Optimización del procesamiento de lotes de consultas Espacio-temporales

Gilberto Gutiérrez[1] and Andrea Rodríguez[2]

[1] Universidad del Bío-Bío – Chillán / Chile
ggutierr@ubiobio.cl *
[2] Universidad de Concepción – Concepción / Chile
andrea@udec.cl

Abstract. Hasta ahora los esfuerzos en el procesamiento de consultas espacio-temporales (principalmente consultas del tipo *time-slice* y *time-interval*) se han concentrado en el diseño de estructuras de datos y algoritmos (método de acceso espacio-temporales) para procesarlas de manera eficiente teniendo en cuenta la ejecución aislada de cada consulta. Sin embargo, en ambientes altamente dinámicos y de alta demanda por procesamiento de información, se necesita evaluar eficientemente una gran cantidad o lotes de consultas espacio-temporales, generadas en intervalos de tiempo muy cortos. En este trabajo se presenta un primer algoritmo para evaluar lotes de consultas espacio-temporales. La idea detrás del algoritmo consiste en utilizar los objetos espacio-temporales, recuperados para evaluar una consulta, en la evaluación de aquellas que aún no han sido evaluadas. El algoritmo considera que el conjunto de objetos espacio-temporales se encuentra indexado por un MVR-tree. Resultados experimentales preliminares muestran que nuestro algoritmo produce ahorros entre un 5% y un 40% del total del tiempo requerido para procesar todas las consultas de manera independiente unas de otras.

1 Introducción

Las Bases de Datos Espacio-temporales están compuestas de objetos espaciales que cambian su posición y/o forma a lo largo del tiempo [12]. Su objetivo es modelar y representar la naturaleza dinámica de las aplicaciones del mundo real [7]. Algunos ejemplos de aplicaciones espacio-temporales se pueden encontrar en el ámbito del transporte, medioambiental y en aplicaciones multimedia. Los tipos de consultas espacio-temporales que han recibido mayor atención en la literatura son *time-slice* y *time-interval* [1, 10, 11, 9, 6, 4, 3, 12, 8]. Una consulta de tipo *time-slice* recupera todos los objetos que intersectan un rango espacial (window) en un instante de tiempo específico. Una consulta de tipo *time-interval* extiende la misma idea, pero a un intervalo de tiempo. Estas y otras consultas están basadas en las coordenadas de las posiciones que los objetos van alcanzando a lo largo del tiempo.

* Parcialmente financiado por el proyecto de investigación "Algoritmos para evaluación de consultas espacio-temporales", número 073218 4/R de la Universidad del Bío-Bío.

Con el objeto de procesar eficientemente las consultas de tipo time-slice y time-interval se han propuesto varios métodos de acceso espacio-temporales, destacando entre ellos RT-tree [15], 3D R-tree [14], HR-tree [7], MVR-tree/MV3R-tree [10], SEST-Index [3], entre otros. Todos los métodos de acceso mencionados anteriormente, se concentran en procesar de manera eficiente una sola consulta a la vez.

Sin embargo, existen varias situaciones prácticas donde es necesario procesar una gran cantidad o lote de consultas espacio-temporales. Algunos ejemplos son los siguientes:

- *Servicios implementados a través de la Web o cliente/servidor.* Considere un servicio donde se mantiene la información espacio-temporal de un conjunto de objetos a los cuales se necesita consultar por muchos usuarios. Este tipo de servicios exigirá del procesamiento de una gran cantidad de consultas generadas en lapsos muy breves de tiempo.
- *Procesamiento de consultas complejas.* Existen tipos de consultas espacio-temporales complejas, como las que se proponen en [5]. Este tipo de consultas corresponde a una secuencia de consultas time-slice/time-interval. Para procesarlas es posible dividirlas en consultas más simples y considerarlas como un lote de consultas a evaluar.
- *Reunión espacio-temporal.* En este caso el conjunto más pequeño se puede considerar como un lote de consultas espacio-temporales realizadas sobre el conjunto más grande.

Con el propósito de vislumbrar potenciales ahorros de tiempo de procesamiento se realizaron varios experimentos preliminares (ver Sección 4.1). Los resultados fueron muy promisorios, pues muestran un rango bastante amplio en donde es posible realizar optimizaciones por medio de estrategias de evaluación de las consultas espacio-temporales dentro de un lote. En este trabajo nos concentramos en la optimización del procesamiento de lotes de consultas espacio-temporales realizadas sobre un conjunto de objetos espacio-temporales los cuales se encuentran indexados mediante un MVR-tree. Nuestra propuesta reorganiza las consultas de manera diferente al orden en que fueron generadas y luego las procesa secuencialmente aprovechando de verificar la pertenencia de un objeto, recuperado al procesar una consulta particular, en todas las consultas que aún no han sido evaluadas. Nuestro enfoque difiere principalmente de aquellos que manejan bloques en memoria mediante la técnica de reemplazo LRU (Least Recently Used) para evaluar consultas, en que cada vez que se accede a un bloque, los objetos almacenados se procesan contra todas las consultas del lote pudiendo ser deshechado posteriormente y solamente se indica que tal bloque ya fue considerado.

El resto del artículo está organizado de la siguiente manera. En la sección 2 se describe el método de acceso MVR-tree, pues dicho método es utilizado para indexar el conjunto de objetos espacio-temporal. En la sección 3 se presenta y discute nuestro algoritmo como también las ideas subyacentes. En la sección 4 se muestran y discuten los resultados de una serie de experimentos realizados para demostrar la eficacia de nuestro algoritmo. Finalmente, en la sección 5,

presentamos las conclusiones de nuestro trabajo y delineamos futuras extensiones del mismo.

2 MVR-tree

El MVR-tree [10, 9] es un método de acceso espacio-temporal multiversión y corresponde a una extensión de MVB-tree [2], donde el atributo que cambia a través del tiempo es el espacial.

Similar al MVB-tree, cada entrada en el MVR-tree tiene la forma $\langle R, t_i, t_f, ref \rangle$, donde R corresponde a un MBR (Minimum Bounding Rectangle). Una entrada está *viva* en el instante t si $t_i \leq t \leq t_f$ y *muerta* en otro caso. MVR-tree mantiene ciertas restricciones respecto del número de entradas almacenadas en sus nodos. Una de estas restricciones asegura que en un instante t existan cero o al menos $b \cdot p_{version}$ entradas vivas en cualquier nodo interno, donde $p_{version}$ es un parámetro de la estructura y b es la capacidad de un nodo. Esta condición agrupa las entradas vivas en instante de tiempo con el propósito de evaluar eficientemente consultas de tipo time-slice. Otras restricciones (denominadas strong version overflow y strong version underflow) aseguran una buena utilización en los algoritmos de inserción y eliminación [10, 9].

Cada raíz abarca un intervalo temporal conocido como intervalo de jurisdicción y que corresponde al intervalo temporal mínimo que incluye a todos los intervalos temporales de las entradas en la raíz. Los intervalos de jurisdicción son mutuamente disjuntos. El procesamiento de una consulta de tipo time-slice o time-interval comienza recuperando aquellas raíces cuyos intervalos de jurisdicción se intersectan con el intervalo de la consulta. A continuación la búsqueda continúa guiada por R, t_i y t_f hasta alcanzar las hojas.

Similar al MVB-tree, un MVR-tree tiene múltiples R-trees (árboles lógicos) que organizan la información espacial en intervalos de tiempo que no se sobreponen.

Existe una variante del MVR-tree, llamada MV3R-tree [10] y tiene como propósito mejorar el procesamiento de las consultas de tipo time-interval cuando el largo del intervalo temporal es muy amplio (sobre un 5% del intervalo temporal almacenado en la base de datos) lo que logra agregando un 3D R-tree [14] el cual se construye considerando como objetos espacio-temporales los bloques hojas del MVR-tree.

3 Nuestra propuesta

En esta sección se propone un algoritmo para evaluar lotes de consultas espacio-temporales y que tiene como propósito minimizar el tiempo total de procesamiento de las consultas contenidas en el lote. Nuestro algoritmo supone que los objetos espacio-temporales se encuentran indexados por un MVR-tree, uno de las estructuras más eficientes conocidas para evaluar consultas de tipo time-slice y time-interval. También se supone que cada consulta del lote es de uno

de estos dos tipos. Nuestro algoritmo no considera los nodos (bloques) internos de los R-tree's del MVR-tree para conseguir ahorros, es decir, si a un bloque interno se accede en más de una oportunidad, el algoritmo no toma ventaja de esta situación, concentrándose solamente en los nodos o bloques hojas.

La idea básica del algoritmo es aprovechar el acceso a un bloque, provocado por la ejecución de una consulta del lote, para evaluar todas las restantes consultas que aún no han sido evaluadas completamente. El bloque (sólo su identificador) es mantenido en memoria con el propósito de indicar que ya fue considerado por consultas previamente procesadas. El algoritmo se muestra en Alg. 1, donde Q representa el lote de consultas espacio-temporales y cada consulta es de la forma (T, R) con T el intervalo temporal $[ti, tf]$ (ti el tiempo inicial y tf el tiempo final) y R el rango espacial. De la misma manera suponemos que los bloques del MVR-tree tienen un rango temporal y que cada una de sus entradas tiene asociado un rango temporal T y un rango espacial R. Adicionalmente, cada bloque hoja mantiene un atributo indicando la cantidad máxima de entradas ($MaxEntries$) y cada entrada almacena el identificador del objeto espacio-temporal (Oid). C representa un conjunto donde se registran separadamente las respuestas de cada consulta, es decir, $\forall q_i \in Q, i = 1, m$ con m la cantidad de consultas en Q, existe c_i con la respuesta de la consulta q_i (conjunto respuesta). Notar que los conjuntos respuestas pueden ser implementados de manera muy diversas. Si la aplicación corresponde a una que provee servicios a través de la Web, por ejemplo, las respuestas se pueden ir enviando conforme se van agregando objetos a los conjuntos. En otras aplicaciones se pueden considerar archivos independientes. Notar que el costo en operaciones de entrada/salida para almacenar las respuestas es independiente del orden en que estas se evaluan. El algoritmo también utiliza una tabla hashing H (organizada por $IdBloque$), de tamaño máximo tb, donde tb es la cantidad máxima de identificadores de bloques ($IdBloque$) que ya han sido accedidos por alguna de las consultas de Q. Finalmente, utilizamos un heap M (MINHEAP) donde se almacenan objetos del tipo $\langle ti, idBloque \rangle$ el cual se encuentra organizado por el atributo ti. Se utiliza la técnica de reemplazo LRU para decidir sobre $IdBloque$ que debe ser reemplazado (ver Alg. 3) cuando la tabla H se encuentra llena.

Al algoritmo aprovecha la propiedad del MVR-tree la cual consiste en que los intervalos temporales juridiccionales de sus raíces se encuentran ordenados por el tiempo (de menor a mayor). De esta forma, antes de proceder a evaluar las consultas de Q, estas se ordenan por el tiempo inicial ti (línea 4 del Alg. 1). Esta estrategia permite que si dos consultas que se encuentran cercanas o sus intervalos temporales se traslapan, es probable que los bloques hojas accedidos para evaluar una de ellas también sean útiles para evaluar la segunda.

Luego de ordenar las consultas del lote, el algoritmo procesa cada una de ellas (líneas 10 a 14 de Alg. 1) actualizando tanto el conjunto respuesta de la i–ésima consulta como los conjuntos respuestas de todas las consultas restantes para las cuales el bloque hoja recién recuperado contiene objetos que la satisfacen (Alg. 2). Notar que la verificación del predicado especificado en la línea 11 del Alg. 1 (si

Alg. 1 Algoritmo para evaluar lotes de consultas espacio-temporales

1: **EvaluaLotesConsultas(**Q**,** tb**)** {Q es el conjunto de consultas espacio-temporales y tb es la cantidad de identificadores de bloques de disco ($IdBloque$) que se mantienen en memoria.}
2: Sea m la cantidad de consultas en Q.
3: Sea C un conjunto tal que cada $c_i \in C$, representa la respuesta de la consulta $q_i, i = 1, \ldots, m$
4: Sort(Q) {Ordena las consultas de Q por el límite inferior (ti) del intervalo temporal}
5: Sea H una tabla hashing de tamaño tb organizada por $IdBloque$
6: Sea M un heap (MINHEAP) de tamaño tb organizado por ti
7: **for** i = 1 to m **do**
8: $c_i = \emptyset$
9: **end for**
10: **for** i = 1 to m **do**
11: **for** cada bloque hoja $n \mid n.IdBloque \notin H \wedge q_i.T \cap n.T \neq \emptyset \wedge q_i.R \cap n.R \neq \emptyset$ **do**
12: ActualizarRespuestas(Q, C, n, i, m)
13: ActualizarLRU(n, M, H)
14: **end for**
15: Enviar c_i {La respuesta de q_i}
16: Eliminar c_i.
17: **end for**

Alg. 2 Algoritmo para actualizar las respuestas de las consultas en Q.

1: **ActualizarRespuestas(**Q**,** C**,** n**,** i**,** m**)**
2: **for** j=i to m **do**
3: **for** k=i to n.MaxEntries **do**
4: **if** $q_j.T \cap n_k.T \neq \emptyset \wedge q_j.R \cap n_k.R \neq \emptyset$ **then**
5: $c_j = c_j \cup n_k.Oid$
6: **end if**
7: **end for**
8: **end for**

Alg. 3 Algoritmo para actualizar los bloques que ya han sido procesados.

1: **ActualizarLRU(**n**,** H**,** M**)**
2: **if** H está llena **then**
3: k = Min(M) {Obtiene la entrada en M con el mínimo valor de t_i}
4: **if** $k.ti \leq n.ti$ **then**
5: k = EliminarMin(M)
6: Eliminar desde H entrada con clave $k.idBloque$
7: Insertar $\langle n.ti, n.idBloque \rangle$ en M
8: Insertar $n.IdBloque$ en H
9: **end if**
10: **else**
11: Insertar $\langle n.ti, n.idBloque \rangle$ en M
12: Insertar $n.IdBloque$ en H
13: **end if**

un determinado bloque n es hoja y $n.IdBloque \notin H \wedge q_i.T \cap n.T \neq \emptyset \wedge q_i.R \cap n.R \neq \emptyset$) se realiza antes de acceder al nodo propiamente tal; concretamente se lleva a cabo en el bloque o nodo interno padre del bloque n en el recorrido desde la raíz hacia las hojas de los R-tree's seleccionados al evaluar la consulta.

4 Experimentación

Los experimentos consideraron lotes de tamaños entre 100 y 1.000 consultas espacio-temporales time-slice/time-interval. Las consultas se formaron considerando 4 posibles longitudes máximos del intervalo temporal a, saber 5,10, 15 y 20 y 4 rangos espaciales (re) formados por el 5%, 10%, 15% y 20% de cada dimensión espacial. De esta forma un lote formado por consultas con intervalo temporal de longitud 10 y rango espacial $re = 15$, significa que las consultas tienen un largo temporal entre 1 y 10 y rangos espaciales formados entre el 1% y 15% de cada dimensión. En ambos rangos se consideró distribución uniforme. En los experimentos se utilizó un conjunto de 23.268 puntos obtenidos mediante el generador de objetos espacio-temporales GSTD [13] con una movilidad de un 10%, con distribución inicial uniforme dentro del espacio $[0, 1) \times [0, 1)$ y un intervalo temporal de la base de datos de 200 instantes de tiempo. Además, se consideró que los objetos se encuentran indexados por medio de un MVR-tree. Adicionalmente consideramos 100, 500 y 1.000 como tamaños (tb) de la tabla hashing, medidos en cantidad de *idBloques* a mantener en memoria. Las cantidad de almacenamiento requerido para mantener la tabla hashing H y el heap M en memoria es pequeña. Por ejemplo, para $tb = 1.000$ el tamaño de H y de M es de 5 Kbytes y 10 Kbytes respectivamente.

4.1 Experimentos preliminares

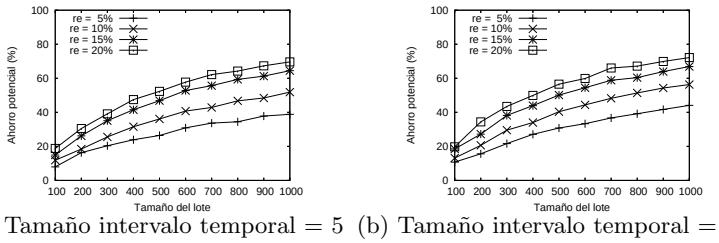
De manera preliminar se realizaron experimentos para detectar el ahorro potencial que se puede alcanzar al procesar lotes de consultas. Para obtener dicho ahorro se procesaron todas las consultas del lote y se obtuvo la cantidad de bloques hojas del MVR-tree accedidos en total (considerando bloques duplicados) y la cantidad de bloques accedidos considerando los bloques repetidos como un sólo acceso. Con estos valores se obtiene finalmente el ahorro potencial. Los resultados del experimento se encuentran en la Fig. 1, donde se puede observar que es posible conseguir ahorros importantes en el tiempo total de ejecución de las consultas de los lotes, los que van desde un 15% hasta un 80%. También es posible observar que en la medida que aumentan los tamaños, del intervalo temporal, del rango espacial (re) y del lote, el ahorro potencial aumenta lo que se explica por la mayor cantidad de veces que un bloque determinado se necesita acceder.

4.2 Rendimiento de nuestro algoritmo

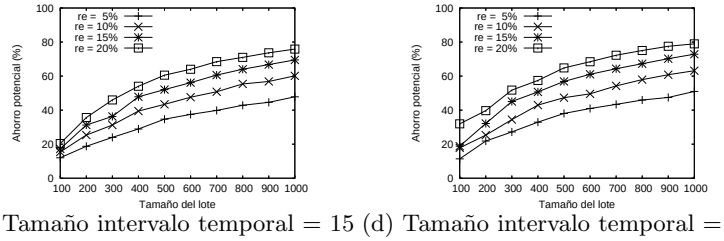
Con el objeto de medir la eficacia de nuestro algoritmo para procesar lotes de consultas espacio-temporales, se desarrollaron una serie de experimentos.

Los ahorros generados por el algoritmo se centran en las operaciones de entrada/salida (acceso a bloques de discos), pues consideramos que estas son las operaciones que predominan en el tiempo total requerido por la consulta por sobre las operaciones que realiza el algoritmo en memoria. De esta forma la unidad de medida utilizada en las evaluaciones de las consultas corresponden a accesos a bloques de disco.

Los resultados de los experimentos se resumen en las figuras 2, 3 y 4 las que muestran que es posible conseguir ahorros importantes. Por ejemplo, al observar las figuras 2-(a) y 3-(a) es posible observar que el ahorro real logrado por el algoritmo, para un de $tb = 500$, es muy cercano al potencial para lotes de tamaño en torno a las 200 consultas. También se puede observar que en la medida que el tamaño de tb aumenta, el ahorro real se aproxima al ahorro potencial, especialmente cuando tanto el tamaño del intervalo temporal y del rango espacial (re) de la consulta son pequeños (figuras 2-(a), 2-(b), 3-(a) y 4-(a)). También es posible apreciar, en las figuras 2, 3 y 4 el efecto del tamaño de la tabla hashing tb sobre el ahorro real, el cual aumenta conforme lo hace dicho parámetro lo que se explica por el simple hecho de que al tener más $IdBloques$ en la tabla hashing H, disminuye la posiblidad de recuperar un bloque requerido por una consulta. Sin embargo, en la medida que la amplitud máxima del intervalo temporal y del rango espacial aumentan, el ahorro producido por el algoritmo se empieza a alejar del ahorro potencial aún considerando valores altos para tb el cual se agudiza en la medida que aumenta el tamaño del lote.

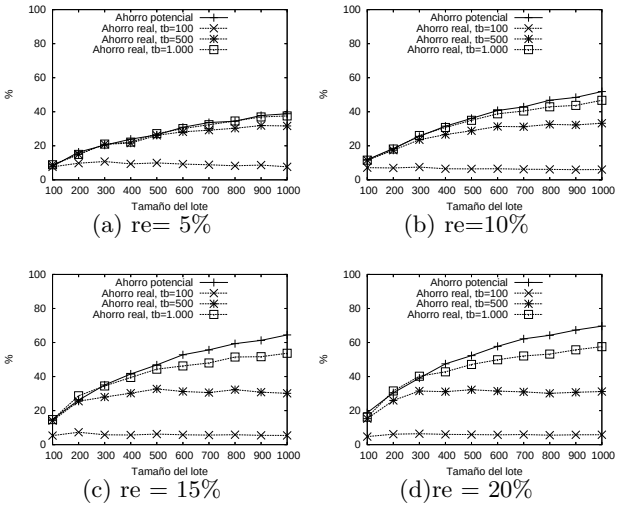


(a) Tamaño intervalo temporal = 5 (b) Tamaño intervalo temporal = 10



(c) Tamaño intervalo temporal = 15 (d) Tamaño intervalo temporal = 20

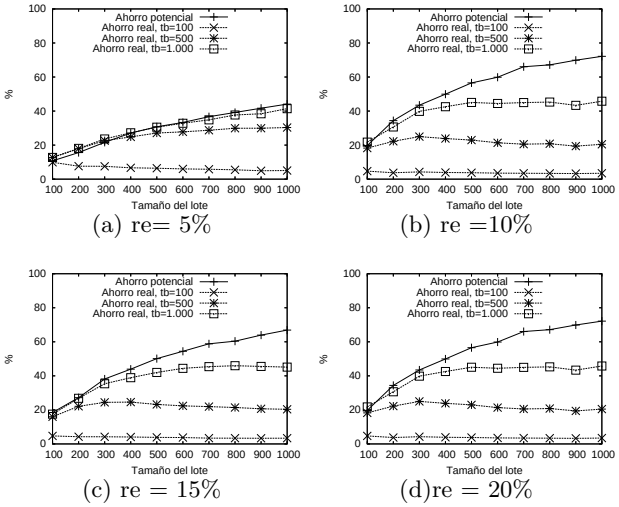
Fig. 1. Ahorro potencial considerando lotes de consultas espacio-temporales con diferentes rangos temporales y espaciales



(a) re= 5% (b) re=10%

(c) re = 15% (d)re = 20%

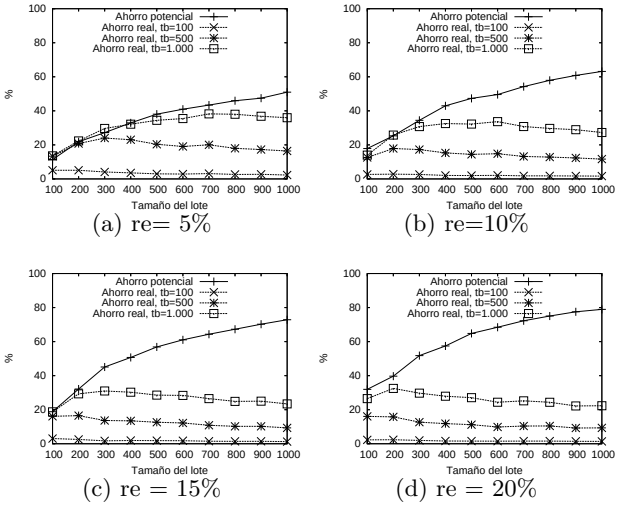
Fig. 2. Ahorro de accesos a bloques conseguidos por el algoritmo (Ahorro real) considerando un intervalo temporal de amplitud máxima 5 y diferentes rangos espaciales (re) máximos.



(a) re= 5% (b) re =10%

(c) re = 15% (d)re = 20%

Fig. 3. Ahorro de accesos a bloques conseguidos por el algoritmo (Ahorro real) considerando un intervalo temporal de amplitud máxima 10 y diferentes rangos espaciales (re) máximos.



(a) re= 5% (b) re=10%

(c) re = 15% (d) re = 20%

Fig. 4. Ahorro de accesos a bloques conseguidos por el algoritmo (Ahorro real) considerando un intervalo temporal de amplitud máxima 20 y diferentes rangos espaciales (re) máximos.

5 Conclusiones y Trabajo Futuro

En este trabajo presentamos un primer algoritmo para evaluar lotes de consultas espacio-temporales. Los resultados experimentales muestran que procesando las consultas bajo nuestra estrategia es posible lograr ahorros reales que van desde un 5% a un 40% respecto del costo de evaluar las consultas separadamente y de manera independiente. El ahorro logrado aumenta en la medida que también lo hace el el almacenamiento en memoria destinado para almacenar información de los bloques accedidos. Sin embargo, en la medida que tanto la amplitud del intervalo temporal, como el rango espacial de las consultas aumentan, la eficacia del algoritmo decae alejándose significativamente del ahorro potencial.

Como trabajo futuro, pretendemos integrar en nuestro algoritmo los atributos espaciales y temporales de los objetos, pues suponemos que es posible lograr un mayor ahorro en la medida que las consultas se ordenen temporal y espacialmente. En la misma dirección también nos interesa exlorar con estructuras de datos más complejas destinadas a organizar, en memoria, las consultas del lote de tal manera de tomar ventajas de sus propiedades. Finalmente, pretendemos complementar nuestro algoritmo mediante técnicas de paralelización, bien sea por el lado del MVR-tree, del lote de consultas o de ambas.

References

1. Pankaj K. Agarwal, Lars Arge, and Jeff Erickson. Indexing moving points. In *Symposium on Principles of Database Systems*, pages 175–186, 2000.
2. Bruno Becker, Stephan Gschwind, Thomas Ohler, Bernhard Seeger, and Peter Widmayer. An asymptotically optimal multiversion B-tree. *The VLDB Journal*, 5(4):264–275, 1996.
3. Gilberto Gutiérrez, Gonzalo Navarro, Andrea Rodríguez, Alejandro González, and José Orellana. A spatio-temporal access method based on snapshots and events. In *Proceedings of the 13th ACM International Symposium on Advances in Geographic Information Systems (GIS'05)*, pages 115–124. ACM Press, 2005.
4. Ralf Hartmut Güting and Markus Schneider. *Moving Objects Databases*. Morgan Kufmann, 1st edition, 2005.
5. Marios Hadjieleftheriou, George Kollios, Petko Bakalov, and Vassilis J. Tsotras. Complex spatio-temporal pattern queries. In *Proceedings of the 31st international conference on Very large data bases (VLDB '05)*, pages 877–888. VLDB Endowment, 2005.
6. Mohamed F. Mokbel, Thanaa M. Ghanem, and Walid G. Aref. Spatio-temporal access methods. *IEEE Data Engineering Bulletin*, 26(2):40–49, 2003.
7. Mario A. Nascimento, Jefferson R. O. Silva, and Yannis Theodoridis. Evaluation of access structures for discretely moving points. In *Proceedings of the International Workshop on Spatio-Temporal Database Management (STDBM '99)*, pages 171–188, London, UK, 1999. Springer-Verlag.
8. Dieter Pfoser and Nectaria Tryfona. Requirements, definitions, and notations for spatio-temporal application environments. In *Proceedings of the 6th ACM International Symposium on Advances in Geographic Information Systems (GIS'98)*, pages 124–130. ACM Press, 1998.
9. Yufei Tao and Dimitris Papadias. Efficient historical R-tree. In *SSDBM International Conference on Scientific and Statical Database Management*, pages 223–232, 2001.
10. Yufei Tao and Dimitris Papadias. MV3R-tree: A spatio-temporal access method for timestamp and interval queries. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 431–440, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
11. Yufei Tao, Dimitris Papadias, and Jun Zhang. Cost models for overlapping and multiversion structures. *ACM Trans. Database Syst.*, 27(3):299–342, 2002.
12. Yannis Theodoridis, Timos K. Sellis, Apostolos Papadopoulos, and Yannis Manolopoulos. Specifications for efficient indexing in spatiotemporal databases. In *IEEE Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, pages 123–132, 1998.
13. Yannis Theodoridis, Jefferson R. O. Silva, and Mario A. Nascimento. On the generation of spatiotemporal datasets. In *Proceedings of the 6th International Symposium on Advances in Spatial Databases (SSD '99)*, pages 147–164. Springer-Verlag, 1999.
14. Yannis Theodoridis, Michalis Vazirgiannis, and Timos K. Sellis. Spatio-temporal indexing for large multimedia applications. In *Proceedings of the 1996 International Conference on Multimedia Computing and Systems (ICMCS '96)*, pages 441–448, Washington, DC, USA, 1996. IEEE Computer Society.
15. 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.