データベースシステム 116－36
（１９９８．７．９）

# 事象による検索を考慮したタイムインデックスの性能評価

天笠 俊之　櫻井 紀美子　有次 正義　金森 吉成

群馬大学工学部情報工学科

{amagasa, sakurai, aritsugi, kanamori}@dbms.cs.gunma-u.ac.jp

本稿では，時間データベースにおける検索処理のためのインデックス構造を提案する．このインデックスはタイムインデックスに基づいており，既存のインデックス構造では効率よく処理することができない，事象をキーにした時区間の検索を効果的に処理することができる．また事象の休止期間を表す空時区間を格納することができるので，事象の存在と同様，事象の休止期間に対しても検索を行うことができる．さらに，このインデックスの有効性を検証するために，タイムインデックス，R*-tree と性能の比較を行い，既存のインデックスでは，キーとなる事象の集合のサイズに比例した処理時間がかかるのに対して，提案したインデックスでは，キーのサイズが大きい場合でも高速な処理が可能であることを示す．

# An Evaluation of Enhanced Time Index for Event Queries

Toshiyuki Amagasa　Kimiko Sakurai　Masayoshi Aritsugi　Yoshinari Kanamori

Department of Computer Science, Gunma University

In this paper, a new indexing structure for temporal queries is proposed. It is based on the Time Index, and can effectively process event queries including a lot of events. In addition, it can maintain null-time intervals which represent suspension of events, thereby, we can retrieve time intervals in which events are suspended as well as those in which events exist efficiently. We compare the performance of the index with Time Index and R*-tree. The results shows that the performance of conventional indexing structures is affected as the number of events grows. On the other hand, our index can efficiently process event queries even if the number of events is quite large.

## 1 はじめに

時間データベースにおいて，実世界の時間的側面をモデル化し，モデル化されたデータに対して効率の良い検索を行うことは重要な課題である．前者の目的のために，我々は時区間概念モデル [3] を提案している．

我々のモデルが対象としているのは，図 1に示すような実世界の履歴データである．これはある患者に対する投薬履歴を表している．図において，$a$, $b$, $c$, $d$ は薬の種類であり，横軸は時間軸である．各薬の履歴において，ハッチングされた部分はその薬が患者に投与されていたことを示しており．白抜きの部分は投与が休止されていたことを示している．このような構造は投薬履歴だけに見られるものではない．例えば，$a$, $b$ を動画像の再生時間，$c$, $d$ を音声の再生時間とすれば，この図はマルチメディアデータの同期再生スケジュールと見ることもできる．その他の応用例として時系列画像のモデル化を挙げることができる [4, 13].

時区間概念モデルは，空時区間 [3, 10] を導入することによって，事象が休止していた期間を明示的に表現することができる．また，時区間の概念のみに基づいているので，時区間を表現できるようなどのようなデータベースモデルに対しても適用が可能である．また我々はこのモデルをオブジェクトデータベースのクラスライブラリとして実装している [2]．これは履歴情報を扱う必要のあるどのようなアプリケーションプログラムからも利用することができる，汎用のクラスライブラリである．

問い合わせ処理の効率化に関しては，これまで多くの研究者によって多種多様なインデックス構造が提案されている [5, 6, 7, 8, 9, 11, 14]．時間データベースに対する問い合わせは，大まかに，(1) 時間をキーにした事象または時区間の検索，(2) 事象をキーにした時間の検索の二種類に分類できる．図 1に示した投薬履歴に対する問い合わせを例にとると，「治療の 5 日目から 10 目までに投与された薬は？」という問い合わせは上記 (1) に相当し，「薬 $a$ と $c$ が同時に投与されていたのはいつか？」という問い合わせは (2) に相当する．上に挙げた既存のインデックス構造は，主に時間をキーにした検索を対象にしており，事象をキーにした検索を考慮したものは少ない．また，事象を考慮したインデックスでも，個々の事象を独立して格納するものがほとんどであるために，大量の事象にまたがった検索を行いたい場合には，事象の数に比例した処理時間が要求される．

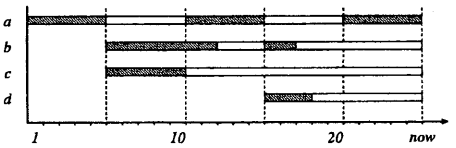


図 1: ある患者に対する投薬履歴

本稿では，時間をキーにした検索だけでなく，事象をキーにした検索に関しても効率の良い検索を行うことができるインデックス構造を提案する．これはタイムインデックス [7] に基づいている．また，時区間概念モデルの特徴である空時区間を格納できるので，事象が休止していた期間に対する問い合わせも，事象が存在していた期間同様に処理することができる．

本稿の構成は以下の通りである．第 2 章では時区間概念モデルの概要について述べる．第 3 章では事象の検索を考慮したタイムインデックスについて述べる．第 4 章で提案したインデックスの性能評価を行う．第 5 章では関連研究との比較を述べる．第 6 章はまとめである．

## 2 時区間概念モデルの概要

時区間概念モデル [3] では，事象の履歴を時区間によって表現する．本モデルでは特に，時区間を事象が存在したことを表す実時区間と，事象が休止していたことを表す空時区間とに区分して扱う．一般に，実世界の事象は時間の経過と共に出現と消滅を繰り返すので，この様子は実時区間と空時区間を交互に並べることによってモデル化することができる．そこで実時区間と空時区間が交互に *meets*[1] しているような時区間の集合を複合時区間と定義する[1]．例えば，図 1に示した投薬履歴は，複合時区間によって以下のように記述される．

$$
\begin{aligned}
C_1 &= \{(1,5;\{a\}),(6,10;\{\bar{a}\}),(11,15;\{a\}),\\
&\quad (16,20;\{\bar{a}\}),(21,now;\{a\})\}\\
C_2 &= \{(6,12;\{b\}),(13,15;\{\bar{b}\}),(16,17;\{b\}),\\
&\quad (18,now;\{\bar{b}\})\}\\
C_3 &= \{(6,10;\{c\}),(11,now;\{\bar{c}\})\}\\
C_4 &= \{(16,18;\{d\}),(19,now;\{\bar{d}\})\}
\end{aligned}
$$

$C_1$ の最初の要素は，薬 $a$ が 1 日目から 5 日目まで投与されていたことを表す実時区間である．二番目の要素は，薬 $a$ の投与が 6 日目から 10 日目まで休止していたことを表す空時区間であり，$\bar{a}$ は事象の休止を表している．

[1] これに対して，単に時区間を時間順に並べた時区間の集合を収集と定義する．複合時区間は収集の特別な場合である．

時区間概念モデルでは，時間に対する問い合わせを時区間上の演算によって記述する．その例を以下に示す．

**[例 1]** 薬 $a$ を使った治療は，11 日目から現在まで何度休止したか？

$$cardinality(null(C_1) \cup (11, now))$$

**答え：1 回**

最初に *null* 演算によって $C_1$ から空時区間を抽出する．これは治療が休止していた期間を表している．次に共通演算 ($\cup$) によって，$(11, now)$ との共通部分が抽出される．最後に *cardinality* 演算によって，休止した回数を数え，答えを得ることができる．これは時間をキーにした問い合わせの一例である．

**[例 2]** 薬 $a$ が投与され，それ以外の薬 $b$, $c$, $d$ の投与が休止していた期間はいつか？

$$include(C_1 \hat{\uplus} C_2 \hat{\uplus} C_3 \hat{\uplus} C_4, \{a, \bar{b}, \bar{c}, \bar{d}\})$$

**答え：21 日目から現在まで**

被覆演算 ('$\hat{\uplus}$') は，引数として与えられた二つの収集（複合時区間）を一つの収集に統合する演算である．これを何度も適用することによって，複数の事象に関する履歴を一つの収集にまとめることができる．

$$\begin{aligned} C_1 \hat{\uplus} C_2 \hat{\uplus} C_3 \hat{\uplus} C_4 = \{&(1,5;\{a\}),(6,10;\{\bar{a},b,c\}),\\ &(11,12;\{a,b,\bar{c}\}),(13,15;\{a,\bar{b},\bar{c}\}),\\ &(16,17;\{\bar{a},b,\bar{c},d\}),(18,18;\{\bar{a},\bar{b},\bar{c},d\}),\\ &(19,20;\{\bar{a},\bar{b},\bar{c},\bar{d}\}),(21,now;\{a,\bar{b},\bar{c},\bar{d}\})\}. \end{aligned}$$

*include* 演算は収集の中から指定した事象が含まれる時区間を抽出する．この例では，事象に $\{a, \bar{b}, \bar{c}, \bar{d}\}$ を含む $(21, now; \{a, \bar{b}, \bar{c}, \bar{d}\})$ が抽出されるので，21 日目から現在までが答えだと分かる．これは事象をキーにした問い合わせの一例である．

時区間概念モデルの特徴の一つは，被覆演算によって複数の事象に関する履歴を一つの収集に統合できることである．この結果から任意の事象が含まれる時区間を抽出することが可能である．また，モデルに事象の休止が明示的に取り込まれているので，事象の休止期間に対する問い合わせも，事象の存在していた期間と同様に行うことができることも特徴の一つである．

## 3 事象を考慮したタイムインデックス

前章で述べたように，時区間概念モデルは時間をキーにした問い合わせと事象をキーにした問い合わせのどちらも扱うことができる．しかしながら，既存の

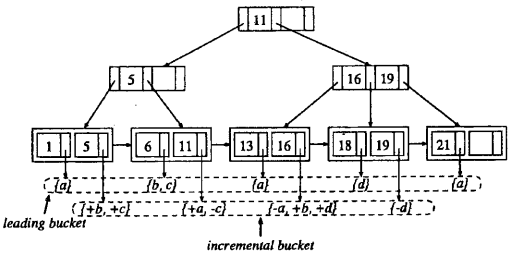


図 2: タイムインデックス

インデックス構造では事象の検索を効果的に処理することができない．さらに，本モデルの特徴である空時区間を格納することもできない．

そこで本章では，時区間概念モデルにおける問い合わせ処理を効率的に行うことができるインデックス構造を提案する．これは Elmasri 等によるタイムインデックス [7] を元に改良を加えたものである．最初に，タイムインデックスの概要を述べる．

### 3.1 タイムインデックスの概要

タイムインデックス [7] は $B^+$-tree 上に構成される．タイムインデックスでは，時区間を格納するために，時区間が変化する点に着目する．これをインデックスポイントと呼ぶ．インデックスポイントは，(1) 新たな時区間が始まるか，(2) ある時区間が終わった直後に作られる．各々のインデックスポイントにおいて，その時間に有効だった事象が保持される．

実際の時間データベースでは，バケットに格納される事象の数は膨大であるが，そのほとんどが前後のバケットと重複するという性質を持つ．タイムインデックスでは，この冗長な部分を減らすために，各リーフノードの先頭のバケット (leading bucket) でのみ全ての事象を格納し，続くバケット (incremental bucket) では事象の増減のみを記録する．例えば，図 1に示した投薬履歴をタイムインデックスに格納した例が図 2である．図において，時刻 11 は増減を記録したバケットである．$+a$ はその時刻で事象 $a$ が開始したことを表し，$-c$ はその直前に $c$ が終了したことを示している．

### 3.2 事象を考慮したタイムインデックス

時区間概念モデルの問い合わせ処理にタイムインデックスを用いようとすると，以下の問題が生じる．

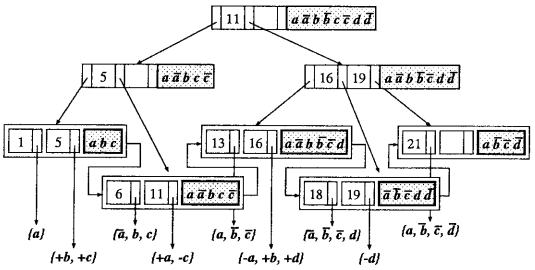


図 3: 事象を考慮したタイムインデックス

(1) タイムインデックスは事象による検索を効率よく処理することができない

(2) タイムインデックスは空時区間を格納することができない

これらの問題点を解消するために，タイムインデックスのバケットと内部ノードにそれぞれ変更を加える．

#### 3.2.1 事象を考慮したタイムインデックスの内部ノードの構成

タイムインデックスで事象をキーに時区間を検索するためには，全てのリーフノードを走査し，全てのバケットに関して事象が有効であったかどうかを調べなければならない．この方法では，問い合わせ処理のために，インデックスポイントの数に比例した処理時間が必要になってしまう．

そこで木の探索空間を減らすために，各ノードにおいて，そのノードから下の部分木に含まれる全ての事象の集合を保持させることにする．図3は，図2のタイムインデックスに，この変更を加えたものである．例えば時刻に5を持った左端の内部ノードでは，事象の集合として $\{a, \bar{a}, b, c, \bar{c}\}$ を保持している．これはそのノードから下の部分木にこれらの事象が含まれることを示している．このインデックスに事象の検索を行う場合，このノードを訪れた時点で $\bar{b}$, $d$, $\bar{d}$ は左部分木には含まれないことが分かるので，これらの事象を対象とした検索では，左部分木を探索する必要がなくなる．このように内部ノードに事象の情報を持たせることによって，探索空間を減らすことができる．

#### 3.2.2 事象を考慮したタイムインデックスのバケットの構成

タイムインデックスでは空時区間を格納することができないので，バケットの構成を変更することによって空時区間を格納できるようにする．まず，各リーフノードの先頭のバケットでは，事象の定義を以下のように変更する．

- 事象 $a$ が存在している．すなわち実時区間である場合，バケットには $a$ を挿入する．
- 事象 $a$ が休止している．すなわち空時区間である場合，バケットには $\bar{a}$ を挿入する．

続いて，それ以外のバケットにおける事象の解釈を以下のように変更する．

- $+a$ は事象 $a$ が新たに出現したか，または空時区間から実時区間に変化したことを表す．
- $-a$ は逆に事象 $a$ が実時区間から空時区間へ変化したことを表す．

以上の変更によって，タイムインデックスに空時区間を格納する．

### 3.3 アルゴリズム

本節では，事象を考慮したインデックスにおける探索，挿入のアルゴリズムを説明する．

#### 3.3.1 事象をキーにした検索

アルゴリズム **SearchEvent** は，与えられた事象の集合をキーに，それらの全てが有効であった時区間を検索する．実際には，与えられた事象の集合が存在していた期間とそうでない期間との変化点を検索し，それらを集合として返す．このアルゴリズムは，各ノードを訪れた際，まずそのノードが検索対象の事象を含むかどうかを各ノードが持つ事象の集合によって判断し，対象となる事象が含まれるノードだけを探索する．

```
SearchEvent(E)
Input E: set of events
begin
    n ← root node of enhanced Time Index;
    return SearchEventSub(E, NotExist, n);
end

SearchEventSub(E, s, n)
Input E: set of events
    s: current status of event set E
    n: node of enhanced Time Index
begin
    R ← φ; (* R: result containing set of time points *)
    if (n is not a leaf node) then
```

```
        q ← number of tree pointers in node n;
        foreach i in 1 ≤ i ≤ q begin
            if (E ⊆ n.P_i.E) then
            (* n.P_i: the i^th tree pointer of node n *)
            (* n.E: the set of events in n *)
                R ← R ∪ SearchEventSub(E, s, n.P_i);
        end
    else
        ps ← s; (* ps: previous status *)
        q ← number of data buckets in node n;
        foreach i in 1 ≤ i ≤ q begin
            if (E ⊆ n.B_i) then
            (* n.B_i: i^th full bucket in node n *)
                s ← Exist;
            else
                s ← Suspend;
            if (s ≠ ps) then
                if (s = Exist) then
                    R ← R ∪ n.K_i;
                    (* n.K_i: the i^th key in node n *)
                else
                    R ← R ∪ −n.K_i;
                endif
            endif
        end
    endif
    return R;
end
```

### 3.3.2 事象の内部ノードへの挿入

アルゴリズム **InsertEvent** は与えられた時区間に含まれる事象を，適切な内部ノードに挿入するアルゴリズムである．

```
InsertEvent(t)
Input t: time interval (t_s, t_e; E)
begin
    n ← root node of enhanced Time Index;
    InsertEventSub(t, n);
end

InsertEventSub(t, n)
Input t: time interval (t_s, t_e; E)
    n: node of enhanced Time Index
begin
    if (n is not a leaf node) then
        n.E ← n.E ∪ E;
        if (t_s ≤ n.K_1) then
            i ← 1;
        else
            search i such that n.K_i < t_s ≤ n.K_{i+1};
        endif
        if (t_e ≤ n.K_1) then
            j ← 1;
        else
            search j such that n.K_j < t_e ≤ n.K_{j+1};
        endif
        foreach k such that i ≤ k ≤ j begin
            InsertEventSub(t, n.P_i);
        end
    else (* n is a leaf node *)
        n.E ← n.E ∪ E;
    endif
end
```

これは通常のタイムインデックスの挿入が行われた後に用いられる．従って，事象を考慮したタイムインデックスにおける時区間の挿入は，通常のタイムインデックスに比べて事象を内部ノードに挿入する分，コストが余計にかかってしまう．しかし，実際の時間データベースにおける挿入は，ほとんどの場合が時間の経過に伴う新たな時区間の追加であると考えられる．この場合，このアルゴリズムは木の一番右側のノードをリーフノードまで辿って事象を挿入するだけでよく，そのコストはそれほど大きくないと考えられる．

### 3.3.3 事象の内部ノードへの一括挿入

前節で説明した挿入アルゴリズムは，少量の時区間をその都度インデックスに挿入したい場合に有効であるが，木の初期状態を作る時のように，大量の時区間を一度にインデックスに挿入したい場合には効率的でない．一つの時区間を挿入する度に，その事象のコピーを内部ノードに作成する手間が掛かってしまうためである．アルゴリズム **SetupEvent** は，このような場合に有効である．まず通常のタイムインデックスの挿入アルゴリズムによって，タイムインデックスを構成した後に，**SetupEvent** を適用する．**SetupEvent** はバケットに含まれる事象を，適切な内部ノードに一度にコピーするので，**InsertEvent** より効率が良いと考えられる．

```
SetupEvent()
begin
    n ← root node of enhanced Time Index;
    return SetupEventSub(n);
end

SetupEventSub(n)
n — node of enhanced Time Index
begin
    if (n is not a leaf node) then
        q ← number of tree pointers in node n;
        foreach i such that 1 ≤ i ≤ q begin
            n.E ← n.E ∪ SetupEventSub(n.P_i);
        end
```

表 1: 実験環境

| | |
|---|---|
| machine type | Sun Ultra 30 |
| CPU | UltraSPARC-II (296 MHz) |
| memory size | 128 MB |
| OS | Solaris 2.5.1 |
| disk drive | Sun 4.0 GB |
| compiler | SPARCompiler 4.2 C++ |
| database system | ObjectStore 5.0.0.0 |

```
    else (* n is a leaf node *)
        q ← number of data buckets in node n;
        foreach i such that t_s ≤ n.K_i ≤ t_e begin
            n.E ← n.E ∪ n.B_i
        end
    endif
    return n.E;
end
```

## 4 性能評価

本章では，事象を考慮したタイムインデックスの性能を評価する．具体的には通常のタイムインデックス，R*-tree との比較を行う．R*-tree では，時区間を格納するために，時区間 $(t_s, t_e; event)$ を三次元空間内の点 $(t_s, t_e, event)$ に射影した．

実験に使用した計算機環境は，表 1に示す通りである．データベースシステムには ObjectStore[12] を用いた．実験に用いた時区間のデータは以下の通りである．(1)100 個の事象が存在する，(2) 各時区間の開始時刻は $[1, 10000]$ に一様分布する．(3) 各事象が持つ実時区間の個数は $[1, 100]$ に一様分布する．

生成した時区間をそれぞれのインデックスに挿入し，データベースのサイズを計測した結果が表 2である．表から，R*-tree のサイズが一番大きいこと，事象を考慮したタイムインデックスは通常のタイムインデックスよりもサイズが大きいことが分かる．事象を考慮したタイムインデックスでは，各ノードに事象のデータを保持しているために，通常のタイムインデックスよりもサイズが大きいと考えられる．

図 4は，タイムインデックス，事象を考慮したタイムインデックス，R*-tree における時間をキーにした検索の処理時間である．この実験では，問い合わせのキーとなる時区間の範囲を 1 から 1,000 まで変化させ，それぞれの値において問い合わせを 100 回処理するのにかかった時間を測定している．縦軸は時間，横軸は検索範囲である．結果から，(1) 二種類

表 2: データベースサイズ

| Index | Size (Kbytes) |
|---|---|
| Time Index | 3,584 |
| Enhanced Time Index | 4,608 |
| R*-Tree | 5,632 |

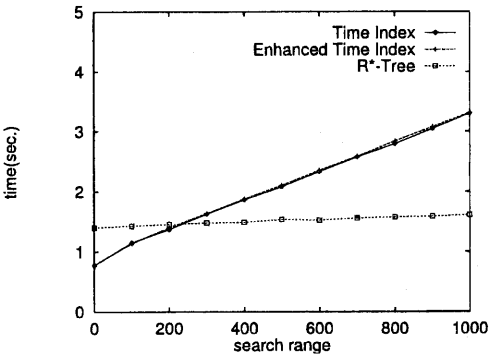


図 4: 時間をキーにした検索

のタイムインデックスは，検索範囲の大きさに比例して処理時間が増加している．一方，R*-tree では検索範囲の大きさは，性能にほとんど影響を与えない．(2) 検索範囲が狭い時にはタイムインデックスが有利であり，広い場合には R*-tree の方が有利である．(3) 事象を考慮したタイムインデックスでは，通常のタイムインデックスよりわずかに処理が遅いこと，などが分かる．(3) は，事象を考慮したタイムインデックスでは，内部ノードに事象の情報を持っている分データベースのサイズが大きくなっていることに起因すると思われるが，その差は無視できる程度である．

図 5は，事象を考慮したタイムインデックスと R*-tree における事象をキーにした検索の処理時間である．この実験では 100 個の事象の中から $n$ 個の事象をランダムに選択し，それが全て含まれる時区間を検索している．縦軸は時間，横軸は $n$ である．また，通常のタイムインデックスは事象の検索に膨大な時間がかかるために省略している．図から，検索対象となる事象の数が少ない場合には R*-tree の方が性能が良いが，事象の数が大きくなるにつれて事象を考慮したタイムインデックスの方が性能が良くなることがわかる．これは，事象の数が少ないときには探索すべきノードが多く，事象の数が増えるにつれて検索対象を含むノードの数が絞り込まれ，探索範囲が狭くな

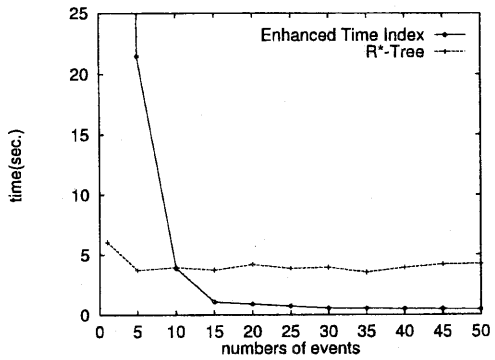


図 5: 事象をキーにした検索 (1)

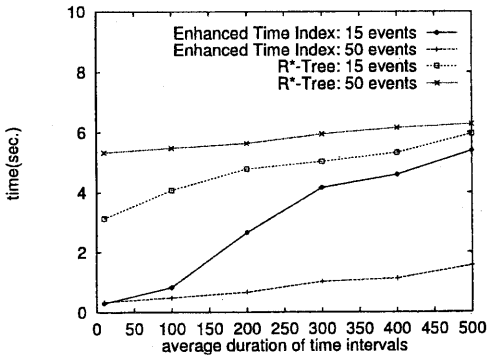


図 6: 事象をキーにした検索 (2)

ることを示している．すなわち，検索対象となる事象の数が多ければ多いほど事象を考慮したタイムインデックスの方が有効であると言える．また，検索したい事象の数が少ないときは R*-tree に比べてかなり性能が悪いが，この場合はインデックスではなく，直接時区間のデータを参照すれば良いので，問題にはならないと考えられる．

図 6は，インデックスに挿入される時区間の持続期間を変化させたときに，事象をキーにした検索に必要な時間の変化である．縦軸は時間，横軸は時区間の平均持続期間である．また，それぞれのインデックスにおいて，キーとなる事象の数が 15 個と 50 個の場合について測定を行った．結果より，全てのインデックスが時区間の持続期間に影響を受けていることが分かる．これは時区間の持続期間が長くなることによって時区間が重複する部分が増えることによる．特に事象を考慮したタイムインデックスにおいて，少ない事象を対象に検索を行った場合は影響が顕著である．しかし，総じて R*-tree よりも我々のインデックスの方が性能が良いと言える．

## 5 関連研究

Elmasri 等は時間と事象をキーにした検索のために，タイムインデックスを二段に重ねたインデックス (2-level combined attribute/timeindex) を提案している [7]．このインデックスにおいて，上段のインデックスは事象に関して構成された $B^+$-tree である．また，リーフノードの各エントリはタイムインデックスへのポインタになっている．すなわち，各事象に関して一つのタイムインデックスが存在する．

Gunadhi 等は AP-tree と入れ子 ST インデックスを提案している [8]．これは上で述べた Elmasri 等のインデックスと類似した構造を持つ．すなわち，二段に構成された最上段のインデックスは事象に関して張られ，下段のインデックスは各事象の履歴情報を保持する．

現在活発に研究されている多次元インデックス [5, 6, 9] においても，時区間 $(stp, etp; event)$ を三次元空間内の点 $(stp, etp, event)$ に射影することによって，時間や事象に関する検索を行うことが可能である．

しかし，これらのインデックス構造のいずれにおいても，各事象の履歴は独立に管理されている．このため，事象をキーに時区間を検索しようとすると，まず各事象の履歴を検索し，最後にその結果を併合しなければならない．それゆえ，事象を大量に含むような検索を処理する場合には，事象の数に比例した処理時間が必要になる．一方，本稿で提案しているインデックスでは，全ての事象を単一のインデックスで管理する．また事象をキーにした検索では，内部ノードに保持した事象の情報によって，探索範囲を絞り混むことができるので，検索対象が増えても，効率の良い検索処理を行うことが可能である．

## 6 まとめ

本稿では，我々が提案している時区間概念モデルの問い合わせを効率よく処理するためのインデックス構造として，事象を考慮したタイムインデックスを提案した．このインデックスでは，事象をキーにした検索を効率よく行うために，各ノードにそのノードから下の部分木に含まれる事象の情報を保持させる．これによって，事象をキーにした検索を効果的に処

理することが可能である．また，事象の休止期間を表す空時区間を明示的に格納できるため，事象の休止期間に対しても，存在期間と同様に検索処理を行うことができる．また，タイムインデックス，R*-tree と性能を比較し，有効性を示した．

今後は，実世界のデータに対して本インデックスを適用し，実データに対してどの程度有効であるかを検証する予定である．

## 謝辞

金森研究室の田中貴之君には R*-tree の実装をして頂いた．ここに記して謝意を示す．また，本研究の一部は，文部省科研費重点領域研究「高度データベース」(課題番号 08244101) によるものである．

## 参考文献

[1] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol.26, pp.832–843, Nov. 1983.

[2] T. Amagasa, M. Aritsugi, and Y. Kanamori, "Implementing Time-Interval Class for Managing Temporal Data," *Proc. Workshop on DEXA'98*, Aug. 1998 (to appear).

[3] T. Amagasa, M. Aritsugi, Y. Kanamori, and Y. Masunaga, "Interval-based modeling for temporal representation and operations," *IEICE Trans. on Info. and Syst.*, vol. E81–D, no.1, pp.47–55, Jan. 1998.

[4] M. Aritsugi, T. Tagashira, T. Amagasa, and Y. Kanamori, "An approach to spatio-temporal queries – Interval-based contents representation of images –," *Proc. DEXA'97*, LNCS 1308, pp.202–213, Sep. 1997.

[5] N. Beckmann, H-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectanlges," *Proc. ACM SIGMOD Conf.*, pp.322–331, 1990.

[6] S. Berchtold, D. Keim, and H-P. Kriegel, "The X-Tree: An index structure for high-dimensinal data," *Proc. VLDB Conf.*, pp.28–39, 1996.

[7] R. Elmasri, G. Wuu, and V. kouramajian, "The time index and the monotonic B+-tree," in *Temporal databases: theory, design and implementation*, Chapter 18, A. Benjamin/ Cummings, 1993.

[8] G. Gunadhi and A. Segev, "Efficient indexing methods for temporal relations," *IEEE Trans. Knowledge and Data Engineering*, pp.496–509, vol.5, no.3, June 1993.

[9] A. Guttman, "R-trees: A dynamic index structure for spatial searching," *Proc. ACM SIGMOD Conf.*, pp.47-57, 1984.

[10] Y. Masunaga, "A temporal expansion to the multimedia object model in OMEGA," *Proc. DASFAA'95*, pp.430–440, Apr. 1995.

[11] 仲篤起，才脇直樹，辻本浩章，西田正吾，「時空間データの高速検索のためのデータ管理方式」情報処理学会, *Proc. ADBS'96*, pp.71–78, Dec. 1996.

[12] Object Design, Inc., "ObjectStore C++ API user guide, " ObjectStore relese 4.0.2, June 1996.

[13] T. Tagashira, T. Amagasa, M. Aritsugi, and Y. Kanamori, "Interval-based representation of spatio-temporal concepts," *Proc. CAiSE'97*, LNCS 1250, pp.231–244, June 1997.

[14] H. Shen, B. Ooi, and H. Lu, "The TP-Index: A dynamic and efficient indexing mechanism for Temporal Databases," *Proc. ICDE*, pp.274–281, 1994.