# On Computing Temporal Aggregates Over Null Time Intervals

Kai Cheng

Graduate School of Information Science, Kyushu Sangyo University,
2-3-1, Mtsukadai, Higashi-ku, Fukuoka 813-8503, Japan
http://www.is.kyusan-u.ac.jp/~chengk/

**Abstract.** In a temporal database, each data tuple is accompanied by a time interval during which its attribute values are valid. In this paper, we consider the null time intervals, that is, time intervals not intersected by any time intervals in the temporal database. We deal with the problem of computing temporal aggregates over null time intervals with length constraints. By interval folding, we transform the problem into aggregates over stabbing groups, maximal stabbing interval sets. We describe the detailed algorithms and report the experimental results.

**Keywords:** temporal database, temporal aggregation, stabbing query, null time interval, interval folding

## 1 Introduction

We consider the problem of computing temporal aggregates over null time intervals. In a temporal database, data tuples are typically accompanied by time intervals that capture the valid time of the information or facts. Consider a scheduling system where scheduled activities for individuals or groups are stored in a temporal relation. In order to create a new activity for a group of people, one has to find time intervals during which all members can participate. We call such time intervals *null time intervals*. A time interval is said to be a null time interval when no time intervals in the database intersect with it. The qualifying null time intervals should also satisfy length constraint. For example there must be at least 90 minutes free time for the new activity. Furthermore, when no qualifying null time interval is available, a partially null time interval can also be seen as a feasible choice. For example, a query for free time intervals of 10 members may accept results with 1 or 2 members absent.

To report qualifying null time intervals, it is important to compute temporal aggregates. Support for temporal aggregates is a predominant feature of many data management systems. When aggregating temporal relations, tuples are grouped according to their timestamp values. There are basically two types of temporal aggregation: instant temporal aggregation and span temporal aggregation [6, 12]. *Instant temporal aggregation (ITA)* computes aggregates on each time instant and consecutive time instants with identical aggregate values are

coalesced into so-called *constant intervals*, i.e., tuples over maximal time intervals during which the aggregate results are constant [7]. On the other hand,*Span temporal aggregation (STA)* allows an application to control the result size by specifying the time intervals, such as year, month, or week. For each of these intervals a result tuple is produced by aggregating over all argument tuples that overlap that interval.

**(a) Event time**

| | Symbol | Event Time |
|---|---|---|
| $e_1$ | A | $[1, 4]$ |
| $e_2$ | A | $[14, 17]$ |
| $e_3$ | B | $[7, 12]$ |
| $e_4$ | B | $[19, 21]$ |
| $e_5$ | A | $[25, 29]$ |
| $e_6$ | B | $[3, 5]$ |

**(b) Null time**

| | Symbol | Null Time |
|---|---|---|
| $r_1$ | A | $[5, 13]$ |
| $r_2$ | A | $[18, 24]$ |
| $r_3$ | A | $[30, 50]$ |
| $r_4$ | B | $[1, 2]$ |
| $r_5$ | B | $[6, 6]$ |
| $r_6$ | B | $[13, 18]$ |
| $r_7$ | B | $[22, 50]$ |

**(c) Aggregates**

| Null Time | Group | CNT |
|---|---|---|
| $[20, 21]$ | $\{A\}$ | 1 |
| $[\mathbf{22}, \mathbf{24}]$ | $\{A, B\}$ | **2** |
| $[26, 29]$ | $\{B\}$ | 1 |
| $[\mathbf{30}, \mathbf{35}]$ | $\{A, B\}$ | **2** |

Fig. 1: Running example of null time aggregates on $[20, 35]$

In this paper, we study the problem of temporal aggregates over null time intervals. Fig. 1 gives a running example. Assume the time domain is $[1, 50]$. Event time in Fig. 1 (a) is physically stored in a temporal database, in which each event symbol is accompanied by an event time. Null time in Fig. 1 (b) is a derived relation during query time. For example, $A$ has an event sequence $\{[1, 4], [14, 17], [25, 29]\}$, from which the null time $\{[5, 13], [18, 24], [30, 50]\}$ is derived. Temporal aggregates on $[20, 35]$ as shown in Fig. 1 (c) are computed by grouping tuples in the query range at first and then applying aggregate functions to each group. During $[22, 25]$, $[30, 35]$, both $A$ and $B$ are not overlapped by any event time, we call them truly null time, whereas $[20, 21]$ and $[26, 29]$are partially null time.

Support for null time intervals is not provided by current database products. Syntactically, all relational database management systems (RDBMS) support a representation of "missing information and inapplicable information". Null (or NULL) is a special value to indicate that a data value does not exist in the database. However, to our knowledge, there not exist database systems that support null time since it not a practical solution to explicitly store null time intervals in databases. Neither NOT IN or NOT EXISTS is suitable for querying time intervals that not intersected by other intervals because null time intervals depend on time domain.

Our contributions include: (1) we introduce a new operation called interval folding and transform the problem to interval stabbing problem; (2) we propose stabbing group as a new temporal grouping method to solve the interval stabbing problem. (3) We develop a balanced tree based data structure and algorithms for efficient computation of temporal aggregates over null time intervals.

The rest of paper is organized as follows. In Section 2, we define the problem and proposes the main techniques. Section 3 describes the main techniques. Section 4 introduces the experimental results. Section 6 concludes the paper and points out some future directions.

## 2 Problem Definition

Let $E = \{e_1, e_2, \cdots, e_k\}$ be the set of event symbols and $N$ be the time domain. The triplet $(e_i, s_i, f_i) \in E \times N \times N$ is an *event interval* or *real time interval*. The two time points $s_i, f_i$ are called event times, where $s_i$ is the starting time and $f_i$ is the finishing time, $s_i \leq f_i$. The set of all event intervals over $E$ is denoted by $I$. An *event sequence* is a series of event interval triplets $E_S = \langle (e_1, s_1, f_1), (e_2, s_2, f_2), \cdots, (e_n, s_n, f_n) \rangle$, where $s_i \leq s_{i+1}$, and $s_i < f_i$. A temporal database $D$ is a set of records, $\{r_1, r_2, \cdots, r_m\}$, each record $r_i$ $(1 \leq i \leq m)$ consists of a sequence-id and an event interval.

For $S \subseteq D$, a *null time interval* $\boldsymbol{a}$ is an interval that for any $\boldsymbol{b} \in S$, $\boldsymbol{a} \cap \boldsymbol{b} = \emptyset$. As shown in Fig. 2, during real time intervals events are valid, while events are invalid during null time intervals. We assume in this paper that only real time intervals are explicitly stored in database. Given the temporal database $D$ and a query interval $[p, q]$ null time intervals can be derived as follows.

$$[p, q] - \bigcup_{[s,f] \in D} [s, f]$$

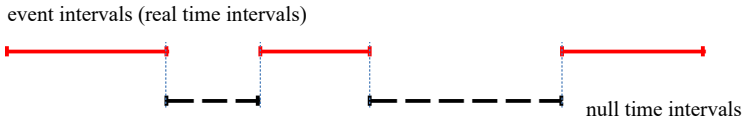



Fig. 2: Real time interval vs. null time interval

Given an event sequence $q = \langle (e_1, s_1, f_1), (e_2, s_2, f_2), \cdots, (e_n, s_n, f_n) \rangle$, the set $T = \{s_1, f_1, s_2, f_2, \cdots, s_i, f_i, \cdots, s_n, f_n\}$ is called a time set corresponding to sequence $q$ where $1 \leq i \leq n$. If we order all the elements in $T$ and eliminate redundant elements, we can derive a sequence $T_S = \langle t_1, t_2, t_3, \cdots, t_k \rangle$ where $t_i \in T$, $t_i < t_{i+1}$. $T_S$ is called a time sequence corresponding to event sequence $q$.

When discussing time intervals, it is important to describe pairwise relationships between two time interval-based events. According to Allen's temporal logics [1], the basic temporal relations between any two event intervals are shown in Fig. 3. Except (g), each of (a)–(f) has its inverse relation. For example, "A before B" also means "B after A", "A contains B" implies "B is contained by A", etc. These relationships can describe any relative position of two intervals based on the arrangements of the starting and finishing time points.

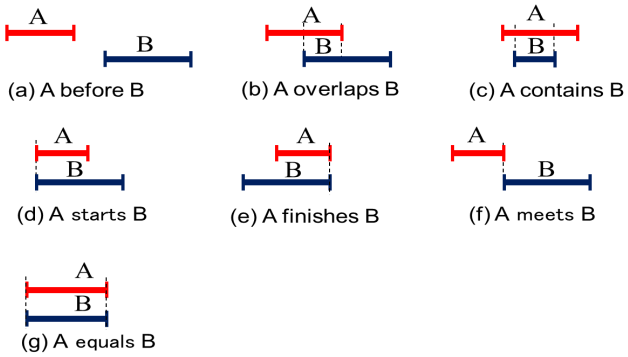



Fig. 3: Temporal relations between two intervals

Now we can formulate the problem we target as follows:

*Null Time Reporting Problem*: Given a temporal database $D$, a query interval $[p, q]$ and a parameter $\alpha$, report all null time intervals $\boldsymbol{b} = [b_s, b_f] \subseteq [p, q]$ and $|\boldsymbol{b}| \geq \alpha$.

### 2.1 Interval Folding

We introduce *interval folding*, an operation that transforms an interval to a shorter one. For interval $\boldsymbol{b} = [b_s, b_f]$, the $\alpha$-folding of $\boldsymbol{b}$ is defined as:

$$\boldsymbol{b} - \alpha = [b_s + (1 - \lambda)\alpha, b_f - \lambda\alpha]$$

where $\lambda$ is a parameter, which can be any real value between 0 and 1, for example

1. $\lambda = 0$, $[s, f] \rightarrow [s + \alpha, f]$
2. $\lambda = 1$, $[s, f] \rightarrow [s, f - \alpha]$
3. $\lambda = \frac{1}{2}$, $[s, f] \rightarrow [s + \frac{\alpha}{2}, f - \frac{\alpha}{2}]$

The $\alpha$-folding of an interval set $S$, denoted by $S - \alpha$, is defined by applying $\alpha$-folding to each element interval.

**Lemma 1.** *Let $S$ be a set of intervals. If $\bigcap_{\mathbf{b} \in S}(\mathbf{b} - \alpha) \neq \emptyset$, then $\bigcap_{\mathbf{b} \in S}(\mathbf{b} - \alpha) = \bigcap_{\mathbf{b} \in S} \mathbf{b} - \alpha$*

*Proof.* Let $\hat{s} = \max\{b_s \mid [b_s, b_f] \in S\}$, $\hat{f} = \min\{b_f \mid [b_s, b_f] \in S\}$. Then

$$\bigcap_{\boldsymbol{b} \in S} \boldsymbol{b} = [\hat{s}, \hat{f}]$$

It is obvious that $\max\{b_s + (1 - \lambda)\alpha \mid [b_s, b_f] \in S\} = \hat{s} + (1 - \lambda)\alpha$, $\min\{b_f - \lambda\alpha \mid [b_s, b_f] \in S\} = \hat{f} - \lambda\alpha$, that is

$$\bigcap_{\boldsymbol{b} \in S} (\boldsymbol{b} - \alpha) = [\hat{s} + (1 - \lambda)\alpha, \hat{f} - \lambda\alpha] = \bigcap_{\boldsymbol{b} \in S} \boldsymbol{b} - \alpha.$$

**Theorem 1.** *The intersection of $S$ has a length larger than $\alpha$, if and only if the intersection of $S - \alpha$ is non–empty:*

$$\bigcap_{\mathbf{b}\in S} \mathbf{b} - \alpha \neq \emptyset \Leftrightarrow \bigcap_{\mathbf{b}\in S} (\mathbf{b} - \alpha) \neq \emptyset$$

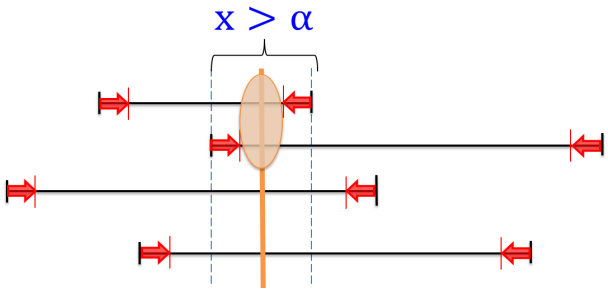



Fig. 4: Interval folding

*Proof.* Let $\boldsymbol{x}$ be the intersection of $S$ and $\boldsymbol{x} - \alpha \neq \emptyset$. As for any $\boldsymbol{b} \in S$, $\boldsymbol{b} - \alpha \supseteq \boldsymbol{x} - \alpha$, therefore

$$\bigcap_{\boldsymbol{b}\in S} (\boldsymbol{b} - \alpha) \supseteq \boldsymbol{x} - \alpha \neq \emptyset$$

On the other hand, if $\bigcap_{\boldsymbol{b}\in S}(\boldsymbol{b} - \alpha) \neq \emptyset$, then by Lemma 1, $\bigcap_{\boldsymbol{b}\in S} \boldsymbol{b} - \alpha = \bigcap_{\boldsymbol{b}\in S}(\boldsymbol{b} - \alpha) \neq \emptyset$.

Theorem 1 tells that by $\alpha$-folding, a null time reporting problem can be transformed into the interval stabbing problem: given a query interval $[p, q]$, report all non–empty null time intervals in $[p, q]$.

## 3 Temporal Aggregates Over Null Time Intervals

We now present the techniques for computing temporal aggregates over null time intervals. Given a temporal database $D$, a query interval $[p, q]$ and the length threshold $\alpha$, the basic idea to compute the temporal aggregates is to derive null time intervals from the event times, and then by $\alpha$-folding, transform the problem to interval stabbing problem.Thus, one just need to report all non–empty null time intervals contained in the query interval.

### 3.1 Instant temporal aggregation

A solution to interval stabbing problem is instant temporal aggregation. The key idea is to partition the timeline into elementary intervals. The elementary intervals are obtained by sorting the endpoints of argument intervals and consecutive

two endpoints define an elementary intervals. For each elementary interval, an aggregate value is computed.

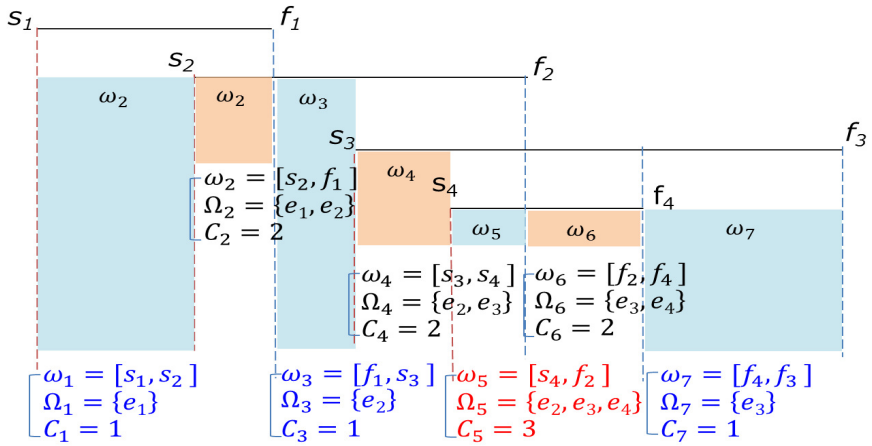



Fig. 5: Instant temporal aggregation

Fig. 5 shows the temporal aggregates by instant temporal aggregation over $\{< e_1, s_1, f_1 >, < e_2, s_2, f_2 >, \cdots, < e_4, s_4, f_4 >\}$. The timeline is partitioned into a sequence of elementary intervals $\omega_1, \cdots, \omega_7$ from left to right: $\omega_1 = [s_1, s_2], \omega_2 = [s_s, f_1], \omega_3 = [f_1, s_3], \omega_4 = [s_3, s_4], \omega_5 = [s_4, f_2], \omega_6 = [f_2, f_4], \omega_7 = [f_4, f_3]$. With each elementary interval, we maintain a list of event symbols $\Omega_i$ and a count $C_i$.

$$\omega_i = \bigcap_{e_k \in \Omega_i} [s_k, f_k], C_i = |\Omega_i|$$

### 3.2 Stabbing groups

The brute–force approach to computing instant temporal aggregates requires multiple passes to scan the argument relation. We propose a balanced tree based approach for efficient computation. The basic idea is to maintain the aggregate groups using a balanced search tree.

As shown in Fig. 6, each node of the balanced tree keeps a stabbing group. A stabbing group is a set of intervals stabbed by a common set of points. An interval $\boldsymbol{b}$ is *stabbed* by a point $q$ is $q \in \boldsymbol{b}$. The common set of points is called *group representative*, which is actually the intersection of the argument intervals. In Fig. 6, the the representative of stabbing group $I_{mid}$ is an interval $\boldsymbol{x}_{mid}$. The two children of $I_{mid}$, stabbing groups $I_{left}$ and $I_{right}$ have their group representatives $\boldsymbol{x}_{left}$ and $\boldsymbol{x}_{right}$, and $\boldsymbol{x}_{left} < \boldsymbol{x}_{mid} < \boldsymbol{x}_{right}$. The $<$ relation is the same as Allen's *before* relation, that is, $\boldsymbol{a} = [a_s, a_f], \boldsymbol{b} = [b_s, b_f]$

$$\boldsymbol{a} < \boldsymbol{b} \Leftrightarrow a_f < b_s$$

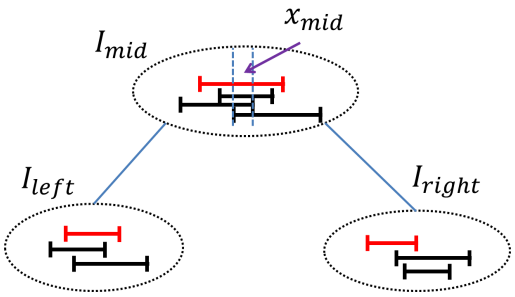



Fig. 6: Balanced tree for stabbing groups

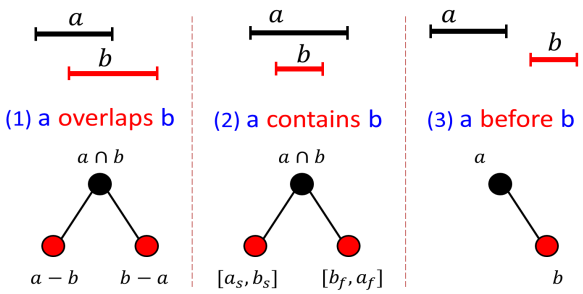



Fig. 7: Insert new intervals into the tree

To insert a new interval into the balanced tree, we first do binary tree search by comparing with the group representatives on the search path. Whenever the interval to be added overlaps the group representative of a node, it is added to that node, which may cause change to the group representative. Fig. 7 depicts how $\boldsymbol{b}$ is inserted into the tree. Suppose $\boldsymbol{a}$ is the group representative of the target group,

1. if $\boldsymbol{a}$ overlaps $\boldsymbol{b}$, the representative of the $\boldsymbol{a}$ is changed to $\boldsymbol{a} \cap \boldsymbol{b}$.In addition, $\boldsymbol{a} - \boldsymbol{b}$, and $\boldsymbol{b} - \boldsymbol{a}$ are added recursively to the left and right subtrees.
2. if $\boldsymbol{a}$ contains $\boldsymbol{b}$, the representative of the current group is changed to $\boldsymbol{a} \cap \boldsymbol{b}$.In addition, $[a_s, b_s]$, and $[b_f, a_f]$ are added recursively to the left and right subtrees.
3. if $\boldsymbol{a}$ before $\boldsymbol{b}$, if the right child is absent, $\boldsymbol{b}$ will be added as a new right child of $\boldsymbol{a}$.

### 3.3 Stabbing temporal aggregation(BTA)

We now describe the algorithm in detail. Since the temporal aggregates are computed based on stabbing groups, the algorithm is called *stabbing temporal aggregation* (BTA).

**Algorithm 1** BTA (Building aggregation tree)

**Input:** $\alpha$-folded null time intervals $S = \{[s_1, f_1], [s_2, f_2], \cdots, [s_n, f_n]\}$
**Output:** Temporal aggregates of $S$

```
1: for i ← 1 · ·n do
2:     b ← [s_i, f_i]
3:     t ← T.find(b)
4:     if t = nil then          ♯ Add a new node
5:         t.add(b)
6:     else          ♯ Update existing nodes recursively
7:         a ← t.ω
8:         t.ω ← a ∩ b
9:         t.Ω ← t.Ω ∪ {b}          ♯ For report
10:        t.C ← t.C + 1          ♯ For count
11:        L ← t.leftChild
12:        R ← t.rightChild
13:        if a overlaps b then
14:            L.insert(a − b)
15:            R.insert(b − a)
16:        else if a contains b then
17:            L.insert([a_s, b_s])
18:            R.insert([b_f, a_f]);
19:        end if
20:    end if
21: end for
22: return T
```

Given a temporal database $D$, the algorithm takes a query interval $[p, q]$ and the parameter $\alpha$ as input, and report all null time intervals that satisfy the length constraint. For count, report the number of qualifying null time intervals. The process of computing aggregates are outlined as follows.

1. Query the database $D$ for event times that intersect $[p, q]$;
2. Derive the null time intervals from the obtained event times;
3. Apply $\alpha$-folding to the null time intervals;
4. Create the balanced search tree for the $\alpha$-**folded** null time intervals;
5. Traverse the tree and report groups and their representatives.

In Algorithm 1, we use an AVL tree $T$ as the main data structure. $T$ is built by a recursively inserting intervals.

- $T.find()$: Search the tree/subtree rooted on $T$ for a given interval, return the first node whose representative intersecting with the argument interval
- $T.insert()$: Insert a node to the tree/subtree rooted on $T$
- $t.add()$: Add a new node to the current location $t$

The time complexity of BTA algorithm is the complexity of constructing a balanced search tree $O(n \log n)$. Notice step 2, 3 and 4 can be done simultaneously, which means the proposed method needs to scan the query result only once.

## 4 Experimental Analysis

To verify the proposed method, we implemented naive ITA and the proposed BTA algorithms for computing temporal aggregates over null time intervals. We evaluate algorithms in terms of response time and memory space. The response time includes database query, $\alpha$-folding, and temporal aggregation. We do experiments for two types of queries: *stabbing report* and *stabbing count*. In stabbing report, the detail of stabbing groups are reported, while in stabbing count, only count for each group is maintained and output.

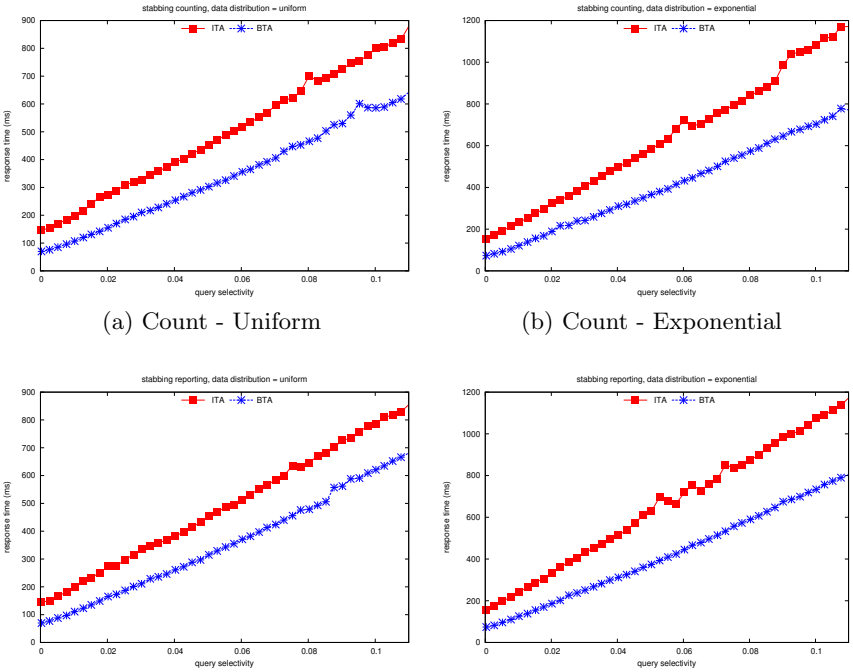
The input to the algorithm consists of $1,000$ event symbols and $1,000$ event times for each event. The starting times are random numbers with uniformly distributed on $[1 \cdot \cdot 10^6]$. The length of the random intervals are either uniform or short. Uniform random intervals have a uniform distributed among $[1, 2,000]$. Short random intervals have an exponentially distributed length with expected value $2,000$.

The results are shown in Fig. 8–Fig. 11. In each of the plots, $x$-axis represents query intervals, ranging from 1% to 11%. First, in terms of response time, BTA outperforms naive ITA under uniform dataset as well as exponential dataset and for both count and report queries. Database query time as a part of response time varies with the query ranges but it is not a dominant part (only 10% of response time, see Fig. 9). The aggregation time is the main part of response time has a similar trend with response time (Fig. 10). The memory space for BTA is significantly smaller than naive ITA (Fig. 11). In total, BTA provides better performance for computing null time aggregates.

## 5 Related Work

In [12], Kline and Snodgrass proposed an algorithm for computing temporal aggregation using a main memory-based data structure, called aggregation tree. It builds a tree while scanning a temporal relation. After the tree has been built, the answer to the temporal aggregation query is obtained by traversing the tree in depth-first search. It should be noted that this tree is not balanced. The worst case time to create an aggregation tree is $O(n^2)$ for $n$ stored tuples. In an extension of his previous work , Kline proposed using a $2-3$ tree, which is a balanced tree, to compute temporal aggregates [3]. The leaf nodes of the tree store the time intervals of the aggregate results. Like the aggregation tree, this approach requires only one database scan. The running time is $O(n \log n)$ given that the database was initially sorted.

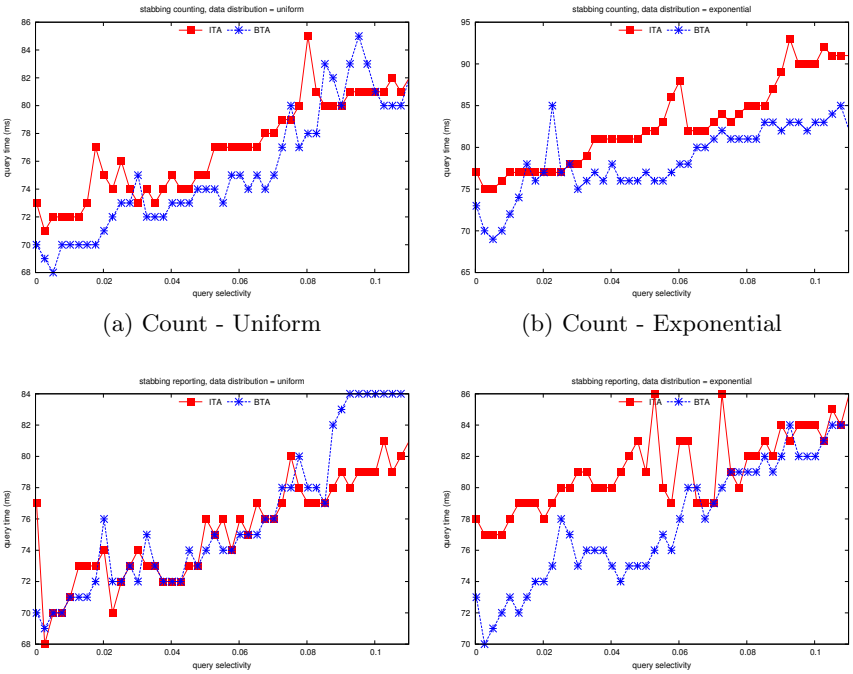
Interval stabbing problem is well-known problem in computational geometry and there exist a number of results, for example [10]. However, interval stabbing in this context is aimed to preprocess the intervals into a data structure for repetitive query. The query time is at least $\Omega(1 + k)$ for output size $k$. The preprocessing is often expensive, requires multiple scans.




(a) Count - Uniform (b) Count - Exponential

(c) Report - Uniform (d) Report - Exponential

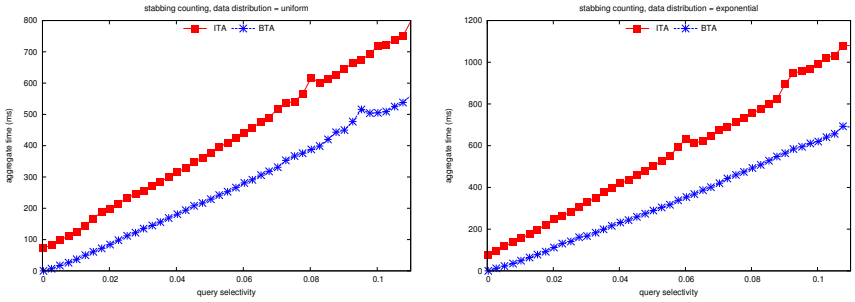
Fig. 8: Response times



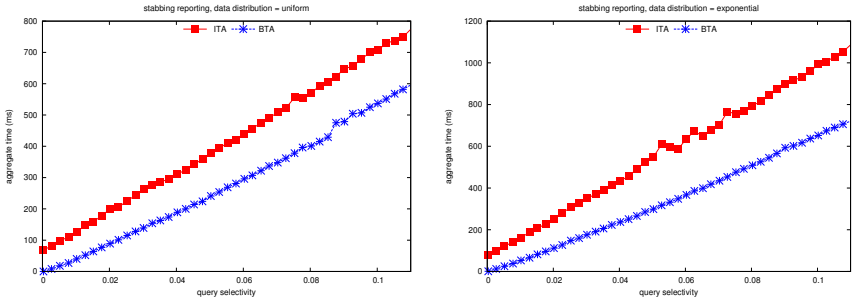

(a) Count - Uniform (b) Count - Exponential

(c) Report - Uniform (d) Report - Exponential

Fig. 9: Database query cost

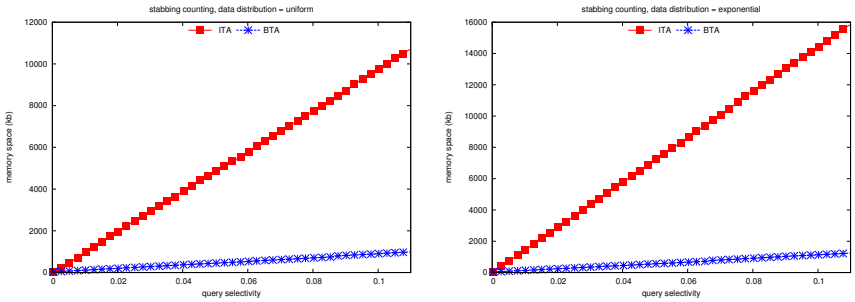



(a) Count - Uniform (b) Count - Exponential

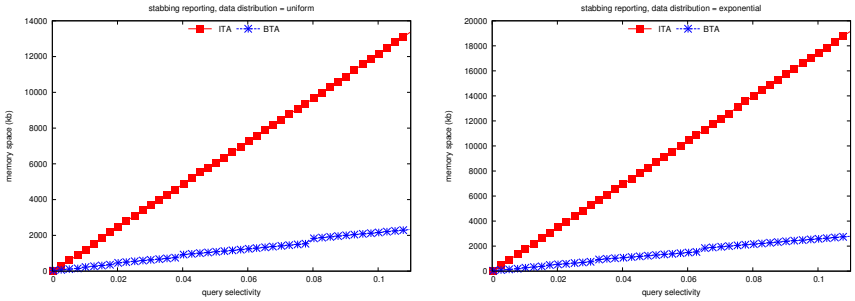



(c) Report - Uniform (d) Report - Exponential

Fig. 10: Aggregate computation cost




(a) Count - Uniform (b) Count - Expotential




(c) Report - Uniform (d) Report - Expotential

Fig. 11: Memory space

## 6 Conclusion

In this paper, we dealt with the problem of computing temporal aggregates over null time intervals. Null time intervals are intervals not overlapped by any event time intervals. We introduced $\alpha$-folding and transformed the problem into the interval stabbing problem. To compute aggregates for stabbing groups efficiently, we proposed balanced search tree based data structure that maintains stabbing groups and their associated aggregates. The proposed algorithm requires only scan the argument intervals exactly once.

## Acknowledgments

We would like to thank Prof. Yuichi Asahiro for advices that led to this paper. We also thank the anonymous reviewers for their valuable comments.

## References

1. James F. Allen. Maintaining Knowledge about Temporal Intervals. Communications of ACM, vol.26, issue 11, p.832–843, 1983
2. Nick Kline, Richard Thomas Snodgrass, Computing Temporal Aggregates, In Proceedings of the Eleventh International Conference on Data Engineering, p.222–231, March 06-10, 1995
3. Nick Kline, Aggregation in Temporal Databases, Ph.D thesis. Univ. of Arizona, May 1999.
4. Michael H. Böhlen, Richard Thomas Snodgrass, and Michael D. Soo. Coalescing in Temporal Databases. In Proceedings of the 22th International Conference on Very Large Data Bases (VLDB '96), San Francisco, CA, USA, p.180-191.
5. Kriegel, Hans-Peter, Marco Ptke, and Thomas Seidl. Managing Intervals Efficiently in Object-Relational Databases. In Proceedings of the 26th International Conference on Very Large Data Bases. p.407–418. Morgan Kaufmann Publishers Inc., 2000.
6. Ines Fernando Vega Lopez, Richard T. Snodgrass, Bongki Moon, Spatiotemporal Aggregate Computation: A Survey, IEEE Transactions on Knowledge and Data Engineering, v.17 n.2, p.271–286, February 2005
7. Michael Böhlen, Johann Gamper, et al. Multi-dimensional aggregation for temporal data. In: Proceedings of the 10th International Conference on Extending Database Technology, p.257–275. Springer, Berlin (2006)
8. Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. Computational Geometry (3rd revised ed.). Springer-Verlag, 2008.
9. Curtis E. Dyreson. Temporal coalescing with now granularity, and incomplete information. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data (SIGMOD '03). p.169–180. ACM, New York, NY, USA
10. Schmidt, J. M. (2009, December). Interval stabbing problems in small integer ranges. In International Symposium on Algorithms and Computation (pp. 163-172). Springer Berlin Heidelberg.
11. Juozas Gordevičius, Johann Gamper, Michael Böhlen, Parsimonious temporal aggregation, The VLDB Journal The International Journal on Very Large Data Bases, v.21 n.3, p.309–332, June 2012

12. N. Kline and R.T. Snodgrass, Computing Temporal Aggregates, Proc. Intl Conf. Data Eng., pp. 222-231, Mar. 1995.
13. Kai Cheng. Approximate Temporal Aggregation with Nearby Coalescing, DEXA (2) 2016: 426-433.
14. Martin Kaufmann, Amin Amiri Manjili, Panagiotis Vagenas, Peter Michael Fischer, Donald Kossmann, Franz Frber, Norman May. Timeline index: a unified data structure for processing queries on temporal data in SAP HANA, In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD 2013). p.1173–1184. 2013