4. Section 5 addresses the application of the temporal affinity to temporal data clustering. In Section 6, we discuss performance of a clustering method with the temporal affinity. Finally, we conclude with a summary of our contributions in Section 7.

2 Background

2.1 Temporal databases

In general, temporal databases support two dimensions of time to model the constantly changing real world: *valid time*(a time interval during which a fact is true in the real world) and *transaction time*(a time interval during which a fact is recorded in the database). Temporal databases are classified into *rollback databases*, *historical databases*, and *bitemporal databases* (2TDBs) according to the time dimensions supported. 2TDBs support both valid time and transaction time [4]. In this paper, temporal databases denote 2TDBs and temporal queries denote *bitemporal queries* in 2TDBs.

There have been various proposals to represent temporal data based on the relational data model. In this study, we assume the tuple timestamped representation scheme which is addressed in [5]. Table 1 shows an example of representing temporal data in the tuple timestamped manner. An attribute in a temporal relation is one of *surrogate*, *temporal attributes*, *nontemporal attributes*, and *time attributes*. A *surrogate* is an attribute of a temporal relation which can identify an entity in the real world. A *temporal attribute* is an attribute whose value varies over time. A *nontemporal attribute* is an attribute that have a fixed value over time. *Time attributes* represent the start time and end time of valid time interval and transaction time interval. For example, in Table 1, the surrogate of the *Emp_Sal* relation is *Employee* and *Sex* is the nontemporal attribute. *Salary* is the temporal attribute, and Vs, Ve, Ts, and Te are the time attributes of the relation. In this work, we assume that the temporal domain is a sequence of discrete time instants.

Table 1: A temporal relation *Emp_Sal*

Employee	Sex	Salary	Vs	Ve	Ts	Te
Mary	F	50	0	12	0	3
John	M	40	0	6	0	3
Peter	M	60	0	5	0	2
Mike	M	50	6	12	0	2
Peter	M	60	3	6	3	now
Mike	M	60	6	12	3	now
Mary	F	50	0	3	4	5
Mary	F	70	4	8	6	now
John	M	50	7	12	4	now

2.2 Canonical temporal operators

In temporal databases, special operators are used to specify temporal constraints of data objects. Database users can describe various temporal relationships among data objects with these operators. For example, TSQL2 [6], which is one of the most important query language in temporal databases, uses *when* clauses. Within the *when* clauses, a number of temporal operators can be used, including *overlap, precede, follow, etc.*

In this work, we mainly focus on 6 temporal operators – *overlap, contain, precede, follow, as-of* and *within*. We call these operators as *canonical temporal operators*. The selection of these operators are based on the operators addressed in [3, 6, 7]. Table 2 shows the canonical temporal operators and their interpretations in forms of predicates. In the table, d is a temporal data object and $d.Vs$, $d.Ve$, $d.Ts$, and $d.Te$ represent the time attributes of d. An operand t is a time point and (a, b) is a time interval.

Table 2: Canonical temporal operators

	Operator	Predicate
Valid Time	*d overlap (a,b)*	$d.Vs \leq b \wedge d.Ve \geq a$
	d precede t	$d.Ve < t$
	d follow t	$d.Vs > t$
	d contain (a,b)	$d.Vs \leq a \wedge d.Ve \geq b$
Transaction Time	*d as-of t*	$d.Ts \leq t \leq d.Te$
	d within (a,b)	$(a \leq d.Ts \leq b) \vee$ $(a \leq d.Te \leq b) \vee$ $(d.Ts \leq a \wedge d.Te \geq b)$

The meanings of the above four valid time operators are clear. The *as-of* transaction time operator finds a valid version of a real world entity at a given time point. An operand of the *as-of* operator means *a reference time* – a time point on which query processing is based. In the same manner, the *within* transaction time operator finds all valid versions during a given time interval. The *within* operator is used for the version scanning of a real world entity.

3 Analysis of temporal queries

Since there are a variety of temporal query patterns depending on the temporal operators used in the query, we choose typical query patterns based on [3, 7, 8] and call them as *canonical temporal queries*. The canonical temporal queries contain the following types of temporal queries:

Reference time query This type of query specifies a past/future snapshot of a database and is evaluated based on the specified database state. The canonical transaction time operator *as-of* is used in the query. It is also known as *rollback query*.

Version scanning query It finds data objects which correspond to several versions of a real world entity. The canonical transaction time operator *within* is used.

Historical query It inquires the real world history of data objects in temporal database. One of the canonical valid time operators, *precede, follow, overlap,* and *contain,* is used.

To understand the canonical temporal queries more clearly, let's consider an example in Table 1. Table 1 is a temporal relation that represents the time-varying salaries of employees. Consider the query, "What is Peter's salary during time range [2, 5] as of time 2." It is a reference time query because it finds Peter's salary based on the database state at time 2. It is also a historical query since it asks the real world history of Peter's salary. The query, "Get all the records of Mary's salary that have existed during time range [2, 4]," is an example of version scanning query.

4 Temporal affinity of temporal data objects

In this section, we address the temporal affinity of temporal data objects. We will first define the closeness of two temporal data objects in viewpoints of temporal query processing. Then we derive a temporal affinity, which reflects the closeness of temporal data objects, for each canonical temporal query.

4.1 Closeness of temporal data objects

As we mentioned earlier, the closeness of temporal data objects means the possibility that two temporal data objects are selected together by a given temporal query. The closeness of two temporal data objects is defined as follows:

Definition 1 *Let* **d1, d2** *be two temporal data objects and* **q** *be a temporal query in which canonical temporal operators are used. Then the closeness of* **d1** *and* **d2** *for* **q**, **C(d1,d2,q)**, *is defined as the probability that* **q** *selects* **d1** *and* **d2** *together.*

From the above definition, it is clear that to cluster *d1* and *d2* having a large *C(d1,d2,q)* value is advantageous for efficient processing of *q*. Though there could be many different types of *q*, we focus on canonical temporal queries which have been shown to be the most useful and effective for general temporal database applications.

4.2 Temporal affinities with respect to the canonical temporal queries

Reference time query

To be selected by a reference time query, a temporal data object must have a transaction time interval which contains the time point given in the query. Figure 1 shows time relationships of three temporal data objects *d1*, *d2*, and *d3*.

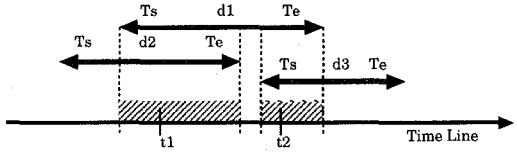

Figure 1: Time relationships of temporal data objects for reference time queries

As illustrated in Figure 1, in order that *d1* and *d2* are selected together by a reference time query, the parameter value *t1* of a reference time query must be in [*d1.Ts, d2.Te*]. In case of *d1* and *d3*, the parameter value *t2* must be in [*d3.Ts, d1.Te*]. From this observation, we can notice that, in reference time queries, the closeness of two temporal data objects has a close relationship with the length of overlap of their transaction time intervals. Therefore, temporal affinity with respect to the reference time queries is defined as follows. It computes the length of the overlapped transaction time interval for two temporal data *d1* and *d2*.

$$AF_r(d1,d2) = \begin{cases} (d1.Te - d1.Ts) + (d2.Te - d2.Ts) \\ \quad -(\max(d1.Te, d2.Te) - \min(d1.Ts, d2.Ts)) \\ \qquad \text{if } d1.Te \geq d2.Ts \wedge d2.Te \geq d1.Ts \\ 0 \qquad \text{otherwise} \end{cases} \quad (1)$$

The following theorem shows the feasibility of AF_r.

Theorem 1 *$AF_r(d1,d2)$ is proportional to C(d1,d2, q_r) for a reference time query q_r.*

Proof: Let *d1*, *d2*, and *d3* be temporal data objects. Assume that $AF_r(d1,d2) = \alpha$, $AF_r(d1,d3) = \beta$, and

$\alpha > \beta$. To prove Theorem 1, it is sufficient to show that the probability of selecting *d1* and *d2* together (p_1) is greater than the probability of selecting *d1* and *d3* (p_2) for any reference time query. Let t be a parameter value of a reference time query which is evaluated at time T_{now}. Also let *lt(d1,d2)* be the length of overlapped transaction time interval of *d1* and *d2*. Then,

$$p_1 = \frac{\text{range of } t \text{ to select } d1 \text{ and } d2}{\text{all possible range of } t} = \frac{lt(d1,d2)}{\text{total time range}} = \frac{\alpha}{T_{now}}$$

In the same manner, $p_2 = \frac{\beta}{T_{now}}$. Since we assumed that $\alpha > \beta$, it is clear that $p_1 > p_2$. □

Version scanning query

Version scanning queries find temporal data objects which correspond to a number of versions of a real world entity. The parameter $[t_s,t_e]$ of a version scanning query specifies the time range to which transaction times of the selected data objects belong. A temporal data object is selected by a version scanning query only if its transaction time interval and the parameter interval of the query overlap. Therefore, two temporal data objects are selected together by a version scanning query only if their transaction time intervals overlap the given parameter interval of the query respectively. Figure 2 illustrates time relationships of three temporal data objects.

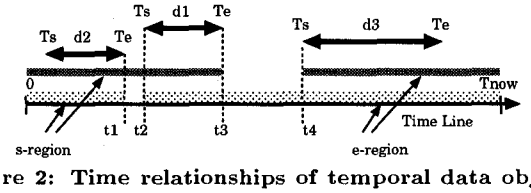

Figure 2: Time relationships of temporal data objects for version scanning queries

The *s-region* means the time range to which the query parameter value t_s should belongs in order to select two temporal data objects together. The *e-region* is defined in the same way for the parameter value t_e. In Figure 2, for *d1* and *d2*, the s-region is $[0,t_1]$ and the e-region is $[t_2,T_{now}]$. For *d1* and *d3*, the s-region is $[0,t_3]$ and e-region is $[t_4,T_{now}]$. Thus the closeness of two temporal data objects for version scanning queries can be determined by multiplying the length of the s-region by the length of the e-region.

Temporal affinity of temporal data objects for version scanning queries is defined as follows. It computes the value obtained by multiplying the length of the s-region by the length of the e-region for two temporal data objects *d1* and *d2*.

$$AF_v(d1,d2) = \begin{cases} \min(d1.Te, d2.Te) \cdot (T_{now} \\ \quad - \max(d1.Ts, d2.Ts)) \\ \qquad \text{if } d1.K = d2.K \\ 0 \qquad \text{otherwise} \end{cases} \quad (2)$$

In AF_v, T_{now} is the time when the affinity is evaluated and $d1.K$ means the surrogate value of $d1$.

The following theorem shows the feasibility of AF_v.

Theorem 2 *$AF_v(d1,d2)$ is proportional to $C(d1,d2,q_v)$ for a version scanning query q_v at time T_{now}.*

Proof : Let *d1*, *d2*, and *d3* be temporal data objects. Assume that $AF_v(d1,d2) = \alpha$, $AF_v(d1,d3) = \beta$, $\alpha > \beta$, $d1.Ts > d2.Te$, and $d1.Ts > d3.Te$. To prove Theorem 2, it is sufficient to show that the probability of selecting *d1* and *d2* together (p_1) is greater than the probability of selecting *d1* and *d3* (p_2) for any version scanning query. Since the current time is T_{now}, the maximum value of possible transaction time is T_{now}. Let $[t_s,t_e]$ be a parameter of a version scanning query. Then t_s and t_e should satisfy the following condition to select both *d1* and *d2* as in Figure 3:

$$0 \le t_s \le d2.Te \wedge d1.Ts \le t_e \le T_{now}$$

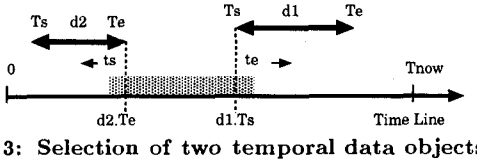

Figure 3: Selection of two temporal data objects for a version scanning query

Therefore, in Figure 4, p_1 is calculated as follows:

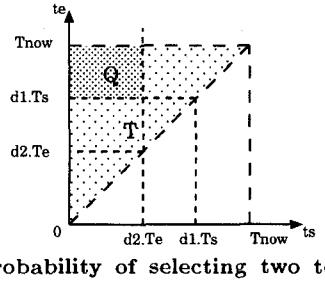

Figure 4: Probability of selecting two temporal data objects together for a version scanning query

$$p_1 = \frac{\text{area in which } d1 \text{ and } d2 \text{ are selected (Q)}}{\text{total area (T)}} = \frac{d2.Te \cdot (T_{now} - d1.Ts)}{\frac{T_{now}^2}{2}} = 2 \cdot \frac{d2.Te \cdot (T_{now} - d1.Ts)}{T_{now}^2}$$

Since $\min(d1.Te, d2.Te) = d2.Te$ and $\max(d1.Ts, d2.Ts) = d1.Ts$, $p_1 = \frac{2 \cdot \alpha}{T_{now}^2}$. In the same manner, $p_2 = \frac{2 \cdot \beta}{T_{now}^2}$. Hence, it is clear that $p_1 > p_2$ from the assumption, $\alpha > \beta$. □

Historical query

Historical queries inquire the real world history of data objects in temporal databases. Since some of the canonical valid time operators, which are used in historical queries, have exclusive properties, it is impossible to derive a single temporal affinity for historical queries. Therefore, we present temporal affinity for each canonical valid time operator. The main idea of each temporal affinity is addressed. The feasibility of the presented affinity could be shown in similar ways as Theorem 1 and Theorem 2 without much effort.

- *Case 1 : overlap*

Historical queries with *overlap* operator select temporal data objects whose valid time interval $[Vs, Ve]$ overlap the parameter interval $[v_s, v_e]$ given in the query. This condition is similar to that of version scanning queries. However, there is a crucial difference in that valid time intervals of two temporal data may overlap. In version scanning queries, it is impossible for two transaction time intervals to overlap since we concern only the objects correspond to versions of a real world entity. Thus the difference should be considered in the temporal affinity for historical queries with *overlap* operator.

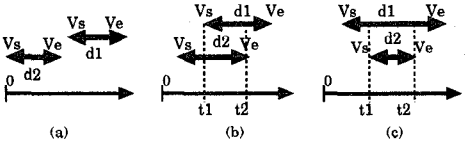

Figure 5: Relative positions of two valid time intervals

Figure 5 shows relative positions of two valid time intervals on the time line. Since (a) is the same case with version scanning queries, we concentrate on (b) and (c). In the case (b) and (c), to select both *d1* and *d2*, a historical query with *overlap* operator should have a parameter interval $[v_s, v_e]$ which satisfy the following condition.

$$\neg((v_s < t1 \wedge v_e < t1) \vee (v_s > t2 \wedge v_e > t2))$$

The condition denotes the negation of the condition that two temporal data objects *d1* and *d2* are not selected together. Therefore, temporal affinity of two temporal data objects for historical queries with *overlap* operator is given as follows. In AF_{h-o}, T_{max} denotes the maximum time that is permitted in the temporal database.

The first part of AF_{h-o} is for the case that valid time intervals of two temporal data objects overlap. It computes the area Q in which two temporal data objects are selected together as in Figure 6. The second part of AF_{h-o} is conceptually identical with AF_v.

$$AF_{h-o}(d1, d2) = \begin{cases} \min(d1.Ve, d2.Ve) \cdot (T_{max} \\ \quad - \max(d1.Vs, d2.Vs)) \\ \quad - \frac{(\min(d1.Ve, d2.Ve) - \max(d1.Vs, d2.Vs))^2}{2} \\ \quad \text{if } d1.Vs \leq d2.Ve \wedge d1.Ve \geq d2.Vs \\ \min(d1.Ve, d2.Ve) \cdot (T_{max} \\ \quad - \max(d1.Vs, d2.Vs)) \\ \quad \text{otherwise} \end{cases} \quad (3)$$

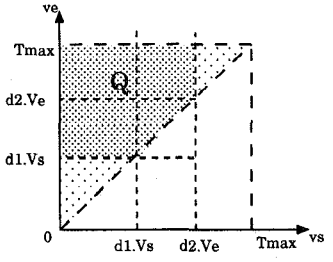

Figure 6: The possible range of $[v_s, v_e]$ in which both *d1* and *d2* are selected

- *Case 2 : contain*

In this case, temporal data objects, whose valid time interval $[Vs, Ve]$ contains the parameter interval of a historical query $[v_s, v_e]$, are selected. This condition is similar with that of reference time queries except that the subject of inclusion is a time interval. Therefore, temporal affinity is derived easily based on the affinity for reference time queries, AF_r, as follows.

$$AF_{h-c}(d1, d2) = \begin{cases} (d1.Ve - d1.Vs) + (d2.Ve - d2.Vs) \\ \quad -(\max(d1.Ve, d2.Ve) \\ \quad \quad - \min(d1.Vs, d2.Vs)) \\ \quad \text{if } d1.Ve \geq d2.Vs \wedge d2.Ve \geq d1.Vs \\ 0 \quad \text{otherwise} \end{cases} \quad (4)$$

It computes the length of the overlapped valid time interval for two temporal data objects *d1* and *d2*.

- *Case 3 : precede*

This type of historical query selects temporal data objects that have valid time intervals taking precedence over a specific time point. Hence, two temporal data objects are selected together by a historical query with *precede* operator only if both of their valid time intervals precede a parameter value t of the query. Figure

7 shows the relationships of temporal data objects for historical queries with *precede* operator.

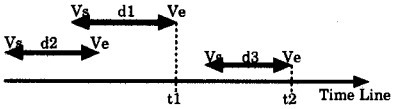

Figure 7: The relationships of temporal data objects for historical queries with *precede* operator

Let T_{max} denotes the maximum time that can be represented in the temporal database. In Figure 7, a parameter value t should be in time range $[t_1, T_{max}]$ for a query to select *d1* and *d2* together. In case of *d1* and *d3*, t should be a value in $[t_2, T_{max}]$. From this observation, temporal affinity for historical queries with *precede* operator is derived as follows.

$$AF_{h-p}(d1, d2) = T_{max} - max(d1.Ve, d2.Ve) \quad (5)$$

It computes the possible range of a parameter for a historical query with *precede* operator that selects *d1* and *d2* together.

- *Case 4 : follow*

In this case, temporal data objects, whose valid time interval $[Vs, Ve]$ follows the time point specified in a query, are selected. Since this is a symmetrical case of *precede* operator, temporal affinity is derived easily as follows.

$$AF_{h-f}(d1, d2) = min(d1.Vs, d2.Vs) \quad (6)$$

5 The application of temporal affinities to temporal data clustering

In Section 4, we have presented the temporal affinity of two temporal data objects with respect to each canonical temporal query. However, an unification of these affinities is needed to apply them to temporal data clustering.

One way of unifying temporal affinities is to construct a single temporal affinity as the weighted sum of each affinity. Thus, the unified temporal affinity is given as follows:

$$\begin{aligned} AF(d1, d2) = {} & \alpha_1 \cdot AF_r(d1, d2) + \alpha_2 \cdot AF_v(d1, d2) \\ & + \alpha_3 \cdot AF_{h-o}(d1, d2) + \alpha_4 \cdot AF_{h-c}(d1, d2) \\ & + \alpha_5 \cdot AF_{h-p}(d1, d2) + \alpha_6 \cdot AF_{h-f}(d1, d2) \end{aligned}$$

In $AF(d1, d2)$, α_i means the weight of each temporal affinity. There are many factors that affect the weight of each temporal affinity. In this work, α_i is determined by the frequency of each canonical temporal query in an application. Thus,

$$\alpha_i = f_i \cdot s_i$$

$$0 \leq f_i \leq 1, \sum_{i=1}^{2} f_i = 1, \sum_{i=3}^{6} f_i = 1$$

where f_i is the frequency of the query pattern on which the corresponding temporal affinity is based and s_i is a scaling factor that is needed to normalize each temporal affinity. Since a valid time condition and a transaction time condition can be specified together in a temporal query, the constraint on f_i is separated into two parts.

Hence, the data objects in temporal databases can be clustered with the proposed clustering measure, unified temporal affinity. The details of the clustering algorithm is not presented in this paper. Since there have been many proposals on the general data clustering algorithm, such as [9, 10], we can introduce one of them to the clustering procedure of temporal data objects with the proposed measure.

6 Performance evaluation

In this section, we present performance evaluation of the proposed data clustering method. Other clustering methods with straightforward clustering measures, such as surrogate, transaction time, and valid time are also evaluated for the comparison with the proposed method.

6.1 An experiment

We conducted an experiment that simulates benchmark query processing on clustered data objects in order to evaluate the performance of clustering methods with various clustering measures. The performance of a data clustering method is represented by the number of disk page accesses during the processing of a set of benchmark queries. To reduce implementation overhead, we observe average number of cluster references during the benchmark query processing for a benchmark query. The number of cluster references represents the performance of a clustering method if the total number of clusters approximates to the total number of disk pages.

The experiment is composed of the following steps.

1. **Data generation** : In this step, a set of temporal data objects are generated by given parameters.
2. **Clustering of data objects** : Temporal data objects, generated in step 1, are clustered using a given clustering measure.
3. **Benchmark query processing** : Benchmark queries are processed on the clustered temporal data objects. During the query processing, average number of cluster references for a benchmark query is observed.

Since we could not obtain any temporal data set in real applications, we generated the temporal data objects whose time attributes are random variables taking particular distributions. For example, we assume that the time interval between transaction start times (Ts) of two consecutive versions takes the *Poisson* distribution because Ts of a temporal data object represents a time point at which the object is recorded in the database. Therefore, Ts takes the exponential distribution.

In step 2, we use CLARA (Clustering LARge Applications) [9] to cluster temporal data objects into predetermined number of clusters. The number of clusters (K) is determined by the total number of data objects (N), the size of a data object (S), and the size of a disk page (P) as $K = \lceil \frac{N}{\lfloor \frac{P}{S} \rfloor} \rceil$. Therefore, K approximates to the number of disk pages.

The benchmark queries, processed in step 3, are selected based on [3]. Table 3 shows the operators that are used in the benchmark queries. We assume that the parameters of benchmark queries are random variables taking the uniform distribution.

Table 3: Operators in benchmark queries

Query type	Valid time operator	Transaction time operator
1	*overlap*	*as-of*
2	*precede*	*as-of*
3	*follow*	*as-of*
4	*contain*	*as-of*
5	*overlap*	*within*
6	*precede*	*within*
7	*follow*	*within*
8	*contain*	*within*

Table 4 shows the important parameters used in the experiment and their values.

Table 4: The parameters

Parameter	Value
Time range	[0, 3,650]
Number of objects (N)	5,000 ... 100,000
Size of an object (S)	30 ... 300
Page size (P)	4K bytes
Number of clusters	$K = \lceil \frac{N}{\lfloor \frac{P}{S} \rfloor} \rceil$
Number of benchmark queries	1,000

6.2 Analysis of the results

In the experiment, clustering methods with the following measures are also evaluated for the comparison with the proposed method. We compared the proposed method with ad-hoc methods since there is no clustering measure proposed for general temporal database environment.

- **Surrogate (KEY)** : Cluster the objects which correspond to versions of a real world entity. In other words, the objects having the same surrogate value are clustered together.
- **Transaction Time (TT)** : Cluster the objects based on the time at which they are stored in the database. The objects having similar Ts values are clustered together.
- **Valid Time (VT)** : The objects having similar Vs values are clustered together.

The results are presented in Table 5. Table 5 shows the average number of cluster references for a benchmark query. The results are obtained by varying the total number of data objects (N) and the size of a data object (S). Since the clustering algorithm CLARA is based on the sampling of underlying data objects, we repeated the same experiment 12 times and present the average of 10 values to the exclusion of the maximum and the minimum value. Moreover, since the clustering method, KEY, shows the worst performance, we present it only in the case that N = 100,000.

Figure 8 presents the average number of cluster references when N = 100,000. In the figure, we observe that the clustering method for conventional databases, KEY, performs badly in temporal databases.

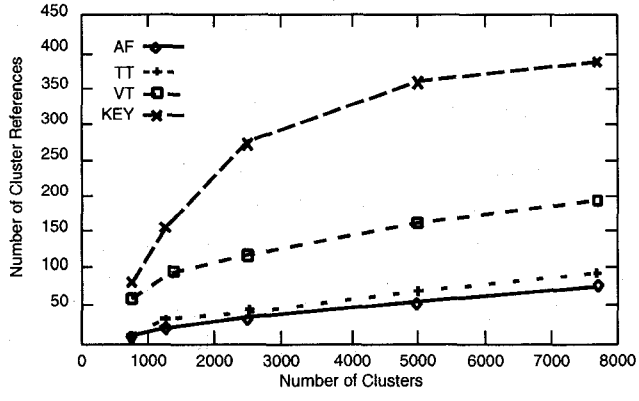

Figure 8: Performance of clustering methods when N = 100,000

Experimental results show that the proposed method, AF, outperforms the other methods in all cases. Therefore, the proposed data clustering method causes less number of disk page accesses than other methods. The results also explain that the performance gap is getting larger as the size of a data object increases. This observation means that the effectiveness of a data clustering measure becomes more critical as the size of a data object grows.

In the experimental results, TT works better than VT. This observation can be explained by the properties of the benchmark queries. That is, it is advantageous to cluster data objects with similar Ts values

Table 5: Results : avg. number of cluster references

Number of objects N	Clustering methods	Size of an object(S)				
		30	50	100	200	300
5000	AF	7.5193	9.3970	12.7682	16.7208	19.8843
	TT	7.6110	10.6383	14.0037	19.6466	23.2365
	VT	10.7831	14.7195	21.4152	29.5773	34.7333
10000	AF	10.1304	13.6445	17.6532	23.6878	28.4242
	TT	11.6308	14.6210	20.4612	28.2998	33.6648
	VT	16.2788	22.7593	33.5147	46.1629	52.3493
50000	AF	24.0835	28.2283	36.0065	48.8135	58.0326
	TT	27.5847	29.3154	41.2513	53.2884	66.3661
	VT	48.2473	57.4623	82.1416	105.3862	120.2427
100000	AF	36.6513	38.1123	47.4983	66.8941	81.3429
	TT	33.7277	37.7987	50.0324	72.7964	92.8960
	VT	63.4208	89.8035	127.1227	168.4137	187.6794
	KEY	78.6571	153.3342	261.3214	354.0892	391.1975

for the canonical transaction time operators, *as-of* and *within*. However, for the canonical valid time operators, *precede* and *follow*, clustering of objects with similar Vs values does not cause performance gain.

7 Conclusion

In this paper, data clustering issues in temporal database environment have been addressed. Since temporal databases maintain a gigantic amount of data, an efficient data clustering method is essential for high performance. In temporal databases, data clustering methods for conventional databases do not perform well because temporal query patterns are different from conventional query patterns. Temporal queries include time conditions that are the most effective filtering conditions in temporal queries. Therefore, a data clustering method for temporal databases should take the time conditions into consideration for efficient processing of temporal queries.

We have proposed a data clustering measure, called temporal affinity, which reflects a temporal data clustering criteria, the closeness of temporal data objects in viewpoints of temporal query processing. We first have analyzed temporal query patterns and classified them into three canonical query patterns, i.e. reference time query, version scanning query, and historical query. Temporal affinity for each canonical query pattern have been defined with a demonstration of the feasibility. Then we also have discussed the unification of the temporal affinities which is needed to apply them to temporal data clustering. We have conducted performance experiments to evaluate the performance of the proposed data clustering method. The experimental results have shown that the proposed method outperforms other data clustering methods.

To the best of our knowledge, we believe that this work is the first attempt to develop an effective data clustering method in general temporal database environment.

References

[1] J. Gray and A. Reuter. *Transaction Processing : Concepts and Techniques.* Morgan-Kaufmann Publishers, 1993.

[2] D. Rotem and A. Segev. Physical Organization of Temporal Data. In *Proceedings of the 3rd ICDE*, pages 547–553, 1987.

[3] I. Ahn and R. Snodgrass. Partitioned Storage for Temporal Databases. *Information Systems*, 13(4):369–391, 1988.

[4] C. Jensen, J. Clifford, S. Gardia, and et al. A Consensus Glossary of Temporal Database Concepts. *ACM SIGMOD Record*, 23(1), 1994.

[5] C. Jensen, M. Soo, and R. Snodgrass. Unifying Temporal Data Model via a Conceptual Model. *Information Systems*, 19(7):513–547, 1994.

[6] R. Snodgrass, I. Ahn, G. Ariav, and et al. TSQL2 language specification. *ACM SIGMOD Record*, 23(1):65–86, 1994.

[7] J. Won and R. Elmasri. Representing Retroactive and Proactive Versions in Bi-Temporal Databases(2TDB). In *Proceedings of the 12th ICDE*, pages 85–94, 1996.

[8] B. Salzberg and V. J. Tsotras. A Comparison of Access Methods for Time Evolving Data. Technical Report NU-CCS-94-21, 1994.

[9] L. Kaufmann and P.J. Rousseeuw. *Finding Groups in Data: an Introdution to Cluster Analysis.* John Wiley and Sons, 1990.

[10] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proceedings of SIGMOD Conference*, pages 103–114, 1996.