Una propuesta de un método de acceso espacio-temporal: I+3 R-Tree

Edilma O. Gagliardi [(1)], **Fernando D. Carrasco,**
Juan C. García Sosa
Departamento de Informática
Facultad de Ciencias Físico,
Matemáticas y Naturales
Universidad Nacional de San Luis, Argentina
{oli, fdcarras, jcgarcia}@unsl.edu.ar
y

Gilberto Gutiérrez Retamal
Departamento de Auditoría e Informática
Facultad de Ciencias Empresariales
Universidad del Bío-Bío, Chile
ggutierr@ubiobio.cl

Abstract

This article presents the offer of a spatio-temporal index structure which deals with objects that change their location and/or shape over time. This access method support spatio-temporal queries considering past and present time. This method supports the following queries: a) *timeslice,* recover all the present objects in a certain region in a given instant, b) *interval*, , recover all the present objects in a certain region in a given time interval, c) *event,* recover all the objects that entered or went out of a certain region and d) *trajectory,* recover the path followed of an object. Our aim is to offer a contribution to the developing spatio-temporal access methods researches.

Keywords: spatio-temporal databases, spatio-temporal access methods.

Resumen

Este artículo presenta la propuesta de una estructura de indexación de datos espacio-temporales para objetos que se mueven en una región fija a lo largo del tiempo. Este método de acceso permite procesar consultas que contemplen relaciones espacio-temporales, considerando no sólo el pasado sino también el presente. El método considera el procesamiento de cuatro tipos de consultas, a) *timeslice*, que permite recuperar todos los objetos presentes en una cierta región en un instante dado *t,* b) *intervalo* la cual permite, recuperar todos los objetos existentes en una cierta región en el intervalo de tiempo $[t_1,t_2]$, c) *evento,* que permite recuperar todos los objetos que entraron/salieron en una determinada región y d) *trayectoria* que permite, recuperar el camino que ha seguido un objeto. Nuestro objetivo es brindar un aporte a las investigaciones que se llevan a cabo para desarrollar métodos de indexación de información espacio-temporal, proponiendo un nuevo método que permita ejecutar de manera eficiente las mencionadas consultas.

Palabras claves: Bases de Datos Espacio-temporales, Métodos de acceso Espacio-temporal.

(1) Este trabajo está enmarcado dentro de la Línea de Investigación Geometría Computacional y Bases de Datos Espacio-Temporales, perteneciente al Proyecto Tecnologías Avanzadas de Bases de Datos 22/F314, Departamento de Informática, Universidad Nacional de San Luis; en el Proyecto AL06_PF_013 Geometría Computacional, subvencionado por a Universidad Politécnica de Madrid; y en el marco de la Red Iberoamericana de Tecnologías del Software (RITOS2), subvencionado por CYTED.

1 INTRODUCCIÓN

Hay aplicaciones del mundo real donde los objetos de interés se caracterizan por mantener dos atributos comunes: el espacio y el tiempo. De esta forma, un objeto se puede identificar por la posición y el área que ocupa en cualquier instante de tiempo; por ejemplo, en un sistema de gestión catastral, se puede dar el caso de que una parcela cambie su forma como consecuencia de que el propietario ha comprado una parcela adyacente. Para contemplar este cambio se crea un nuevo objeto cuya forma esta determinada por la unión de las dos anteriores, pero, ¿cómo establecer una relación entre el nuevo objeto y los dos anteriores? Estos objetos dan origen al tipo de dato espacio-temporal. Así, aquellas aplicaciones informáticas cuya función sea manejar este tipo de objetos deberán proveer un soporte para manejar este tipo de datos, dando origen a los Sistemas de Administración de Bases de Datos Espacio-Temporales (STDBMS). Estos sistemas deben tener capacidades vinculadas a la definición de tipos de datos espacio-temporales, lenguajes de consultas y Métodos de Acceso Espacio-Temporal (MAETs).

En particular, para la indexación de información espacio-temporal, un STDBMS debe proveer MAETs que eviten examinar todos los objetos al momento de responder consultas, y que sean eficientes frente a una alta asiduidad de consultas. Estos índices son estructuras que permiten organizar un conjunto de datos con el objetivo de responder eficientemente distintos tipos de consulta, evitando accesos innecesarios a ciertas partes de la base de datos. La indexación se utiliza para optimizar la recuperación de datos almacenados en memoria secundaria y el tiempo de procesamiento de las consultas.

Los métodos de indexación propuestos se pueden categorizar básicamente en tres áreas:

Métodos para la recuperación de información histórica

Este tipo de MAETs tiene como objetivo recuperar de manera eficiente información espacio-temporal histórica acerca de objetos que se mueven (aparecen, desaparecen, cambian) en el tiempo. Los métodos de acceso relacionados a esta área son RT-Tree [10], HR-Tree [4, 3], 3D R-Tree [8], HR^{+}-Tree [7], 2+3 R-Tree [2] y OLQ (Overlapping Linear Quadtree) [9].

Métodos para la recuperación de la trayectoria

Ciertas aplicaciones necesitan almacenar patrones de movimiento de objetos. El objetivo en este caso es la eficiente recuperación de segmentos de línea que representen la trayectoria de determinados objetos. El TB-Tree [5] y STR-Tree [5] son los métodos de acceso enmarcados dentro de esta área.

Métodos para la predicción de la localización

Este tipo de problemas surge en aquellos sistemas que predicen las posiciones futuras de los objetos basándose en su localización actual y patrones de movimiento. Un método de acceso que permite la predicción de localización es el TPR-Tree [6].

En este trabajo nos abocamos al desarrollo de un nuevo MAET, con el objetivo de contribuir a las investigaciones en el área de los STDBMS. Esta propuesta consiste en el uso de un R-Tree de tres dimensiones, llamado 3D R-Tree, para almacenar datos históricos, y el uso de una nueva estructura denominada I para guardar datos correspondientes a las posiciones actuales de los objetos. El uso de un 3D R-Tree nos permitirá recuperar información relacionada a la permanencia de los objetos en determinadas posiciones o regiones, en instantes de tiempo pasados. Mientras que la segunda estructura se propone con el objeto de recuperar las posiciones actuales de los objetos; es decir, ubicaciones donde se hallan finalmente los mismos. Así, con este método de acceso es posible procesar consultas cuyos predicados contemplen relaciones espacio-temporales, considerando no sólo el pasado sino también el presente. Nuestro método permite procesar consultas

espacio/temporales de cuatro tipos: a) *timeslice*, que permite recuperar todos los objetos presentes en una cierta región en un instante dado *t,* b) *intervalo*, la cual permite recuperar todos los objetos existentes en una cierta región en el intervalo de tiempo $[t_1,t_2]$, c) *evento,* que permite recuperar todos los objetos que entraron / salieron en una determinada región y d) *trayectoria* que permite, recuperar el camino que ha seguido un objeto.

El resto de este artículo está estructurado de la siguiente manera. En la sección 2 se analizan las estructuras de indexación relacionadas, discutiendo las ventajas y desventajas de las mismas. La sección 3 describe el método propuesto desde el punto de vista de la estructura de datos y los algoritmos. Finalmente, en la sección 4 se presentan las conclusiones y el trabajo en progreso.

2 ÍNDICES RELACIONADOS

En esta sección se hace una presentación de los MAETs en los cuales se basa nuestra propuesta, con el objeto de conocer sus características principales y funcionamiento.

2.1 R-Tree

El R-Tree [1] es una extensión del B-Tree, que sirve para almacenar objetos multidimensionales, por ejemplo puntos y regiones en el espacio. Un R-tree organiza una colección de objetos espaciales en forma jerárquica donde las hojas contienen punteros a los datos en sí, y los nodos intermedios contienen el mínimo rectángulo envolvente (MBR: *Minimum Bounding Rectangle*) al objeto espacial que contiene a sus hijos.

Sea M el número máximo de entradas que puede almacenar un nodo y sea $m<=M/2$ el número mínimo de entradas para un nodo. Un R-Tree satisface las siguientes propiedades:

- Cada nodo contiene entre m y M entradas a menos que represente la raíz.
- Para cada entrada (I, *oid*) en un nodo hoja, I es el MBR que contiene al objeto.
- Todo nodo interno tiene entre m y M hijos, a menos que corresponda a la raíz.
- Para cada entrada de la forma <I, *pchild*> de un nodo interno, *pchild* es la dirección del correspondiente nodo hijo en el R-Tree, e I es el rectángulo más pequeño que cubre espacialmente a los rectángulos definidos en el nodo hijo.
- El nodo raíz tiene al menos dos hijos, a menos que sea una hoja.
- Todas las hojas se encuentran a un mismo nivel.

La altura de un R-Tree que almacena n claves es a lo mas $\lceil \log_m N \rceil - 1$, debido a que el número de hijos es como mínimo m. El máximo número de nodos es $\lceil N/m \rceil + \lceil N/m^2 \rceil + \ldots + 1$. En el peor caso la utilización del almacenamiento para todos los nodos con acepción de la raíz es m/M. Los nodos tienden a mantener más que M entradas, lo cual permite que la altura del árbol disminuya y por consiguiente mejore la utilización del almacenamiento.

Los algoritmos de búsqueda e inserción son muy similares a los empleados en un B-Tree para las mismas operaciones.

2.2 3D R-Tree

El 3D R-Tree [8] considera el tiempo como otra dimensión, además de las dimensiones espaciales. La idea principal es evitar la discriminación entre preguntas espaciales y temporales.

Cuando un objeto se mueve a otra posición o cambia de forma, se crea un nuevo MBR para representar el cambio del objeto, y dicho MBR conteniendo tanto la extensión espacial del objeto como así también su vida útil (tiempo), es insertado en el 3DRtree.

Una consulta espacio-temporal se modela como un cubo, donde la superficie de su base representa la región de consulta y su altura corresponde al intervalo temporal de la consulta. Usando el cubo especificado en la consulta, se busca en el 3D R-Tree todos los objetos que se interceptan con él utilizando el mismo procedimiento de búsqueda especificado en la propuesta original del R-Tree.

El 3D R-Tree soporta tanto consultas espaciales como así también temporales, aunque posee algunas desventajas en su desempeño.

Una de las desventajas principales es su ineficiencia en el procesamiento de consultas *timeslice* debido a que el 3D R-Tree puede ser muy grande en altura como consecuencia de la gran cantidad de objetos que es necesario almacenar.

Otra de sus desventajas es que se necesita conocer los intervalos de tiempo de manera anticipada, lo cual impide mezclar operaciones de actualización de la estructura con operaciones de consultas.

Las ventajas del 3D R-Tree son su buena utilización del almacenamiento y la eficiencia para el procesamiento de consultas de tipo *intervalo* debido a que simplemente es necesario recuperar del 3D R-Tree sólo los objetos que se interceptan con el cubo de la consulta.

2.3 2+3 R-Tree

El objetivo del 2+3 R-tree es indexar información actual y pasada de objetos que se encuentran en movimiento. La idea principal es tener dos R-tree's de manera separada. Un R-tree de dos dimensiones para almacenar los objetos que tienen *indefinido* el tiempo final de su estancia o permanencia en un lugar porque aún no se han movido de tal sitio. Y un segundo R-tree Tridimensional, de dos dimensiones espaciales y una dimensión temporal, para almacenar aquellos objetos que tienen *definidos* el tiempo inicial y final de permanencia en una determinada posición. Mientras el tiempo final de un objeto es desconocido, el objeto se almacena en el R-Tree bidimensional guardando su tiempo inicial su *id* y su posición. Una vez que el objeto cambia su posición, se construye un MBR tridimensional, el cual es insertado en el 3D R-tree y es eliminado del R-Tree bidimensional. Dependiendo del tiempo de la consulta, puede ser necesario hacer la búsqueda en ambos árboles.

Evaluaciones realizadas en [11] muestran que el método de acceso 2+3 R-Tree es uno de los más eficientes en almacenamiento, y de todos lo comparados es el que necesita menos espacio. Por otra parte, presenta un rendimiento muy competitivo para procesar consultas de tipo *intervalo*. Sin embargo, para consultas de tipo *timeslice* el método es ineficiente siendo superado por varios.

3 I+3 R-TREE

Tal como se mencionó anteriormente, el 2+3 R-Tree es una estructura que permite mantener datos de tipo espacio-temporal, el cual consta de dos R-Tree, uno bidimensional y otro tridimensional.

La estructura que proponemos, llamada I+3 R-tree, es una alternativa al 2+3 R-Tree, con la cual pretendemos superar algunas desventajas o carencias que posee la misma, por ejemplo el 2+3 R-Tree no resuelve consultas de eventos ni de trayectoria. También se debe considerar el hecho de que el uso de un R-tree de dos dimensiones para recuperar los hechos actuales, es decir los objetos en sus posiciones corrientes, fija un desempeño asociado a la estructura propiamente dicha, dejando de lado aspectos vinculados a los tipos de consultas que se desean resolver.

El I+3 R-Tree reemplaza el R-Tree bidimensional de la estructura original, por una estructura de datos que actúa como un índice de acceso a todos los objetos con sus respectivos últimos movimientos. De esta manera, la información actual se mantiene en el índice, mientras que al igual que en el 2+3 R-Tree la información histórica se mantiene en el 3D R-Tree.

El I+3 R-Tree está enfocado para apoyar aplicaciones en las cuales la cantidad de objetos en movimiento es fija y el espacio donde los objetos se mueven es acotado y conocido, y, además, hay una alta frecuencia de movimientos, por ejemplo: aplicaciones en el ámbito de transporte, medioambiente, social (demografía, salud, etc.) y multimedia. Se asume que los objetos son capaces de informar, en forma discreta las coordenadas y el tiempo en que el objeto alcanzó una nueva posición espacial.

La estructura permite procesar consultas de tipo *timeslice* e *intervalo* de la misma manera que en el 2+3 R-Tree, además permite resolver consultas de tipo *evento* y *trayectoria* lo cual se considera una mejora sobre la estructura de datos en la cual nos basamos.

Con respecto a las desventajas planteadas para el 3D R-Tree, en donde se requiere conocer los intervalos de tiempo de manera anticipada y se impide mezclar operaciones de actualización de la estructura con operaciones de consultas, en esta nueva propuesta queda salvada.

Por otro lado, ninguna de las propuestas anteriores resuelve consultas de trayectoria. Este punto sí es resuelto en el I+3 R-Tree, agregando una modificación a la estructura 3D R-tree. Dicha modificación consiste en mantener una lista para cada objeto enlazando en el 3D R-Tree los cubos correspondientes a las diferentes posiciones por las cuales ha pasado el objeto. El acceso a estas listas se realiza a través del *Índice* desde la posición actual de cada objeto mediante un direccionamiento directo, ya que la cantidad de objetos es fija.

En el 3*D R-Tree* se almacenan los *cubos definidos*, cada uno de los cuales representa la estadía de un objeto en una posición durante un intervalo definido de tiempo. Es decir, toda la información espacio – temporal histórica se mantiene en el 3D R-Tree. Las nuplas almacenadas en el 3D R-Tree son del tipo <*oid, mbr_3D, p_tray*> donde:

- *oid*: código identificador del objeto;
- *mbr_3D*: región tridimensional cuya altura representa el intervalo temporal durante el cual el objeto se mantuvo en la posición espacial definida por su base;
- *p_tray*: puntero al cubo anterior correspondiente al mismo *oid*, utilizado para mantener un historial de trayectoria.

En el Índice se almacenan los *cubos abiertos*. Esto es aquellos cubos para los cuales su techo no está definido; es decir su instante final en una determinada posición aún no está definido. También se guardan las referencias necesarias a los cubos anteriores que describen la trayectoria del objeto. El índice es una lista secuencial de *N* elementos, donde *N* es la cantidad de objetos considerados, que almacena nuplas de la forma <*oid, mbr, t, p3D, pa, ps*> donde:

- *oid*: código identificador del objeto;
- *mbr*: región aproximada que ocupa actualmente el objeto;
- *t*: tiempo de llegada del objeto a su ubicación actual;
- *p3D*: puntero al cubo anterior correspondiente al mismo oid, utilizado para mantener un historial de trayectoria;
- *pa*: puntero al objeto insertado en el instante de tiempo inmediatamente anterior; y
- *ps*: puntero al objeto insertado en el instante de tiempo siguiente.

La lista secuencial se encuentra ordenada por el *oid*; como la cantidad de objetos que se almacenan es fija, esta lista puede ser accedida como un direccionamiento directo, lo cual es una ventaja a la hora de realizar búsquedas por *oid* para resolver consultas de tipo *trayectoria*. Esta lista se mantiene en memoria principal.

Los punteros *pa* y *ps* son utilizados para enlazar los tiempos en orden creciente. Este ordenamiento por tiempo se realiza con el objetivo de poder acceder de manera menos costosa a un instante determinado en el *índice* para resolver consultas de tipo *timeslice*, *intervalo* o *evento*.

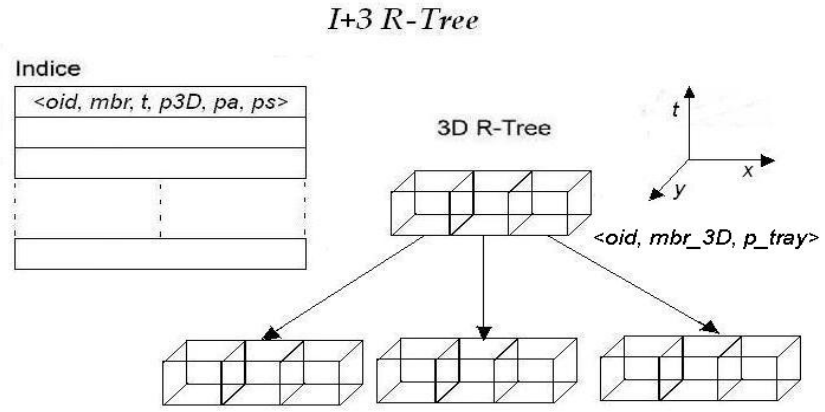

Fig. 1: I+3 R-Tree

4 LOS ALGORITMOS DE CONSULTAS EN EL I+3 R-TREE

A continuación se presentan los algoritmos para la resolución de consultas en el I+3 R-Tree. Se mantienen las variables *TMax3D* y *TMinInd* con el objetivo de mejorar la eficiencia en el procesamiento de las consultas evitando en algunos casos el acceso innecesario a alguna de las estructuras.

La variable *TMax3D* mantiene el tiempo mas reciente registrado en el 3D R-Tree y la variable TMinInd mantiene el tiempo mas antiguo registrado en el Índice.

4.1 Consulta TimeSlice

El algoritmo para el procesamiento de consultas timeslice hace uso de un algoritmo de búsqueda en el índice, y de un algoritmo de búsqueda en el 3D R-Tree y devuelve los resultados en un conjunto *Q* de objetos que conforman la respuesta.

El algoritmo *armar_region()* construye la región de consulta a partir de los puntos que representan los extremos opuestos de la misma y el tiempo de consulta. Para el caso de una consulta del tipo timeslice la región de consulta es plana, por lo tanto, la tercer coordenada que representa la dimensión temporal tendrá el mismo valor t_i para todo punto en la frontera de la región.

El algoritmo *BuscarEn3DR()* es simplemente el algoritmo de búsqueda del 3D R-Tree, que recibe como parámetro una región plana de consulta, y devuelve todos los objetos que se intersectan con esta región de consulta en el tiempo t_i.

Algoritmo *TimeSlice(p1, p2,, t_i)*
Entrada: *p1*, *p2* son los puntos utilizados para armar el rectángulo de consulta, t_i es el instante de tiempo sobre el cual se realiza la consulta.
Salida: *Q* es el conjunto de objetos que responden a la consulta.

```
1.  Si t_i > TMax3D entonces
2.      Q ← BuscarEnIndice(p1, p2, t_i)  // La respuesta está solo en el índice
3.  Sino
4.      region ← armar_region(p1, p2, t_i, t_i)
5.      Si t_i < TMinInd entonces
6.          Q ← BuscarEn3DR(region)
7.      Sino
8.          Q ← BuscarEnIndice(p1, p2, t_i)
9.          Q ← Q ∪ BuscarEn3DR(region)
10. Retornar Q
```

Algoritmo *BuscarEnIndice(*$p1$, $p2$, t_i*)*

Entrada: $p1$, $p2$, son los puntos utilizados para armar el rectángulo de consulta, t_i es el instante de tiempo sobre el cual se realiza la consulta.
Salida: Q es el conjunto de objetos que responden a la consulta.
Nupla es un objeto de la forma <*oid*, *pos*, *tiempo*, *Psig*, *Pant, Ptray*> donde *oid* es el identificador del objeto, *pos* es la posición del objeto, *tiempo* es el instante en el cual el objeto llegó a *pos*, *Psig* es el puntero a la siguiente nupla ordenada por tiempo, *Pant* es el puntero a la nupla anterior ordenada por tiempo y *Ptray* es puntero a la pos anterior del objeto para mantener la trayectoria.

1. *Nupla* ← *primero*
2. *fin* ← *false*
3. *region* ← *armar_region*($p1$, $p2$, t_i, t_i)
4. Mientras *Nupla*(*tiempo*) <= t_i and not *fin*
5. Si *Nupla*(*pos*) ∩ *region*
6. *Q* ← *Q* ∪ *Nupla*
7. Si *Nupla*(*Psig*) <> Null
8. *Nupla* ← *Nupla*(*Psig*)
9. Sino
10. *fin* ← *true*
11. Retornar *Q*

4.2 Consulta Intervalo

El algoritmo para el procesamiento de consultas de intervalo utiliza los mismos algoritmos de búsqueda que el algoritmo para procesar consultas timeslice. Se utiliza el mismo algoritmo *BuscarEnIndice()* debido a que todos los objetos contenidos en el índice se encuentran en su posición actual, por lo tanto el índice solo consta de cubos abiertos y por este motivo solo es necesario hacer la comparación con el piso de los cubos al igual que en consultas de tipo TimeSlice. La región construida por *armar_region()* es un cubo con un piso y un techo dados por los limites inferior (t_i) y superior (t_k) del intervalo de consulta respectivamente. El algoritmo *BuscarEn3DR()* recibe en este caso, a diferencia que para consultas timeslice, una región de consulta cúbica como parámetro, y devuelve todos los objetos que se intersecan con éste cubo de consulta.

Algoritmo *Intervalo(*$p1$, $p2$,, t_i, t_k*)*

Entrada: p1, *p2* son los puntos utilizados para armar el rectángulo de consulta, t_i es el límite inferior del intervalo de tiempo y t_k es el límite superior sobre el cual se realiza la consulta.
Salida: Q es el conjunto de objetos que responden a la consulta.

1. Si t_i > *TMax3D* entonces
2. *Q* ← *BuscarEnIndice(*$p1$, $p2$, t_i*)*
3. Sino
4. *region* ← *armar_region*($p1$, $p2$, t_i, t_k)
5. Si t_i < *TMinInd* entonces
6. *Q* ← *BuscarEn3DR(region)*
7. Sino
8. *Q* ← *BuscarEnIndice(*$p1$, $p2$, t_i*)*
9. *Q* ← *Q* ∪ *BuscarEn3DR(region)*
10. Retornar *Q*

4.3 Consulta Evento

El algoritmo para el procesamiento de consultas de evento utiliza los algoritmos *BuscarEnIndice_evento( )* y *BuscarEn3DR_evento( ).*

El algoritmo *BuscarEn3DR_evento( )* es similar al algoritmo de búsqueda en un 3D R-Tree, en este caso recibe como parámetro una región de consulta plana, devuelve como resultado solamente aquellos objetos cuyo piso o techo intercepten con la región de consulta.

Algoritmo *Evento(p*1, *p*2,, t_i*)*

*Entrada: p*1, *p*2, son los puntos utilizados para armar el rectángulo de consulta, t_i es el instante de tiempo sobre el cual se realiza la consulta.

Salida: Q es el conjunto de objetos que responden a la consulta.

1. Si t_i > *TMax3D* entonces
2. $Q \leftarrow$ *BuscarEnIndice_evento(p*1, *p*2, t_i*)*
3. Sino
4. *region* $\leftarrow$ *armar_region*(*p*1, *p*2, t_i, t_i)
5. Si t_i < *TMinInd* entonces
6. $Q \leftarrow$ *BuscarEn3DR_evento(region)*
7. Sino
8. $Q \leftarrow$ *BuscarEnIndice_evento(region,* t_i*)*
9. $Q \leftarrow Q \cup$ *BuscarEn3DR_evento(region)*
10. Retornar *Q*

Algoritmo *BuscarEnIndice_evento(p*1, *p*2, t_i*)*

*Entrada: p*1, *p*2, son los puntos utilizados para armar el rectángulo de consulta, t_i es el instante de tiempo sobre el cual se realiza la consulta.

Salida: Q es el conjunto de objetos que responden a la consulta.

Nupla es un objeto de la forma <*oid*, *pos*, *tiempo*, *Psig*, *Pant, Ptray*> donde *oid* es el identificador del objeto, *pos* es la posición del objeto, *tiempo* es el instante en el cual el objeto llegó a *pos*, *Psig* es el puntero a la siguiente nupla ordenada por tiempo, *Pant* es el puntero a la nupla anterior ordenada por tiempo y *Ptray* es puntero a la pos anterior del objeto para mantener la trayectoria.

1. *Nupla* $\leftarrow$ *primero*
2. *fin* $\leftarrow$ *false*
3. *region* $\leftarrow$ *armar_region*(*p*1, *p*2, t_i, t_i)
4. Mientras *Nupla*(*tiempo*) = t_i and not *fin*
5. Si *Nupla*(*pos*) $\cap$ *region*
6. $Q \leftarrow Q \cup$ *Nupla*
7. Si *Nupla*(*Psig*) <> Null
8. *Nupla* $\leftarrow$ *Nupla*(*Psig*)
9. Sino
10. *fin* $\leftarrow$ *true*
11. Retornar *Q*

4.4 Consulta Trayectoria

El algoritmo para el procesamiento de consultas de trayectoria, toma como entrada un intervalo de tiempo, y el identificador del objeto del cual se desea conocer su trayectoria durante ese intervalo de tiempo. Por medio de *BuscarEnIndice_tray()* se accede al índice para recuperar la posición actual del objeto de consulta, luego se recorre la lista que mantiene enlazadas las diferentes posiciones del objeto a lo largo del tiempo.

Algoritmo *Trayectoria(oid, t_i, t_k)*

Entrada: oid es el identificador del objeto a consultar, t_i es el límite inferior del intervalo de tiempo y t_k es el límite superior sobre el cual se realiza la consulta.

Salida: Q es el conjunto de posiciones que responden a la consulta.

Nupla es un objeto de la forma <*oid*, *pos*, *tiempo*, *Psig*, *Pant, Ptray*> donde *oid* es el identificador del objeto, *pos* es la posición del objeto, *tiempo* es el instante en el cual el objeto llegó a *pos*, *Psig* es el puntero a la siguiente nupla ordenada por tiempo, *Pant* es el puntero a la nupla anterior ordenada por tiempo y *Ptray* es puntero a la pos anterior del objeto para mantener la trayectoria.

1. *Nupla* ← *BuscarEnIndice_tray(oid)*
2. *fin* ← *false*
3. Si *Nupla*(*tiempo*)>= t_i and *Nupla*(*tiempo*) <= t_k
4. *Q* ← *Q* ∪ *Nupla*(*pos*)
5. Mientras *Nupla*(*Ptray*) <> Null and not *fin*
6. *Nupla* ← *Nupla*(*Ptray*)
7. Si *Nupla*(*tiempo*)>= t_i and *Nupla*(*tiempo*) <= t_k
8. *Q* ← *Q* ∪ *Nupla*(*pos*)
9. Sino
10. Si *Nupla*(*tiempo*) < t_i
11. *fin* ← *true*
12. Retornar *Q*

5 EVALUACIÓN EXPERIMENTAL

En este apartado presentamos algunos resultados de la etapa de evaluación experimental que se encuentra actualmente en progreso. Dicha evaluación experimental consiste en comparar nuestra estructura con respecto a la estructura original en la cual nos basamos [2]. Los criterios de comparación a tener en cuenta son la utilización de espacio en disco y el desempeño de los algoritmos de consulta utilizados.

Para las evaluaciones se utilizan lotes de prueba con 1000, 3000 y 5000 objetos, moviéndose en el espacio durante 50 instantes de tiempo consecutivos, con un porcentaje de movilidad de 1, 3, 5, 7, 9, 11, 13 y 15 por ciento por instante.

5.1 Comparación del uso de espacio en disco.

Se midió la cantidad de bloques utilizados en disco para ambas estructuras, los datos que presentamos en este punto corresponden a un archivo de datos de 3000 objetos con un porcentaje de movilidad de 1, 3, 5, 7, 9, 11, 13 y 15 por ciento.

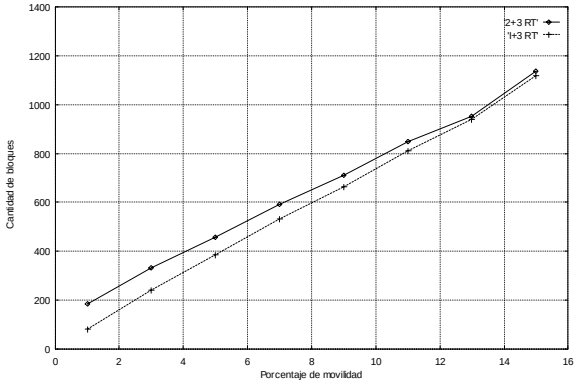

Fig.2: Numero de bloques utilizados para 3000 objetos

Como se puede observar en fig.2, la cantidad de bloques utilizados por las estructuras aumenta a medida que crece el porcentaje de movilidad. También se puede apreciar la diferencia en la cantidad de bloques utilizados por ambas estructuras.

El I+3 R-Tree presenta un menor costo que el 2+3 R-Tree, esto se debe a que nuestra implementación solo maneja en disco la estructura que almacena la información histórica mientras que el índice se mantiene en memoria principal. 2+3 R-Tree maneja ambas estructuras en disco lo cual implica el costo adicional.

5.2 Comparación del número de bloques leídos.

Para medir el desempeño de los algoritmos de consulta se calculó el promedio de bloques leídos tras realizar 100 consultas aleatorias. Los resultados que presentamos aquí corresponden a las consultas de tipo timeslice e intervalo, en ambos casos con áreas de consulta del 10% y para lotes de prueba de 3000 objetos con un porcentaje de movilidad de 1, 3, 5, 7, 9, 11, 13 y 15 por ciento.

5.2.1 Consulta timeslice

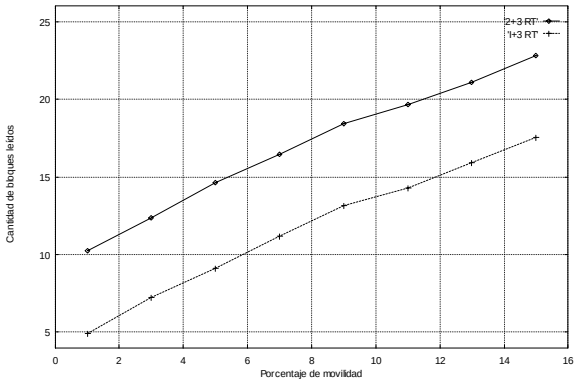

Fig.3: Numero de bloques leídos para Timeslice en un área de 10% para 3000 objetos.

En fig.3 podemos observar que la cantidad de bloques leídos se incrementa a medida que se incrementa el porcentaje de movilidad.

Ambas estructuras se comportan en este caso de manera similar, siendo un poco mas alto el costo de consulta en el 2+3 R-Tree debido a los accesos a disco extras necesarios para obtener las posiciones actuales de los objetos.

5.2.2 Consulta intervalo

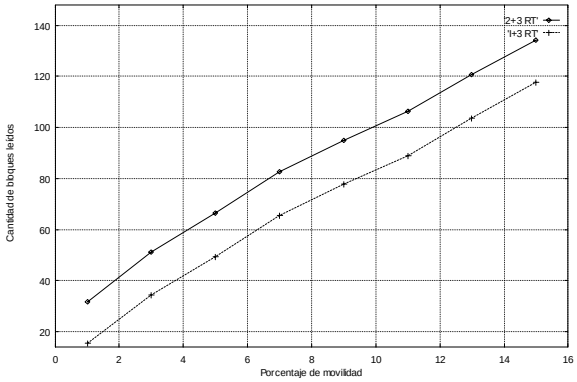

Fig.4: Número de bloques leídos para Intervalo de tamaño 5 en un área de 10% para 3000 objetos.

El comportamiento de las estructuras es en este caso similar al de la consulta de timeslice visto en el punto anterior.

El resto de los aspectos de experimentación están en etapa de desarrollo, faltando el análisis de las consultas de eventos y trayectoria. Sin embargo, consideramos que la implementación de la consulta de trayectoria es una mejora respecto a la estructura sobre la cual nos basamos ya que la misma no brinda soporte para este tipo de consulta.

6 CONCLUSIONES Y TRABAJO EN PROGRESO

Las bases de datos espacio-temporales comúnmente administran grandes volúmenes de datos almacenados a lo largo de un gran período de tiempo. Estos datos pueden ser consultados utilizando métodos de búsqueda exhaustiva, es decir, accediendo a todos los objetos de la base de datos y retornando aquellos que forman parte de la respuesta.

Tales consultas resultan ineficientes consumiendo gran cantidad de tiempo debido al tamaño que las bases de datos pueden alcanzar. Una solución más apropiada, consiste en construir índices sobre los datos de manera tal que las consultas puedan responderse accediendo solamente a una pequeña parte de la base de datos.

El costo de una búsqueda exhaustiva es despreciable si la frecuencia de consultas que utilizan ese método de búsqueda es baja, pero, en el caso de tener una alta frecuencia de consultas es conveniente el desarrollo de estructuras de indexación que permitan un acceso eficiente a la base de datos. En la actualidad este último caso es el más frecuente debido a la alta proliferación de aplicaciones que manipulan información de tipo espacio – temporal.

Este trabajo se enmarca en el área de estudio y desarrollo de nuevos métodos de indexación de información espacio-temporal. Esta es un área de investigación relativamente nueva, pero de importancia en la actualidad debido a la gran cantidad de avances en la tecnología móvil y a la creciente necesidad de uso plataformas que permitan la manipulación de datos espacio-temporales. Nuestro objetivo es brindar un aporte al desarrollo de Sistemas de Administración de Bases de Datos que provean soporte a tipos de datos espacio-temporales.

En la etapa que acaba de concluir, se implementó la estructura utilizando para ello el lenguaje de programación *Java* y el entorno de programación Eclipse.

Actualmente la investigación se encuentra en la etapa de experimentación, donde por los análisis teóricos de los algoritmos y la utilización del espacio, ya se presupone un buen desempeño de la estructura, esperando mostrar efectivamente la superación respecto a la estructura original.

Finalmente, concluimos que nuestra estructura resulta una contribución al área de investigación, ya que proponemos un nuevo método de acceso espacio-temporal que responda eficientemente los principales tipos de consulta: TimeSlice, Intervalo, Evento y Trayectoria, superando de esta manera a los métodos existentes en cuanto a la variedad de consultas que se pueden resolver eficientemente. Además, el trabajo realizado abre varias ramas interesantes para abordar en futuras investigaciones.

Como etapa de trabajo a futuro, se prevé el desarrollo de una interfaz grafica que refleje el desplazamiento de los objetos en una región, como así también permita visualizar el resultado de las consultas espacio-temporales sobre dichos objetos.

REFERENCIAS BIBLIOGRÁFICAS

[1] Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In ACM SIGMOD Conference on Management of Data, pages 47-57, Boston, 1984.

[2] M. A. Nascimento, J. R. O. Silva, and Y. Theodoridis. Evaluation of Access Structures for Discretely Moving Points. In *Proc. of the Intl. Workshop on Spatio-Temporal Database Management, STDBM*, Sept. 1999.

[3] Mario A. Nascimento, Jeferson R. O. Silva, and Yannis Theodoridis. Evaluation of access structures for discretely moving points. In Spatio-Temporal Database Management, pages 171-188, 1999.

[4] M. Nascimento, J. Silva, and Y. Theodoridis. Access structures for moving points, 1998.

[5] Dieter Pfoser, Christian S. Jensen, and Yannis Theodoridis. Novel approaches in query processing for moving object trajectories. In The VLDB Journal, pages 395-406, 2000.

[6] S. Saltenis, C. Jensen, S. Leutenegger, and M. López. Indexing the Positions of Continuously Moving Objects. ACM SIGMOD, 2000.

[7] Yufei Tao and Dimitris Papadias. E_cient historical r-tree. In IEEE International Conference on Scientific and Statical Database Management, 2001.

[8] Yannis Theodoridis, Michalis Vazirgiannis, and Timos K. Sellis. Spatio-temporal indexing for large multimedia applications. In International Conference on Multimedia Computing and Systems, pages 441-448, 1996.

[9] Theodoros Tzouramanis, Michael Vassilakopoulos, and Yannis Manolopoulos. Overlapping linear quadtrees and spatio-temporal query processing. The Computer Journal, 43(4):325-343, 2000.

[10] X Xu, J Han, and W Lu. Rt-tree: An improved r-tree index structure for spatio-temporal database. In 4th International Symposium on Spatial Data Handling, pages 1040-1049, 1990.

[11] Geraldo Zimbrao, Jano Moreira de Souza, Renata Chaomey Wo, and Victor Teixeira de Almeida. Eficient processing of spatiotemporal queries in temporal geographical information systems, 2000.