Universidad de Concepción
Dirección de Postgrado
Facultad de Ingeniería - Programa de Magíster en Ciencias de la Computación

ESTRUCTURA DE DATOS EFICIENTE EN ESPACIO PARA MANEJO DE TRAYECTORIAS SOBRE REDES

Tesis para optar al grado de
MAGÍSTER EN CIENCIAS DE LA COMPUTACIÓN

POR
Rodrigo Alejandro Rivera Fierro
CONCEPCIÓN, CHILE
Mayo, 2018

Profesor guía: Andrea Rodríguez y Diego Seco
Departamento de Ingeniería Informática y Ciencias de la Computación
Facultad de Ingeniería
Universidad de Concepción

Resumen

En este trabajo se presenta una nueva estructura de datos para trayectorias sobre redes que, haciendo uso de una estructura de datos compacta especializada en el problema de *Interval-intersection*, obtiene mejores tiempos de consulta usando una menor cantidad de memoria que las estructuras encontradas en la literatura.

Índice general

Resumen iii

Índice de cuadros vi

Índice de figuras vii

Capítulo 1. Introducción 1

1.1. Objetivos 2

1.2. Hipótesis 3

Capítulo 2. Conceptos Previos y Trabajo Relacionado 4

Capítulo 3. Baseline y Resultados Previos 9

3.1. Modificación del FNR-tree con Interval-tree 9

3.1.1. Estructura 9

3.1.2. Algoritmos 11

3.2. Resultados 14

3.2.1. Descripción de los datos 15

3.2.2. Uso de espacio y tiempo de construcción 16

3.2.3. Tiempo de consulta 17

3.3. Discusión 22

Capítulo 4. Nuestra Propuesta 25

4.1. Estructuras de Datos para Interval Intersection 25

4.1.1. Schmidt 25

4.1.2. Estructura compacta con descomposición en IIS 27

4.2. Solución completa 29

4.2.1. Estructura 29

4.2.2. Algoritmos 29

Capítulo 5. Evaluación Experimental **33**
5.1. Evaluación de las estructuras de Interval-Intersection 33
5.1.1. Comportamiento según la naturaleza de los intervalos 33
5.1.2. Comportamiento según la precisión usada 35
5.1.3. Discusión Interval-Intersection 36
5.2. Comparación de la solución completa con FNR-tree y el baseline 37
5.2.1. Uso de espacio y tiempo de construcción 38
5.2.2. Tiempo de consulta 39

Capítulo 6. Conclusiones y Trabajo Futuro **44**

Bibliografía **45**

Índice de cuadros

Tabla 3.1. Uso de memoria por objeto, cuando hay 5.000 objetos circulando durante 100 unidades de tiempo. Fuente: Elaboración propia. . . 17

Tabla 5.1. Tiempos de consulta para datasets pequeños de intervalos. Fuente: Elaboración propia. 37

Tabla 5.2. Uso de memoria por objeto. Fuente: Elaboración propia. 39

Índice de figuras

Figura 2.1. (a) Conjunto de rectángulos indexados por un R-tree (b) El respectivo R-tree. Fuente: [10] 5

Figura 3.1. (a) Posibilidades de segmento en un MBB y su bandera de orientación. (b) La entrada correspondiente en la hoja del 2D R-tree. Fuente: [12] 10

Figura 3.2. Clasificación de los segmentos según x_{mid}. Fuente: [2] 10

Figura 3.3. Flag de dirección. Fuente: [12] 11

Figura 3.4. En verde los segmentos de red que intersecan la ventana de búsqueda. En morado el segmento de red x cuyo MBB interseca a la ventana de búsqueda, pero el segmento no lo hace, por lo que es descartado en el paso 3. Fuente: Elaboración propia. . . 14

Figura 3.5. Visualización de la red de Oldenburg en el generador de Brinkhoff. La red de San Francisco no pudo ser visualizada debido a su tamaño. Fuente: Elaboración propia. 15

Figura 3.6. Uso de memoria total de ambas estructuras para ambas ciudades. Fuente: Elaboración propia. 16

Figura 3.7. Tiempo de construcción para ambas estructuras. Fuente: Elaboración propia. 18

Figura 3.8. Tiempo de consulta: Grupo 1 - Oldenburg. Fuente: Elaboración propia. 19

Figura 3.9. Tiempo de consulta: Grupo 1 - San Francisco. Fuente: Elaboración propia. 20

Figura 3.10. Tiempo de consulta: Grupo 2 - Oldenburg. Fuente: Elaboración propia. 21

Figura 3.11. Tiempo de consulta: Grupo 2 - San Francisco. Fuente: Elaboración propia. 21

Figura 3.12. Tiempo de consulta: Grupo 3 - Oldenburg. Fuente: Elaboración propia. 22
Figura 3.13. Tiempo de consulta: Grupo 3 - San Francisco. Fuente: Elaboración propia. 22
Figura 3.14. Uso de Memoria en la estructura baseline, para las 2 redes utilizadas Oldenburg y San Francisco. Fuente: Elaboración propia. . 23
Figura 3.15. Distribución de tamaños de los Interval-trees, para las 2 redes utilizadas Oldenburg y San Francisco con 5.000 objetos circulando. Fuente: Elaboración propia. 24

Figura 4.1. Represetación de un grupo de intervalos usando la estructura de Schmidt. Fuente: [8] 26
Figura 4.2. Un conjunto de intervalos independientes (*IIS*) y su respectiva representación con 2 bitvectors. En rojo el último intervalo intersecado en `start` y el primero en `end`. Fuente: Elaboración propia. 28
Figura 4.3. Represetación de la estructura propuesta. Fuente: Elaboración propia 30

Figura 5.1. Intervalos de tamaño definido. Fuente: Elaboración propia. . . . 34
Figura 5.2. Intervalos aleatorios. Fuente: Elaboración propia. 34
Figura 5.3. Intervalos de trayectorias. Fuente: Elaboración propia. 35
Figura 5.4. Comportamiento según precisión usada. En el tercer gráfico, el último punto de la estructura compacta es omitido debido a un salto exponencial que lo hace demorar 40 veces más que el R-tree. Fuente: Elaboración propia. 36
Figura 5.5. Uso de memoria total de las 3 estructuras para ambas ciudades. Fuente: Elaboración propia. 38
Figura 5.6. Tiempo de construcción para las 3 estructuras (escala logarítmica). Fuente: Elaboración propia. 39
Figura 5.7. Tiempo de consulta: Grupo 1 - Oldenburg. Fuente: Elaboración propia. 40

Figura 5.8. Tiempo de consulta: Grupo 1 - San Francisco. Fuente: Elaboración propia. 41
Figura 5.9. Tiempo de consulta: Grupo 2 - Oldenburg. Fuente: Elaboración propia. 41
Figura 5.10. Tiempo de consulta: Grupo 2 - San Francisco. Fuente: Elaboración propia. 42
Figura 5.11. Tiempo de consulta: Grupo 3 - Oldenburg. Fuente: Elaboración propia. 42
Figura 5.12. Tiempo de consulta: Grupo 3 - San Francisco. Fuente: Elaboración propia. 43

Capítulo 1

Introducción

Con la masificación de las redes de sensores, redes inalámbricas y de dispositivos que cuentan con RFID, se ha masificado también la cantidad de datos disponibles sobre objetos en movimiento (también llamadas trayectorias). Estos datos cuentan con un potencial importante para ser usados en muchas aplicaciones reales de toma de decisiones. Por ejemplo, la optimización de sistemas de transporte, sistemas de navegación o planificación vial. Estas trayectorias pueden ser reconstruidas con, por ejemplo, los datos entregados por el GPS de los teléfonos inteligentes, o los entregados por tarjetas inteligentes (como las tarjetas Bip! en el Transantiago).

Las trayectorias pueden ser clasificadas en dos tipos: las trayectorias libres, donde el movimiento no tiene restricciones en el espacio, como es el caso de huracanes o animales; y las restringidas a redes, donde el movimiento se efectúa sobre redes y no puede existir fuera de éstas, como es la red del transporte público que tiene recorridos definidos.

En el transporte público en grandes ciudades los usuarios utilizan tarjetas inteligentes, como la tarjeta Bip! en Santiago, que permiten a un sistema de información recolectar la ubicación y el tiempo en el cual un usuario hizo ingreso al servicio. Con estos datos es posible reconstruir trayectorias que representen el movimiento de los usuarios sobre la red de transporte público. También es posible combinar la información del transporte público con la del movimiento de la gente extraído de la geolocalización de sus teléfonos celulares, permitiendo así análisis más complejos.

El manejo de estas trayectorias puede hacer posibles consultas como: contar el número de personas dentro de una región en un intervalo de tiempo determinado, contar el número de vehículos que entran o salen de una región u obtener el camino más rápido o económico entre 2 paraderos en un intervalo de tiempo determinado.

Producto de la rápida generación de este tipo de datos, su volumen está alcanzando un punto en que se está haciendo inmanejable el almacenamiento y procesamiento mediante estructuras de datos tradicionales.

Por otro lado, las estructuras de datos compactas están teniendo éxito en distintos dominios que sufren del mismo problema como son la Web (indexación de varios billones de páginas web) y bioinformática (indexación de secuencias genómicas cada vez más rápidas y baratas de obtener). Este tipo de estructuras utiliza una representación comprimida de los datos para reducir el espacio de almacenamiento, mejorar el rendimiento aprovechando los niveles de jerarquía de memoria para reducir el número de accesos a los niveles inferiores de memoria (los más lentos), y permitir la realización de consultas sin necesidad de descomprimir los datos. Además, el uso de estructuras de datos compactas, proporciona un enfoque complementario a otros que se han estado utilizando para el manejo de grandes volúmenes de datos [23]. Por ejemplo, en el caso de uso de memoria secundaria las estructuras de datos compactas permiten reducir la cantidad de accesos a discos teniendo una memoria principal (virtualmente) mayor, en el caso de algoritmos de streaming permiten almacenar estimaciones más precisas en el mismo espacio y en el caso de sistemas distribuidos permiten usar menos nodos (memoria conjunta virtualmente mayor). Es decir, permiten un ahorro en términos de hardware, comunicación y energía. Por lo que se estudiará su uso y utilidad en la representación de trayectorias en redes.

1.1. Objetivos

Objetivo general

Diseñar e implementar una nueva estructura de datos para el manejo de objetos en movimiento, que, incorporando estructuras de datos compactas especialistas en el problema temporal, *Interval Intersection*, mejore el rendimiento de las actuales estructuras.

Objetivos específicos

- Diseñar y evaluar experimentalmente una estructura de datos especializada en el problema temporal, *Interval Intersection*, para formar parte de la nueva estructura para trayectorias.
- Diseñar una nueva estructura de datos, y sus algoritmos, para trayectorias sobre redes de 2 niveles, que en su nivel temporal haga uso de la estructura resultante del objetivo anterior.
- Evaluar el rendimiento de la estructura tanto en términos de uso de espacio para su almacenamiento, como en la velocidad con la que responde las consultas en comparación a las estructuras implementadas en un trabajo previo.

1.2. Hipótesis

La inclusión de estructuras de datos compactas especializadas en el problema de *Interval Intersection* en una nueva estructura de datos para trayectorias sobre redes para resolver el problema temporal mejorará el rendimiento, no solo en uso de memoria, sino también en tiempo de consulta.

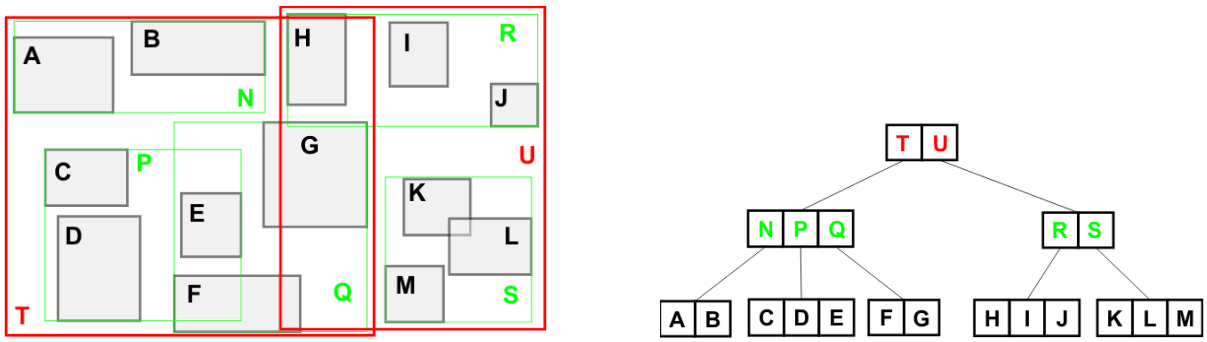
Capítulo 2

Conceptos Previos y Trabajo Relacionado

En primer lugar y, dado que muchas de las estructuras en el contexto de trayectorias utilizan o se inspiran en el clásico R-tree [15], éste es explicado brevemente.
El R-tree es un árbol balanceado de búsqueda, el que puede ser considerado la versión multidimensional del B-tree, utilizado para la indexación de información multidimensional, por lo general espacial en 2 o 3 dimensiones. La idea principal tras el R-tree es la agrupación de objetos cercanos para representarlos con su Minimum Bounding Box (para un espacio bidimensional, el rectángulo más pequeño que encierra a los objetos) en el siguiente nivel superior. Cada hoja del árbol representa un único objeto, mientras que, a medida que se sube de nivel, estos objetos se van agrupando hasta que en el nodo raíz todos los objetos son agrupados (ver Figura 2.1).

El R-tree permite hacer búsquedas de una forma simple y directa, aprovechando los MBBs (Minimum Bounding Boxes) para saber si continuar o no la búsqueda por ese nodo. Si la ventana de búsqueda no interseca con el MBB de un nodo, no lo hará con ninguno de los objetos dentro de ésta, es decir, sus hijos; por otro lado, si es que intersecan, se busca recursivamente entre sus hijos. Finalmente, si en una hoja, el MBB de un objeto en particular interseca con la ventana de búsqueda, éste es agregado al conjunto resultado.

Ya en el contexto de trayectorias, una estructura de datos para éstas debe proveer métodos de acceso que permitan el procesamiento de consultas espacio-temporales. Estas consultas pueden ser clasificadas como basadas en coordenadas y basadas en trayectorias [26]. Entre las consultas basadas en coordenadas se incluyen *time-slice* (determinar la posición de uno, o varios, objetos en un instante dado), *time-interval* (similar a lo anterior pero extendido para un rango de tiempo) y las consultas por vecinos cercanos. Ejemplos de éstas son “encontrar los objetos o trayectorias que estuvieron



(a) (b)

Figura 2.1: (a) Conjunto de rectángulos indexados por un R-tree (b) El respectivo R-tree. Fuente: [10]

presentes en cierta región en un instante o rango de tiempo dado" o "encontrar los k objetos más cercanos a un punto en un instante de tiempo dado".

En cuanto a las consultas basadas en trayectorias, éstas pueden ser clasificadas en topológicas, las que involucran información con respecto al movimiento del objeto, y las relativas a la navegación, que involucran información derivada del movimiento, como la velocidad o la dirección. Éstas también pueden ser combinadas para permitir la realización de consultas por objetos de los cuales hay información, como "dónde estaba el objeto X en un instante dado". Este trabajo se centra en las consultas basadas en coordenadas de tipo *time-slice* y *time-interval*.

Se han propuesto diversas estructuras de datos para soportar este tipo de consultas eficientemente sobre colecciones de trayectorias. Estas estructuras se pueden clasificar principalmente en 2 categorías:

- Las basadas en espacio libre, donde las trayectorias pueden estar en cualquier parte del espacio. Por ejemplo, 3D R-tree (una extensión tridimensional del R-tree [16]), TB-tree [26] (que preserva las trayectorias mientras permite consultas de rango típicas de R-tree) y MV3R-tree [32] (que utiliza un multi-version R-tree, llamado MVR-tree, junto a un 3D R-tree auxiliar).

- Las basadas en redes, donde las trayectorias están restringidas a una red y no pueden estar fuera de ésta, como por ejemplo FNR-tree [12] (que utiliza una combinación de 1D R-tree con 2D R-tree), MON-tree [9] (que utiliza 2 niveles de

2D R-tree) y PARINET [28] (basada en particionamiento de grafos y el uso de B$^+$-tree).

Entre las anteriores, FNR-tree y MON-tree tienen en común separar el problema espacio-temporal en dos dimensiones, la espacial (2 dimensiones) y la temporal (unidimensional), para poder atacar cada uno de estos sub-problemas por separado.

Para solucionar el problema espacial, es decir, la representación de la red en el espacio (plano bidimensional), ambas estructuras hacen uso de un 2D R-tree, almacenando los segmentos de la red como líneas. Con el problema espacial resuelto, lo que quedan son problemas temporales por cada segmento. Este problema es el de buscar todos los intervalos de tiempo (tiempo en el que algún objeto pasó por el segmento) que intersecan con un intervalo de consulta dado, este problema es conocido en la literatura como *interval intersection*, una extensión del más conocido *interval stabbing problem* [30]. Este problema ha sido muy estudiado y se pueden encontrar múltiples estructuras de datos que lo solucionan, como por ejemplo, Interval trees [2] y Priority trees [2].

En cuanto al subproblema en la dimensión temporal, FNR-tree hace uso de un R-tree unidimensional por cada segmento. Estos 1D R-tree indexan los objetos cuyas trayectorias pasan por su segmento de la red, almacenando el momento en que entra y sale del mismo en forma de intervalo temporal $(t_{entrada}, t_{salida})$. Puesto que solo se almacenan estos intervalos, se asume que los objetos no se detienen ni cambian de velocidad o dirección en medio de un segmento, sino que solamente pueden hacerlo en nodos. MON-tree elimina esta restricción reemplazando los R-trees unidimensionales por R-trees bidimensionales, donde almacenan el movimiento relativo dentro de los segmentos como rectángulos en el 2D R-tree de la forma (p_1, p_2, t_1, t_2), donde (p_1, p_2) es un intervalo de posición y (t_1, t_2) un intervalo temporal. Además, esta implementación incorpora otras técnicas para reducir tanto la complejidad espacial como temporal, como no agregar los segmentos de red al 2D R-tree hasta que efectivamente un objeto lo atraviese, manteniendo este nivel superior lo más pequeño posible, logrando así mejorar el rendimiento de FNR-tree.

Si bien algunas de las estructuras soportan consultas eficientemente sobre conjuntos de datos grandes, los volúmenes de datos actuales en algunos dominios están

alcanzando tal punto que estas estructuras no son capaces de manejar los datos. Esto ha forzado el uso de técnicas de compresión para almacenar y transferir estos datos. Se han propuesto técnicas, como por ejemplo, reducir el número de puntos en una curva para comprimir las trayectorias [22] o el aprovechamiento de características en cada punto como la velocidad y la orientación [27]. Ambas técnicas funcionan en espacios libres, pero cuando el movimiento está restringido a una red es posible conseguir una mejor compresión, como las mostradas en [17, 19, 20, 29].

Lo anterior utiliza compresión para mejorar los requerimientos de almacenamiento y tiempo de transmisión. Sin embargo, la compresión puede ser directamente aprovechada por estructuras de datos que pueden mantener una representación compacta de los datos mientras permite capacidades de búsqueda indexada. Estas estructuras han sido denominadas autoíndices y han sido implementadas exitosamente en otros dominios, como recuperación de información. Por ejemplo, hay autoíndices para lenguaje natural que almacena una representación implícita del texto utilizando menos de un 30 % del tamaño original, soportando una búsqueda directa sobre el texto comprimido [24] obteniendo mejores resultados cuando los documentos son muy repetitivos, el que puede ser el caso de las trayectorias.

Recientemente se han utilizado estructuras de datos compactas para la representación de trayectorias, como es el caso de GraCT [6] para trayectorias libres, que utiliza un k^2-tree [7] para almacenar la posición absoluta de los objetos en intervalos de tiempo regulares (snapshots) más logs comprimidos de movimiento relativo para la representación del movimiento entre snapshots; ContaCT [5], una versión mejorada de la anterior, con logs más eficientes; y de CTR [4], para trayectorias restringidas a redes, que combina arreglos de sufijos comprimidos (CSA) sobre los nodos de la red por los cuales se mueve el objeto para la representación de la componente espacial, y matrices de Wavelet balanceadas para la componente temporal.

CTR además se diferencia de las estructuras anteriormente mencionadas, en que éstas fueron diseñadas para responder consultas espacio-temporales donde el espacio y el tiempo son el principal filtro, como por ejemplo, “encontrar las trayectorias que pasaron por una región específica en un instante dado” o “encontrar las trayectorias que se intersecan”. Sin embargo, dichas estructuras no pueden resolver fácilmente

una consulta como “encontrar el número de trayectorias que iniciaron en X y terminaron en Y”, cosa que CTR logra resolver.

A diferencia de CTR nuestro enfoque consiste en desacoplar la red de las trayectorias y representarla mediante estructuras de datos compactas. Este modelo conocido como Network-Matched ha sido utilizado con éxito [20, 21, 11], pero sin emplear estructuras de datos compactas en su implementación.

Este enfoque tiene como ventaja que las trayectorias mapeadas a la red de transporte facilitan el encontrar similaridad y por ende también la repetición de movimientos, esto debido a que dos secuencias de puntos similares en el espacio probablemente sean mapeadas a las mismas aristas de la red. Lo que podría permitir un mejor nivel de compresión.

Capítulo 3

Baseline y Resultados Previos

3.1. Modificación del FNR-tree con Interval-tree

Para probar la factibilidad del uso de una estructura especializada en el problema de *Interval Intersection* en el problema general de trayectorias restringidas a redes se implementó una estructura de datos similar a las mencionadas FNR-tree y MON-tree, utilizando 2 niveles (espacial y temporal).

El primer nivel, el espacial, consta de un 2D R-tree que representa la red de transporte, donde cada una de sus hojas representa un segmento de la red, al que se asocia además un Interval-tree para indexar los intervalos de tiempo en los cuales cada objeto pasa por dicho segmento.

3.1.1. Estructura

Cabe recordar que el R-tree es un árbol balanceado donde cada hoja apunta a un objeto espacial. Las hojas son entradas de la forma (id, MBB), donde id es un identificador y MBB es el Minimum Bounding Box, un intervalo espacial que cubre al objeto.

Puesto que en el 2D R-tree los objetos espaciales a almacenar son segmentos rectos se aprovecha el MBB, para tener el segmento solo agregando una flag que diga la orientación (ver Figura 3.1). Cada una de estas hojas, o segmentos de red, además contiene un puntero a la raíz de un Interval-tree.

El Interval-tree [2] es, como su nombre lo indica, un árbol de búsqueda que almacena intervalos unidimensionales en cada uno de sus nodos y permite la búsqueda eficiente de los que intersequen con un intervalo de búsqueda dado, es decir, el problema de *interval intersection*.

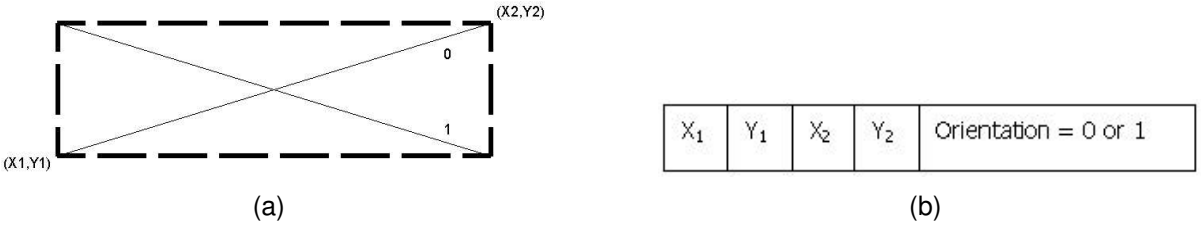


Figura 3.1: (a) Posibilidades de segmento en un MBB y su bandera de orientación. (b) La entrada correspondiente en la hoja del 2D R-tree. Fuente: [12]

El Interval-tree se construye de la siguiente manera:

- Se calcula un x_{med} igual a la mediana del conjunto conformado por los puntos extremos de todos los intervalos, tanto izquierdo como derecho.
- Se clasifican los intervalos en 3 conjuntos: I_{med}, I_{izq} y I_{der}. Quedando en I_{med} todos los intervalos que incluyen a x_{med}, en I_{izq} los que están totalmente a la izquierda de x_{med} y en I_{der} los que están completamente a la derecha de x_{med} (ver Figura 3.2).
- El conjunto I_{med} se almacena en alguna estructura y se asocia a la raíz. Recursivamente se construye otro árbol con los intervalos del conjunto I_{izq}, este se convertirá en el hijo izquierdo del nodo actual. Análogamente se realiza el mismo procedimiento para el hijo derecho con I_{der}.

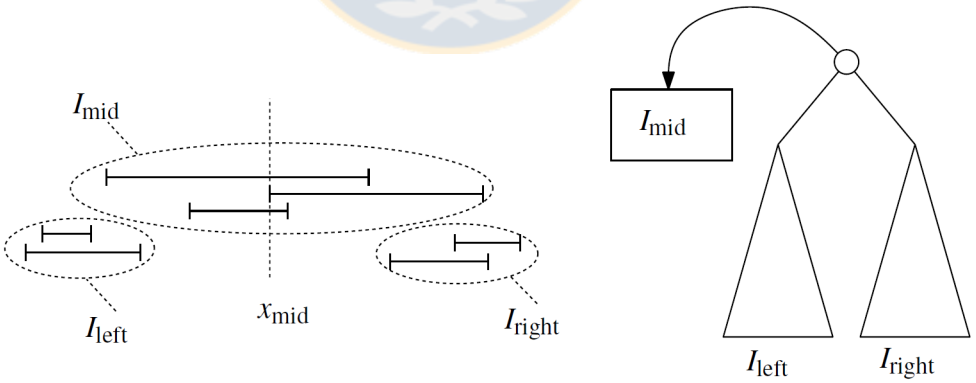


Figura 3.2: Clasificación de los segmentos según x_{mid}. Fuente: [2]

Además, para su uso en el contexto de trayectorias, se le asocia a cada uno de los intervalos el id del objeto, junto a un flag que indica la dirección en la cual se realizó

el movimiento. Éste toma el valor 0 cuando el movimiento se realiza de izquierda a derecha (o de oeste a este) y 1 en caso contrario, en el caso especial en que el segmento sea vertical, este flag tomará el valor 0 cuando el movimiento se realiza desde abajo hacia arriba (o de sur a norte) (ver Figura 3.3).

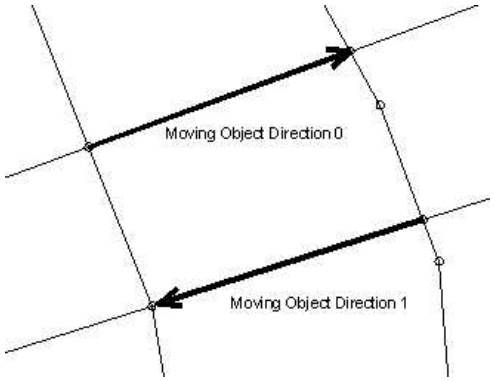


Figura 3.3: Flag de dirección. Fuente: [12]

3.1.2. Algoritmos

Construcción

La estructura diseñada es estática, es decir, no acepta modificaciones ni de la red ni de las trayectorias una vez es construido, esto debido a que la implementación de Interval-tree utilizada lo es. Por esto se utiliza un vector auxiliar temporal en la hoja espacial de cada segmento que almacenará los intervalos antes de la construcción del Interval-tree, donde este vector será eliminado.

Lo primero que necesita la estructura es la red. Estas redes son ingresadas a la estructura mediante un archivo de nodos que contiene tuplas de la forma $(id\_nodo, coord\_x, coord\_y)$ y un archivo de arcos que contiene tuplas de la forma $(id\_arco, id\_nodo\_origen, id\_nodo\_destino)$. Puesto que se necesitan los segmentos de la forma $(id\_seg, X_{origen}, Y_{origen}, X_{destino}, Y_{destino})$ los datos del primer archivo son indexados temporalmente en un `map` de manera de obtener eficientemente las coordenadas de un nodo según su id. Luego, a medida que se leen las tuplas del segundo archivo se consultan las coordenadas de los nodos en el map para hacer directamente la inserción de segmento de red en el 2D R-tree.

Terminado este proceso la red está completamente representada dentro de la estructura, y el map auxiliar puede ser eliminado. Lo siguiente es insertar las trayectorias, para luego finalmente construir la estructura.

La inserción de un segmento de trayectoria se realiza cada vez que un objeto abandona un segmento, en otras palabras, alcanza un nodo, siendo los argumentos para la inserción los siguientes:

- El identificador del objeto id,
- El segmento S, de la forma $(X_{inicio}, Y_{inicio}, X_{fin}, Y_{fin})$, y
- El intervalo de tiempo I durante el que estuvo en dicho segmento $(T_{entrada}, T_{salida})$.

Puesto que los datos son entregados en orden temporal, es necesario almacenar la última posición conocida de cada objeto, de donde se obtienen los argumentos X_{inicio}, Y_{inicio} y $T_{entrada}$. Para esto se emplea un `map` auxiliar, el que puede ser eliminado tras la inserción del último segmento de trayectoria.

`Inserción de un segmento de viaje`

```
1.- Buscar el segmento S en el 2D R-tree.

2.- Obtener la dirección del movimiento dir.

3.- Insertar con el intervalo I el id junto con dir en el
vector auxiliar en la hoja espacial del segmento S.
```

Notar que 2 segmentos diferentes no pueden tener el mismo MBB, por lo que se asume que la orientación es correcta al buscar en el 2D R-tree.

Una vez todos los segmentos de trayectorias están insertos en los respectivos segmentos de red, se procede a construir los Interval-trees dentro de cada uno de estos. Para esto simplemente se itera sobre todos los segmentos de red y se construye el Interval-tree con los intervalos insertados en el vector auxiliar, eliminando luego este último. Tras esto la estructura ya está habilitada para responder consultas.

Búsqueda

Las consultas que se quieren responder corresponden a consultas de rango y son de la forma “encontrar todos los objetos que pasan por cierta área en un cierto intervalo de tiempo”, y permite también hacer consultas como “encontrar todos los objetos que alguna vez pasaron por cierta área” utilizando el 100 % del tiempo como rango. Estas consultas de rango pueden ser fácilmente extendidas para responder consultas como “determinar la posición de todos los objetos en un instante dado”, o “determinar en qué momento cierto objeto pasó por una área dada”.

Estas búsquedas de rango consisten en encontrar todos los objetos cuya trayectoria interseque a una ventana de consulta tridimensional (las 2 espaciales más la temporal), es decir, los argumentos para ésta son: X_{min}, Y_{min}, X_{max}, Y_{max}, T_{min} y T_{max}, y ésta devuelve un conjunto de identificadores de los objetos encontrados.

Búsqueda

```
1- Buscar el conjunto de segmentos S cuyos MBB intersequen
con la ventana de búsqueda (X_min,Y_min,X_max,Y_max).

2- Buscar en el Interval-tree de cada uno de los segmentos
en S los intervalos que intersequen con el intervalo
temporal (T_min, T_max).

3- Por cada segmento en S donde la consulta sobre el
Interval-tree tuvo resultado, verificar si el segmento
interseca con la ventana de búsqueda (X_min, Y_min, X_max,
Y_max), de ser así, agregar los objetos encontrados en el
paso 2 al conjunto resultado (donde no hay duplicados),
de lo contrario descartarlos.
```

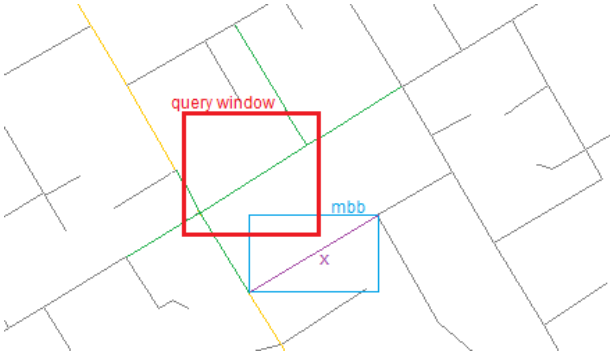


Figura 3.4: En verde los segmentos de red que intersecan la ventana de búsqueda. En morado el segmento de red x cuyo MBB interseca a la ventana de búsqueda, pero el segmento no lo hace, por lo que es descartado en el paso 3. Fuente: Elaboración propia.

3.2. Resultados

Para estudiar el rendimiento de la estructura implementada de manera experimental también se implementó un FNR-tree, y fueron comparados replicando los experimentos en la publicación de este último [12]. En la implementación de ambas estructuras se utilizaron implementaciones disponibles de R-tree [1] y de Interval-tree [13]. FNR-tree fue elegido debido a que ambas estructuras están diseñadas para responder las mismas consultas y ambas tienen las mismas restricciones, además, la estructura presentada en este trabajo nace como una modificación del FNR-tree.

Ambas estructuras fueron implementadas con el lenguaje de programación C++ (11), compiladas con G++ 5.4.0 y probadas en un computador con las siguientes especificaciones:

- Sistema operativo Ubuntu 16.04
- Procesador Intel Xeon E3-1220 v5 de 3.00 GHz
- Memoria de 62 GB

3.2.1. Descripción de los datos

Las trayectorias utilizadas fueron generadas utilizando el generador de trayectorias sobre redes de Thomas Brinkhoff [3] sobre las redes de caminos reales de las ciudades de Oldenburg y San Francisco (ver Figura 3.5). La red de Oldenburg cuenta con 6.105 nodos y 7.305 arcos, mientras que la red de San Francisco con 175.343 nodos y 223.606 arcos.

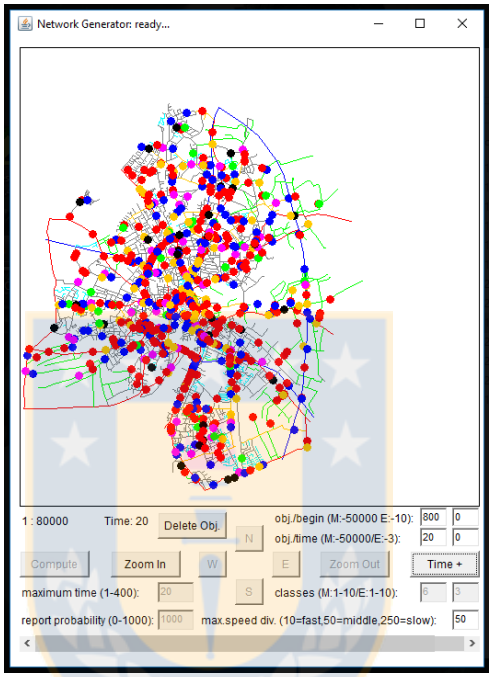


Figura 3.5: Visualización de la red de Oldenburg en el generador de Brinkhoff. La red de San Francisco no pudo ser visualizada debido a su tamaño. Fuente: Elaboración propia.

Para ambas redes se crearon trayectorias para 1.000, 2.000, 3.000, 4.000 y 5.000 objetos durante 100 unidades de tiempo. Para mantener una cantidad de objetos prácticamente constante de objetos durante el tiempo, ya que al llegar a su destino estos objetos desaparecen, se agregó un número de objetos equivalente al 2,5 % del número inicial de estos en cada unidad de tiempo, es decir, para el primer caso donde inicialmente hay 1.000 objetos, 25 objetos fueron incorporados en cada instante de tiempo.

Se configuró el generador para que éste informara la posición de cada objeto cada

vez que estos alcanzaban un nodo, produciendo así un archivo de salida con entradas de la forma (id, t, x, y), donde (x, y) es la posición del nodo en el que se encuentra el objeto id en el instante t. Este archivo también incluye otros datos que no son utilizados en este trabajo, tales como el tipo de objeto o la velocidad del mismo, entre otros.

3.2.2. Uso de espacio y tiempo de construcción

Primero se analiza el uso de espacio total, esto es, considerando tanto la red como las trayectorias. Como se puede ver en los gráficos de la Figura 3.6, cuando la cantidad de objetos moviéndose por la red tiende a 0, la estructura diseñada en este trabajo supera en uso de memoria a FNR-tree, esto dice que la cantidad de memoria base (la necesitada antes de hacer cualquier inserción) es mayor en la estructura diseñada. Esta diferencia se ve acrecentada en la red de San Francisco debido a la cantidad de segmentos que ésta contiene (aproximadamente 31 veces la cantidad de segmentos de Oldenburg), lo que significa una mayor cantidad de hojas en el 2D R-tree, las que almacenan el 1D R-tree del FNR-tree o el Interval-tree de la estructura diseñada. En este experimento sólo se consideraron las estructuras de la parte temporal que tenían 5, o más, intervalos.

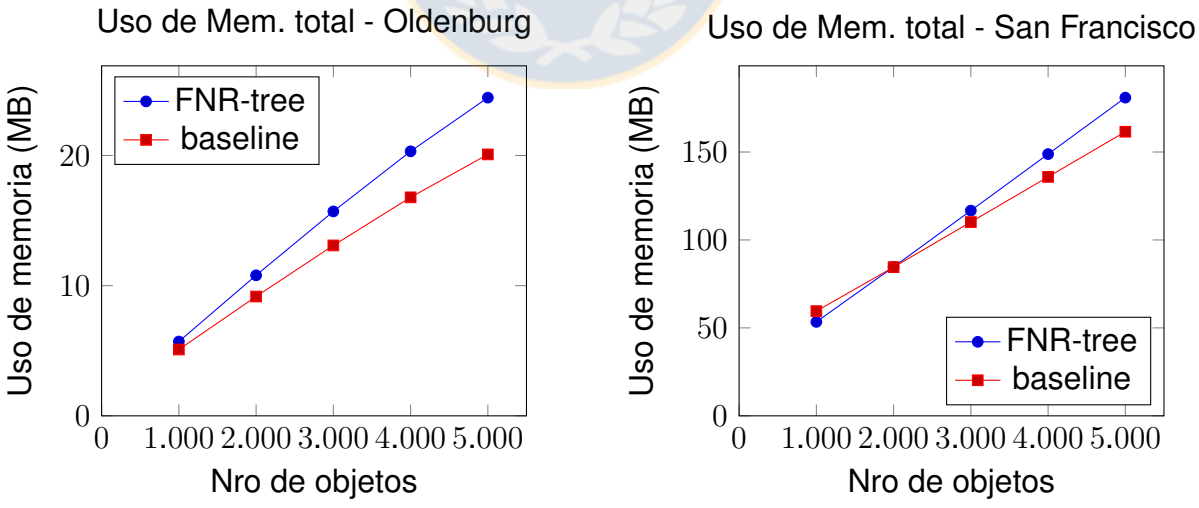


Figura 3.6: Uso de memoria total de ambas estructuras para ambas ciudades. Fuente: Elaboración propia.

A medida que el número de objetos moviéndose por la red se incrementa, la estructura diseñada empieza a tomar ventaja. Esto dice que la cantidad de memoria utilizada por los 1D R-tree para almacenar los movimientos es mayor a la utilizada por los Interval-tree. La cantidad de memoria utilizada por cada objeto para las 100 unidades de tiempos se muestra en la Tabla 3.1. La estructura diseñada requiere de aproximadamente un 20 % menos memoria que el FNR-tree por cada objeto. La cantidad de memoria requerida por objeto en San Francisco es mayor debido a la densidad de esta red, donde cada objeto pasa por una mayor cantidad de segmentos en la misma cantidad de tiempo. Para el dataset más grande, “5.000 objetos por San Francisco” de 299 MB, FNR alcanzó un uso de memoria de 181 MB, mientras que el baseline un uso de 162 MB.

Uso de memoria por objeto		
Estructura	Oldenburg	San Francisco
FNR-tree	∼5 KB	∼32 KB
baseline	∼4 KB	∼26 KB

Tabla 3.1: Uso de memoria por objeto, cuando hay 5.000 objetos circulando durante 100 unidades de tiempo. Fuente: Elaboración propia.

En cuanto al tiempo de construcción se puede apreciar que la estructura diseñada supera en todas las pruebas al FNR-tree en tiempo de construcción (ver Figura 3.7). Cabe señalar que en el proceso de construcción los 1D R-tree de FNR-tree eran actualizados en cada inserción, en cambio los Interval-trees de la estructura diseñada son estáticos, por lo que eran sólo son construidos una vez todas las trayectorias fueron insertadas. Por lo que se puede atribuir estos resultados a la complejidad de la inserción en el R-tree.

3.2.3. Tiempo de consulta

Como ya se ha mencionado, las consultas que se pueden realizar sobre la estructura son de rango, de la forma “encontrar todos los objetos que pasan por cierta área en un cierto intervalo de tiempo” y son expresadas como ventanas de consulta tridimensionales.

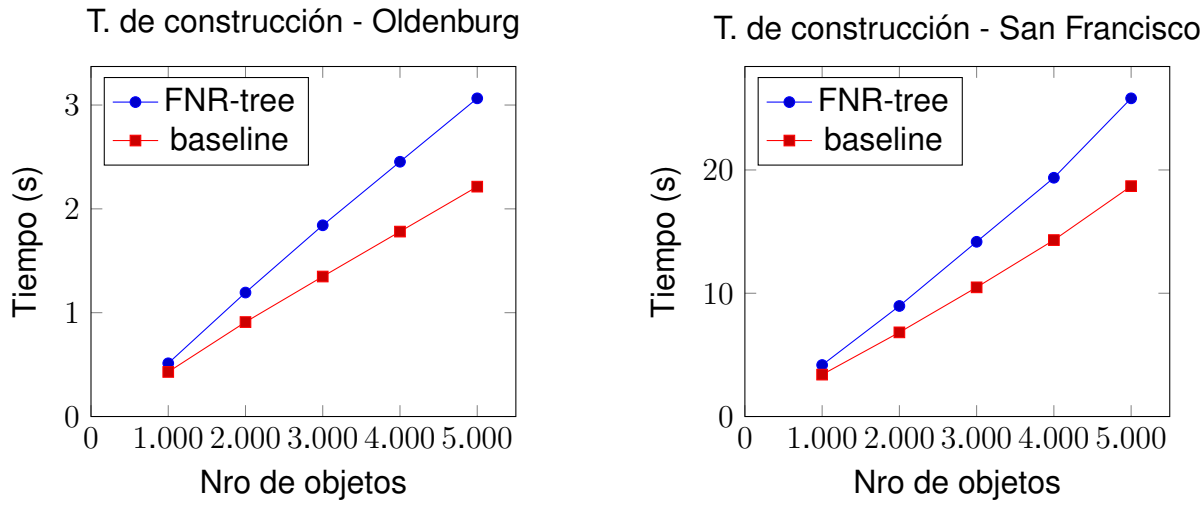


Figura 3.7: Tiempo de construcción para ambas estructuras. Fuente: Elaboración propia.

Para el estudio de rendimiento de éstas se distinguieron 3 tipos de consultas: las "normales" de la forma "encontrar todos los objetos que pasan por cierta área en un cierto intervalo de tiempo", las que usan la totalidad del espacio temporal de la forma "encontrar todos los objetos que alguna vez pasaron por cierta área" y las que utilizan una única unidad del espacio temporal de la forma "determinar todos los objetos que se encontraban en cierta área en un instante dado".

Para cada uno de estos grupos se generaron 3 conjuntos de 500 consultas de la siguiente forma:

- En el primer grupo los conjuntos utilizan 1 %, 10 % y 20 %, respectivamente, de cada una de las 3 dimensiones.
- En el segundo grupo, los primeros 2 conjuntos utilizan un 1 % de ambas dimensiones espaciales con un 10 % y 100 % de la dimensión temporal respectivamente, mientras que el tercer conjunto utiliza un 10 % de cada dimensión espacial con un 100 % de la dimensión temporal.
- En el tercer grupo los conjuntos utilizan un 1 %, 10 % y 100 % de ambas dimensiones espaciales respectivamente con un único punto de la dimensión temporal, estas consultas en particular son conocidas como "timeslice queries".

Estos 9 conjuntos de 500 consultas aleatorias fueron generados para ambas redes, Oldenburg y San Francisco.

Grupo 1: Consultas de rango con igual uso de cada dimensión.

En las Figuras 3.8 y 3.9 se muestran los resultados de tiempo de ejecución para las consultas del grupo 1, para las redes de Oldenburg y San Francisco, respectivamente. Se puede notar que para la red de Oldenburg el FNR-tree supera a la estructura diseñada en todas las pruebas, para cualquier cantidad de objetos. Sin embargo, es importante señalar que gran parte de las consultas retornan conjuntos vacíos de objetos o de poco tamaño. En cambio, para la red de San Francisco las consultas retornan una gran cantidad de objetos, debido a que la densidad de ésta hace que los objetos atraviesen una mayor cantidad de segmentos de la red, y que las ventanas de consulta cubren una mayor cantidad de estos segmentos. Es en estos casos donde la estructura diseñada como baseline supera al FNR-tree.

Como se puede ver, a medida que el tamaño de la ventana de consulta tridimensional crece el baseline diseñado logra tomar ventaja. Lo mismo sucede para el número de objetos.

En las Figuras 3.8-3.13 los gráficos tienen títulos de la forma x_y_z, lo que significa que las ventanas de consulta toman un $x\,\%$ y un $y\,\%$ de las dimensiones espaciales y un $z\,\%$ de la dimenión temporal.

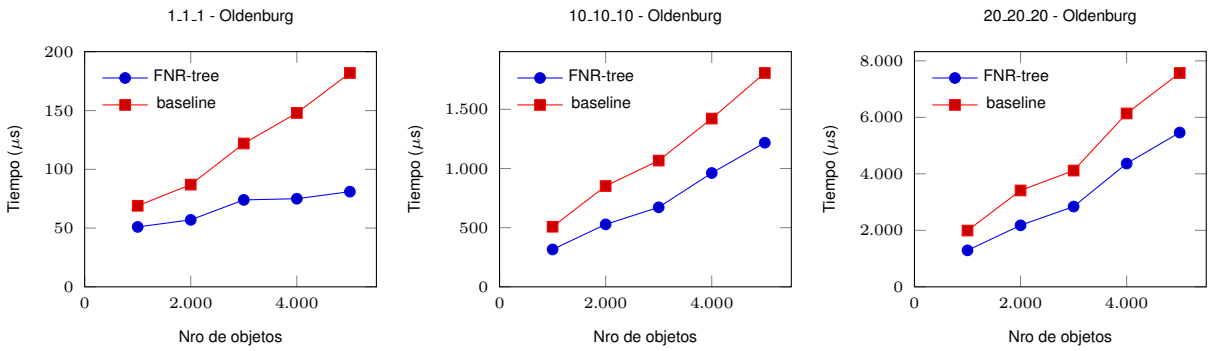


Figura 3.8: Tiempo de consulta: Grupo 1 - Oldenburg. Fuente: Elaboración propia.

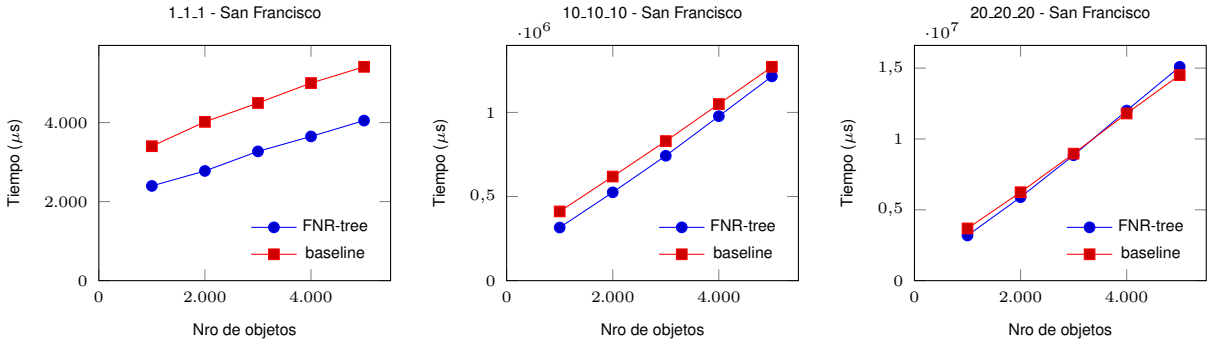


Figura 3.9: Tiempo de consulta: Grupo 1 - San Francisco. Fuente: Elaboración propia.

Grupo 2: Consultas de rango con gran uso de dimensión temporal.

En las Figuras 3.10 y 3.11 se puede ver el resultado del tiempo de consultas para el grupo 2 de consultas, que se caracterizan por utilizar una gran parte de la dimensión temporal (los 2 últimos usando la totalidad de ésta). Se puede ver que los resultados son similares a los del grupo anterior, siendo el FNR-tree superior en todas las pruebas realizadas sobre la red de Oldenburg, solo siendo superada por la estructura diseñada en las últimas pruebas con el tercer grupo (10 % de cada dimensión del espacio con la totalidad de la dimensión temporal) sobre la red de San Francisco.

Llama la atención que en los resultados sobre los 2 primeros grupos de consultas, donde el uso de dimensión espacial es idéntico, pero el de la dimensión temporal cambia drásticamente (de un 10 % a un 100 %) los resultados sean similares. Esto haría pensar que la variación en el uso de la dimensión temporal tiene un menor efecto en el rendimiento que la variación del uso de las dimensiones espaciales. Lo cierto es que para ambos casos la mayoría de las consultas tienen respuestas vacías debido a la estrecha ventana espacial utilizada de 1 % por dimensión, lo que significa solo un 0,01 % del espacio, por lo que la similitud se atribuye a esto.

Grupo 3: Timeslice queries.

Las Figuras 3.12 y 3.13 muestran los resultados del tiempo de ejecución para las consultas del grupo 3, timeslice queries, las que se se caracterizan por utilizar solo instantes de tiempo en la ventana de consulta. Es visible que en la resolución de este

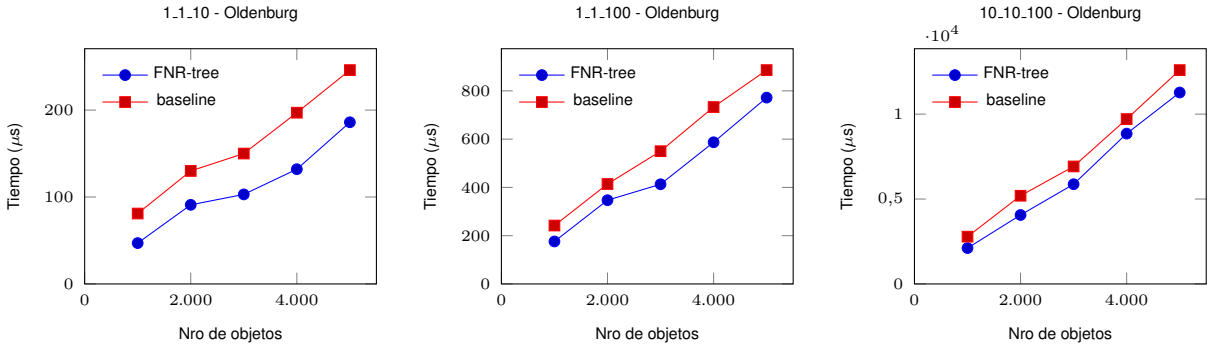


Figura 3.10: Tiempo de consulta: Grupo 2 - Oldenburg. Fuente: Elaboración propia.

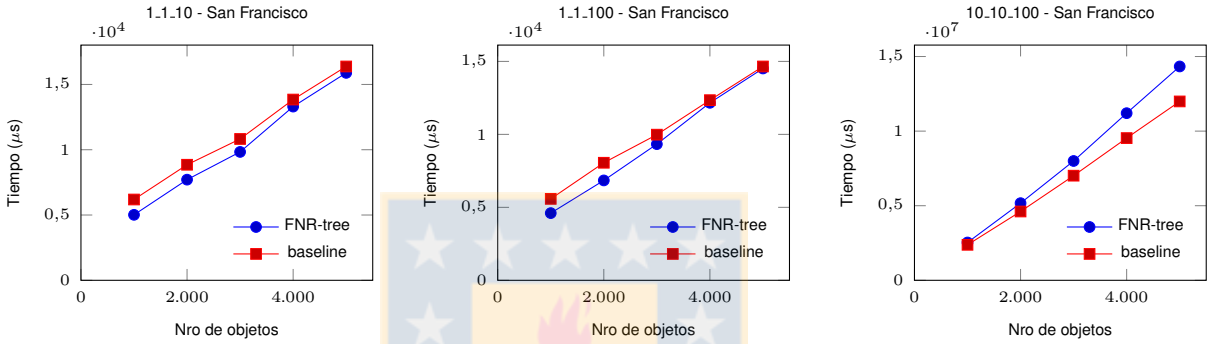


Figura 3.11: Tiempo de consulta: Grupo 2 - San Francisco. Fuente: Elaboración propia.

tipo de consultas la estructura diseñada como baseline tuvo peor desempeño que el FNR-tree en todos las pruebas.

Puesto que ambas estructuras utilizan el mismo 2D R-tree para resolver el problema espacial, toda la responsabilidad en la diferencia de rendimiento cae sobre cómo se esta resolviendo el problema temporal. Con estas pruebas se hace visible que la implementación de 1D R-tree utilizada tiene un mejor rendimiento la implementación de Interval-tree para este tipo de consultas.

Estos intervalos creados por el movimiento de objetos en la red son pequeños, puesto que los objetos pasan poco tiempo dentro de cada segmento de red. Esto implica que durante la construcción del Interval-tree sean pocos los intervalos que quedan en cada nodo (muy pocos nodos caen en el grupo I_{med} de cada nodo durante la construcción), haciendo que el árbol termine con una altura cercana, o alcanzando, al peor caso (una altura de $log_2(N)$ con N nodos, donde cada uno almacena un único

intervalo). Aún así la estructura diseñada logró superar al FNR-tree cuando la cantidad de objetos devueltos por las consultas era grande, pero este caso sólo se dio utilizando grandes ventanas sobre una red de grandes dimensiones y densidad como la de San Francisco. Esto nos dice que Interval-tree tiene un mejor desempeño mientras más grande sea el conjunto solución.

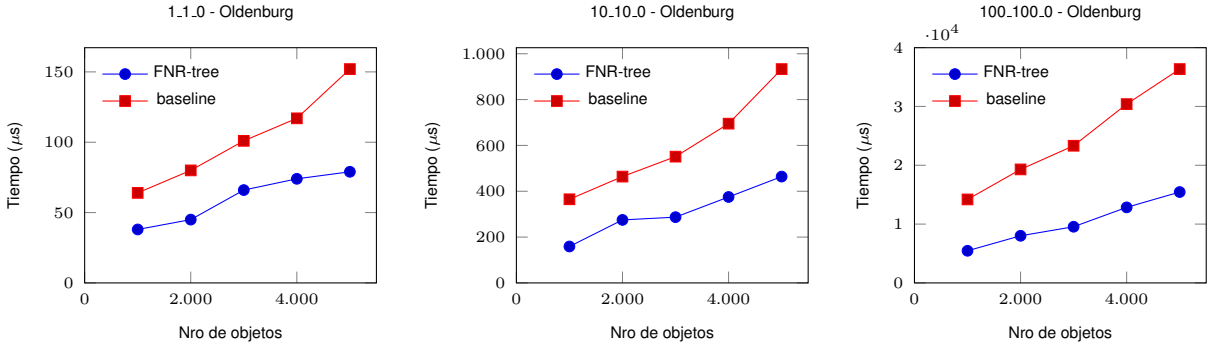


Figura 3.12: Tiempo de consulta: Grupo 3 - Oldenburg. Fuente: Elaboración propia.

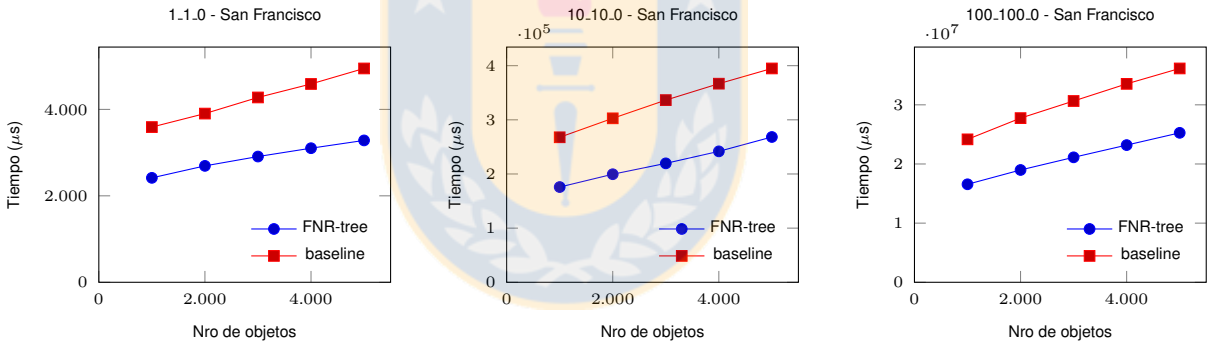


Figura 3.13: Tiempo de consulta: Grupo 3 - San Francisco. Fuente: Elaboración propia.

3.3. Discusión

Con el objetivo de encontrar un buen baseline a este trabajo de tesis se implementó una estructura de datos para el manejo de trayectorias cuyo movimiento está restringido a redes, como por ejemplo la red de transporte utilizando como base la idea del FNR-tree. Es decir, se implementó una estructura de 2 niveles utilizando un

2D R-tree (como el FNR-tree) para el nivel superior y un bosque de Interval-trees como nivel inferior en lugar de los 1D R-tree del FNR-tree original. Se quería probar si el uso de una estructura especializada en el problema de *interval intersection*, como es el Interval-tree, podría mejorar el rendimiento de la estructura para trayectorias en general.

Teniendo en cuenta los resultados, la estructura diseñada logró mejorar el uso de memoria por objeto en, aproximadamente, un 20 %, pero no logró mejorar el rendimiento de consulta del FNR-tree.

Además, desde la implementación se observó que:

- La cantidad de memoria requerida por el nivel espacial (el 2D R-tree) es constante, independientemente del número de objetos circulando por la red, y ésta se ve superada por la cantidad de memoria requerida por el nivel temporal (los Interval-trees) (ver Figura 3.14).

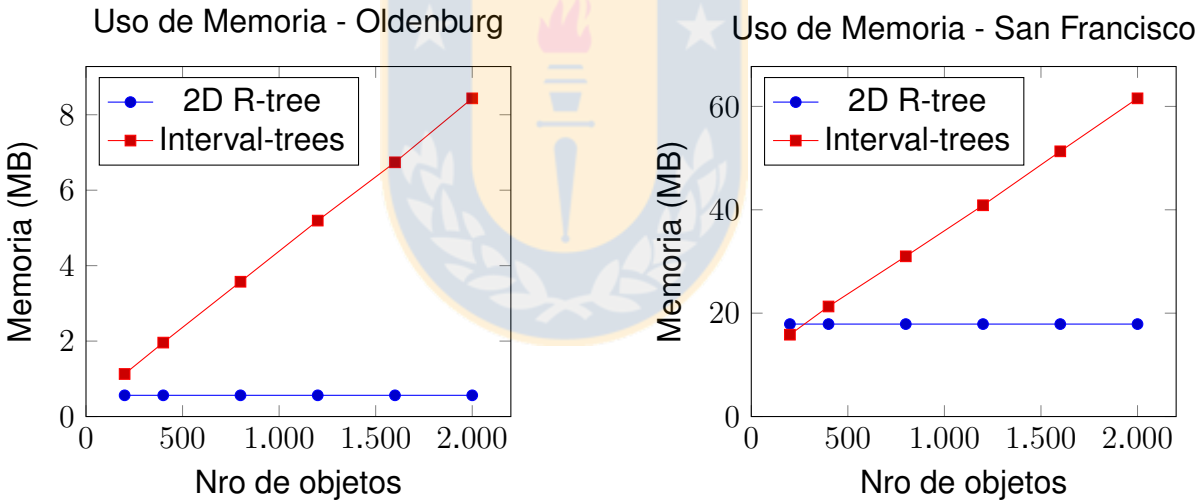


Figura 3.14: Uso de Memoria en la estructura baseline, para las 2 redes utilizadas Oldenburg y San Francisco. Fuente: Elaboración propia.

- El tamaño de los Interval-trees parece seguir una distribución estadística con la forma de una función inversamente multiplicativa ($f(x) = 1/x$) (ver Figura 3.15). Sin embargo, si la cantidad de intervalos de tiempo en la estructura crece lo suficiente, la distribución empieza a tomar una forma de asimetría positiva que

se va desplazando hacia la derecha. Sin embargo, la cantidad de segmentos de trayectoria requeridos para hacer visible esto es muy grande.

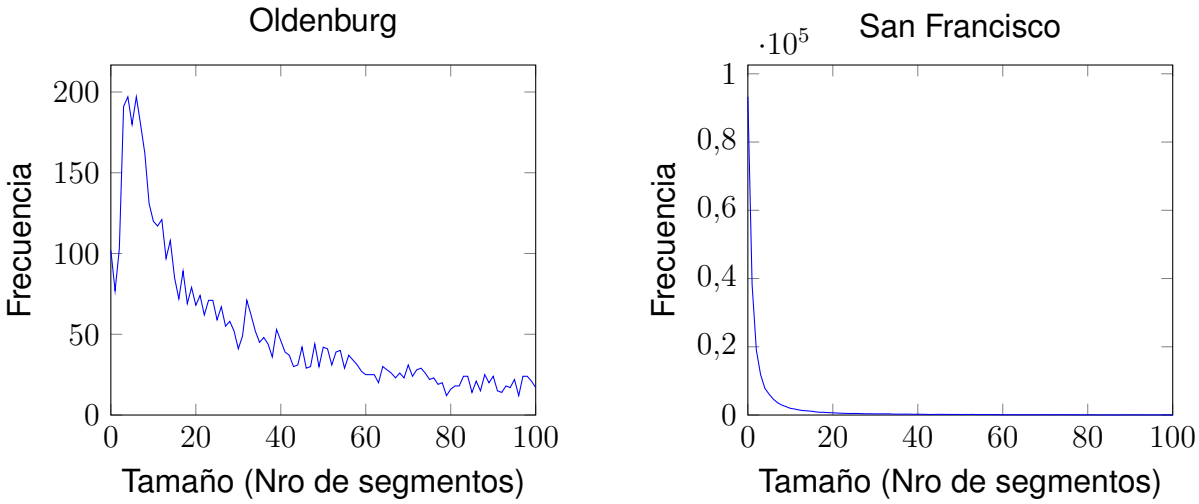


Figura 3.15: Distribución de tamaños de los Interval-trees, para las 2 redes utilizadas Oldenburg y San Francisco con 5.000 objetos circulando. Fuente: Elaboración propia.

Por tanto, nuestra propuesta es, dado que el nivel temporal es el que más memoria utiliza y que esta cantidad de memoria irá creciendo producto del nivel de dinamismo de las trayectorias (que el nivel espacial no tiene, puesto que la red de transporte rara vez se modifica), utilizar alguna estructura de datos compacta en lugar de los Interval-trees en el nivel temporal para reducir este requerimiento de memoria, y estudiar la distribución de los tamaños de éstos para ver si es conveniente, en algunos casos cuando para un segmento de red la cantidad de intervalos de tiempo a manejar es lo suficientemente pequeña, obviar el uso de una estructura de datos en dicho segmento.

De la Figura 3.15 se desprende también que, por lo general, las estructuras de la parte temporal reciben una muy pequeña cantidad de intervalos. En el caso de 5.000 objetos circulando por San Francisco el $41\,\%$ de las estructuras de la parte temporal no recibió ningún intervalo, mientras que solo un $24\,\%$ de estas recibió 5 o más intervalos.

Capítulo 4

Nuestra Propuesta

Nuestra propuesta consiste en una estructura similar al FNR-tree, conservando la idea de utilizar 2 niveles, pero reemplazando los 1D R-tree que utiliza para resolver el problema de *Interval Intersection* por alguna estructura especializada en dicho problema.

La primera sección de este capítulo describe 2 nuevas alternativas para el nivel temporal, que se suman a las ya descritas 1D R-tree e Interval-tree, las que serán evaluadas más adelante para seleccionar la mejor alternativa para formar la solución completa que se describe en la segunda sección de éste capítulo.

4.1. Estructuras de Datos para Interval Intersection

En la literatura se pueden encontrar distintas estructuras de datos que solucionan el problema de *Interval Intersection*, entre estas están el 1D R-tree y el Interval-tree que ya fueron probados como parte de una estructura para trayectorias restringidas a redes. En está sección se presentan 2 nuevas alternativas para resolver este problema, una encontrada en la literatura y una nueva que hace uso de estructuras compactas.

4.1.1. Schmidt

La primera de las alternativas que adaptaremos a nuestro problema fue presentada por J. Schmidt [30], como una estructura para resolver el problema de *Interval Stabbing* en la que define al *padre* de un intervalo como el intervalo que empieza más a la derecha que lo cubre completamente. Esta relación padre forma un árbol, donde los hermanos son ordenados de izquierda a derecha (notar que dos hermanos no pueden cubrirse el uno al otro), y la raíz del árbol es un nodo especial que actúa de padre de todos los intervalos que no son cubiertos por ningún otro. La estructura almacena para

cada punto el intervalo que comienza más a la derecha que interseca dicho punto, en un arreglo llamado `start` (ver Figura 4.1).

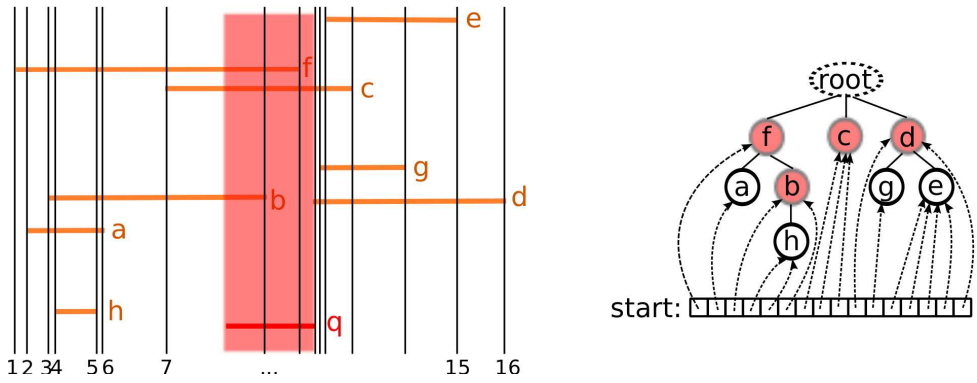


Figura 4.1: Represetación de un grupo de intervalos usando la estructura de Schmidt. Fuente: [8]

Para resolver una consulta de *Interval Stabbbing* primero obtiene, de existir, `start(q)` para reportarlo como parte de la solución. Luego, el algoritmo, de manera recursiva, reporta los hermanos a la izquierda del nodo hasta el primero que no interseque con `q`, buscando también entre los hijos derechos de los nodos reportados; esto para `start(q)` y todos sus ancestros.

Para extender esta estructura para hacerla capaz de resolver el problema de *Interval Intersection* es necesario el uso de un segundo arreglo, `start2`, que almacene en `start2(x)` el intervalo más a la derecha con extremo izquierdo menor o igual a x [8].

Con esto el proceso de consulta por un intervalo (l_q, r_q) es similar al de *Interval Stabbing*, pero comenzando en $max(start(l_q), start2(r_q))$ y los intervalos son reportados mientras sus extremos derechos sean mayores o iguales a l_q.

Por ejemplo, en la Figura 4.1, $start(l_q)$ apunta al intervalo c, y $start2(r_q))$ al intervalo d, por lo que el algoritmo empezará con el intervalo d puesto que éste es mayor, es decir, empieza después que c. El intervalo d es marcado como parte de la solución y se procede a comprobar que su hermano izquierdo, c, cumpla con la condición $r_c \geq l_q$. Ésta se cumple, por lo que c es marcado como solución y se procede a evaluar recursivamente la condición en su hermano izquierdo, f, y su hijo más a la derecha, que éste nodo en particular no tiene. El intervalo f cumple la condición, se marca como solución y se procede a evaluar su hijo más a la derecha, b. Éste también cumple por

lo que es parte de la solución, se procede a evaluar a y h, los que no cumplen la condición y el algoritmo termina.

Encontrar el intervalo que inicia el recorrido ($max(start(l_q), start2(r_q))$) es realizado en tiempo constante $O(1)$, y la evaluación para verificar que cada uno de los k intervalos visitados durante el recorrido es intersecado por el intervalo q también es realizada en tiempo constante $O(1)$, por lo que la complejidad algorítmica de la consulta es de $O(1+k)$, donde k es el número de intervalos intersecados, es decir, el tamaño del conjunto solución.

4.1.2. Estructura compacta con descomposición en IIS

La segunda estructura fue adaptada de una estructura propuesta para el dominio espacial [8]. Para empezar a describirla primero se requiere de la siguiente definición:

Un conjunto de intervalos $I = \{i_1, i_2, ..., i_n\}$ es independientes (o IIS, por *Independent Interval Set*) si no existe ningún intervalo $i_j \in I$ que esté completamente contenido en otro intervalo $i_k \in I$.

Reportar los k intervalos de un *IIS* que intersecan con un intervalo de consulta $Q = [L^q, R^q]$ puede ser fácilmente resuelto si tenemos los intervalos en orden. Notar que, por definición, el orden de los extremos izquierdos de los intervalos es el mismo al de los extremos derechos en un *IIS*. Si se localizan el primer y el último intervalo intersecado por la consulta, basta iterar entre ambos para devolver todos los intervalos intersecados.

Esto requiere un espacio de tiempo discretizado, por lo que no se puede utilizar directamente sobre instantes de tiempo que incluyan una parte decimal. Esto se puede solucionar fácilmente multiplicando por algún factor que luego permita eliminar la parte decimal sin perder precisión. Por ejemplo, si trabajamos con instantes de tiempo con hasta 6 decimales basta multiplicar cada uno de ellos por 10^6 para discretizar el espacio.

Esta estructura utiliza 2 bitvectors por cada *IIS*, uno para los extremos izquierdos, `start`, y otro para los extremos derechos, `end`, de cada intervalo en el conjunto (ver Figura 4.2).

Estos bitvectors, o bitmaps, son una pieza fundamental, y son encontrados, en la mayoría de las estructuras compactas. Dada una secuencia binaria de tamaño n, un bitmap $B[1, n]$ provee acceso a cualquier posición i de la secuencia, contar el número de ocurrencias de un bit v hasta una posición i, $rank_v(B, i)$ y devolver la posición de la j-ésima ocurrencia de un bit v, $select_v(B, j)$. Todas estas operaciones pueden ser soportadas en tiempo constante $O(1)$ con espacio de $O(n + o(n))$ [18]. Cuando la secuencia de bits es dispersa (el número de 1s, m, es mucho menor a n), como es el caso, el espacio puede ser mejorado a $n \times H_0(B) + O(m)$ bits, soportando $select$ en $O(1)$ y $rank$ en $O(min(log(m), log(\frac{n}{m})))$ [25].

Usando estos 2 bitvectors se puede encontrar el primer intervalo intersecado fácilmente con una operación `rank(`L^q`)` en el bitvector `end` y el último con `rank(`R^q`)` en `start`, siendo el resultado de la consulta todos los intervalos entre estos dos.

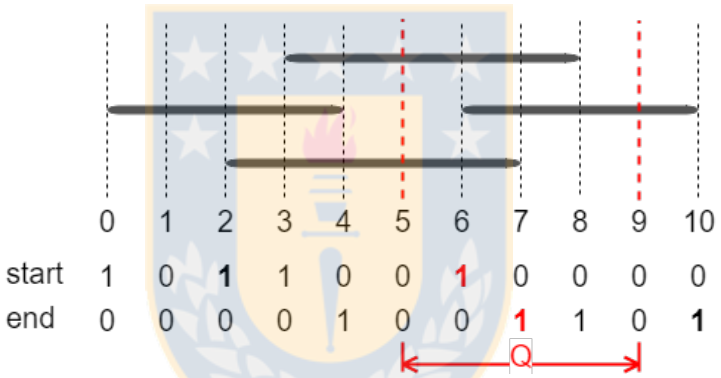


Figura 4.2: Un conjunto de intervalos independientes (*IIS*) y su respectiva representación con 2 bitvectors. En rojo el último intervalo intersecado en `start` y el primero en `end`. Fuente: Elaboración propia.

Por cada uno de los s IIS, los extremos del recorrido se encuentran con 2 operaciones `rank`, de complejidad $O(log(\frac{n}{m}))$, donde n es el tamaño del bitvector, es decir, el tamaño de la dimensión temporal multiplicado por el factor de discretización, y m es el número de intervalos en el IIS, mientras que el recorrido entre estos extremos visita los k intervalos que forman parte de la solución sin realizar ninguna operación adicional, dando a esta estructura una complejidad algorítmica de consulta de $O(s \times \log(\frac{n}{m}) + k)$. Por la naturaleza de los intervalos de trayectorias, el número de IIS s es relativamente bajo, pues que un intervalo cubra completamente otro es un caso poco frecuente, ya

que estos son de tamaño similar, 2 vehículos demoran una cantidad de tiempo similar en recorrer una cuadra.

Si el conjunto de intervalos no es independiente, éste es descompuesto en conjuntos que sí cumplen esta propiedad. Para eso, primero se ordenan los intervalos según su extremo izquierdo, y se itera sobre estos. Por cada intervalo i, se itera sobre los conjuntos independientes verificando que el extremo derecho de i sea mayor al extremo derecho del último intervalo de cada uno de estos conjuntos. Entre los que cumplieron lo anterior, i es insertado en el conjunto cuyo último intervalo tenga mayor extremo derecho, de no existir tal conjunto se crea uno nuevo conteniendo a i.

4.2. Solución completa

Habiendo revisado diversas posibles alternativas de estructura de datos que resuelven el problema de *Interval-intersection* para la parte temporal de una estructura de datos para trayectorias sujetas a redes, y dados los resultados de la sección 5.1, se diseña una nueva estructura cominando los bloques ya explicados.

4.2.1. Estructura

El primer nivel, el espacial, consta de un 2D R-tree que representa la red de transporte (el mismo usado por el FNR-tree y el baseline descrito en el capítulo 3), donde cada una de sus hojas representa un segmento de la red, al que le es asociada una estructura compacta con descomposición en IIS como la descrita en la sección 4.1.2, la que indexará los intervalos de tiempo por los cuales cada objeto pasa por dicho segmento.

La Figura 4.3 ilustra esquemáticamente la estructura propuesta.

4.2.2. Algoritmos

Tanto el proceso de construcción como el de búsqueda es el mismo al descrito para el baseline en el capítulo 3, salvo por la estructura compacta que, ahora, toma el lugar del Interval-tree.

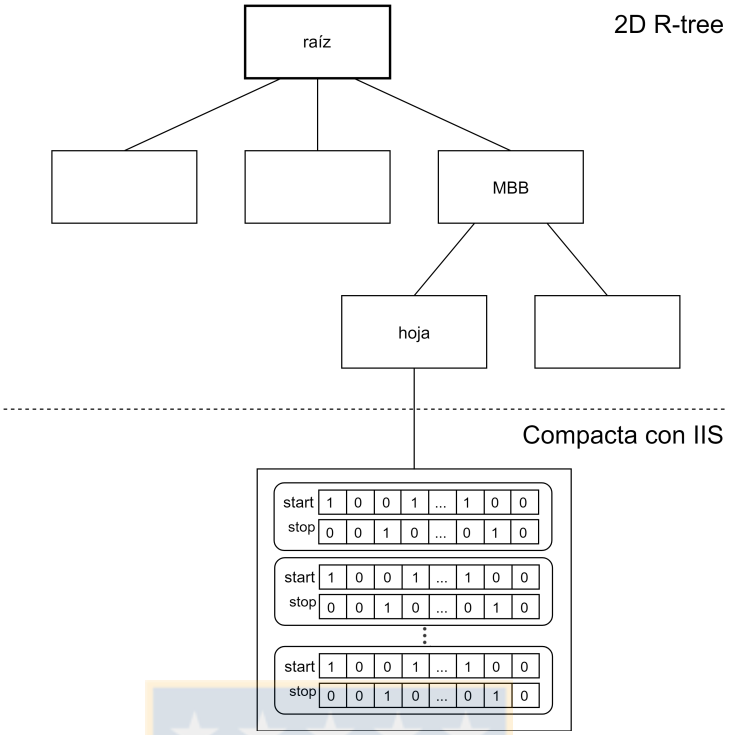


Figura 4.3: Represetación de la estructura propuesta. Fuente: Elaboración propia

Construcción

El proceso de construcción requiere de la red, representada en 2 archivos como nodos y arcos, y un tercer archivo con las trayectorias en sí, representadas como un log de la ubicación de los objetos en la red.

la red, la que es representada en 2 archivos: nodos y arcos, y un archivo con las trayectorias.

Esta estructura, al igual que el baseline, es estática, por lo que no permite modificaciones tras su construcción, por esto en las hojas espaciales se usa un arreglo auxiliar para almacenar, temporalmente, los intervalos antes de construir las estructuras de la parte temporal.

```
Contrucción
```

```
1- La red ingresa a la estructura, primero se lee el archivo
que contiene los nodos y luego el de arcos, creando el 2D R-tree.
```

```
2- Se lee el archivo de trayectorias, y cada vez que un objeto
deja un segmento de red S:
    2.1- Se busca el segmento S en el 2D R-tree.
    2.2- Se obtiene la dirección del movimiento dir.
    2.3- Se inserta con el intervalo I = (t_entrada, t_salida)
    el id del objeto y dir en el arreglo auxiliar de la hoja
    espacial del segmento S.

3- Se itera sobre todas las hojas espaciales construyendo a
partir del arreglo auxiliar la correspondiente estructura
compacta con descomposición en IIS, eliminando el arreglo
posteriormente.
```

Búsqueda

Como ya se mencionó, las búsquedas en la estructura tienen la forma de una ventana tridimensional, con los siguientes argumentos: X_{min}, Y_{min}, X_{max}, Y_{max}, T_{min} y T_{max}. Con esta ventana el proceso de búsqueda se describe a continuación:

Búsqueda

```
1- Buscar el conjunto de segmentos S cuyos MBB intersequen
con la ventana de búsqueda (X_min,Y_min,X_max,Y_max).

2- Buscar en la estructura del nivel temporal de cada uno de los
segmentos en S los intervalos que intersequen con el intervalo
temporal (T_min, T_max).

3- Por cada segmento en S donde la consulta del paso (2) tuvo
```

```
resultado, verificar si el segmento interseca con la ventana de
búsqueda (X_min, Y_min, X_max, Y_max), de ser así, agregar los
objetos encontrados en el paso (2) al conjunto resultado (donde
no hay duplicados), de lo contrario descartarlos.
```

El primer paso del algoritmo recién descrito tiene una complejidad en el peor caso de $O(N)$, siendo N el número total de segmentos de red, cuando la ventana de consulta intersecta con todas las MBB. El segundo paso toma $O(s \times \log(\frac{n}{s \times m}) + k)$ según lo mencionado en la sección 4.1.2, la comprobación del último paso toma tiempo constante $O(1)$ y la inserción en el conjunto solución una complejidad de $\log(k)$. Lo que da a la consulta una complejidad de $O(N \times s \times \log(\frac{n}{s \times m}) + K \times \log(K))$, siendo K el tamaño total del conjunto solución.

Capítulo 5

Evaluación Experimental

5.1. Evaluación de las estructuras de Interval-Intersection

Para evaluar el rendimiento de las 4 estructuras implementadas que solucionan el problema de *Interval-Intersection* y, finalmente elegir la mejor para el problema de trayectorias, se realizaron distintos experimentos.

las estructuras fueron implementadas con el lenguaje de programación C++ (11), fueron compiladas con G++ 5.4.0 y fueron probadas en un computador con las mismas especificaciones de la sección 3.2. Adicionalmente, se utilizaron los códigos de R-tree [1], Interval-tree [13], Schmidt [31], y la biblioteca SDSL [14].

5.1.1. Comportamiento según la naturaleza de los intervalos

Para evaluar el comportamiento de las distintas estructuras según la naturaleza de los intervalos se crearon 3 conjuntos de 800.000 intervalos:

- De tamaño definido, donde todos los intervalos son del mismo tamaño, equivalentes a un 10 % del espacio temporal.
- De tamaño aleatorio, donde los intervalos tienen tamaños distintos, pudiendo ir desde un instante puntual hasta el espacio completo.
- Obtenidos de trayectorias, donde los intervalos fueron extraídos de un conjunto de trayectorias generadas con el generador de Brinkhoff [3] sobre la red de San Francisco.

Los resultados para el primer conjunto, intervalos de tamaño definido, (ver Figura 5.1) muestran que la estructura compacta con descomposición en conjuntos de intervalos independientes (IIS) obtiene los mejores resultados tanto en tiempo de consulta

como en uso de memoria, mientras mantiene un tiempo de construcción competitivo con las otras estructuras. Este conjunto de intervalos en particular tiene como característica que, puesto que todos los intervalos son de igual tamaño, ninguno incluye a otro (salvo por cuestiones de precisión). Esto implica que la cantidad de IIS creados es muy baja (sólo 6 para 800.000 intervalos) lo que permite tiempos de construcción relativamente buenos.

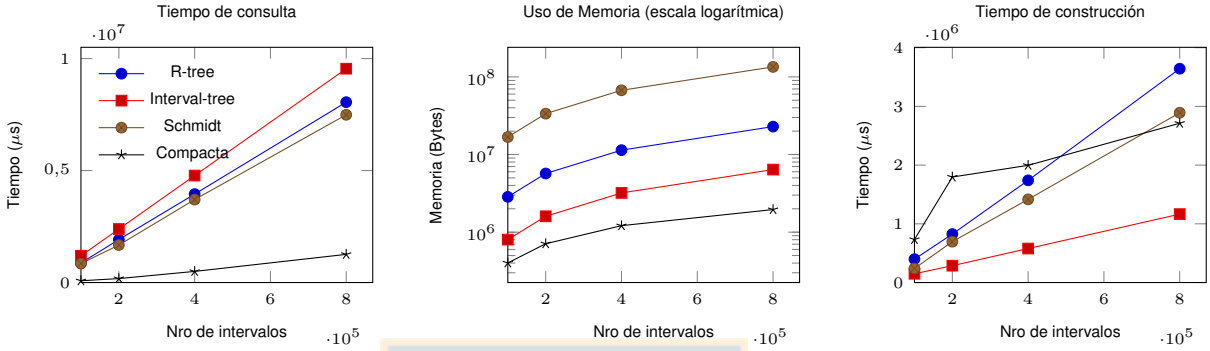


Figura 5.1: Intervalos de tamaño definido. Fuente: Elaboración propia.

Los resultados para el segundo conjunto (ver Figura 5.2) mantienen a la estructura compacta como la con mejores resultados en tiempo de consulta y uso de memoria (en un empate con el Interval-tree), mientras obtiene un tiempo de construcción mayor al de las demás estructuras. El motivo (cerca de 900 veces el tiempo de construcción del Interval-tree para 800.000 intervalos) es el gran número de IIS obtenidos (3.273 para los 800.000 intervalos, a diferencia de los 6 para el mismo número de intervalos de tamaño definido).

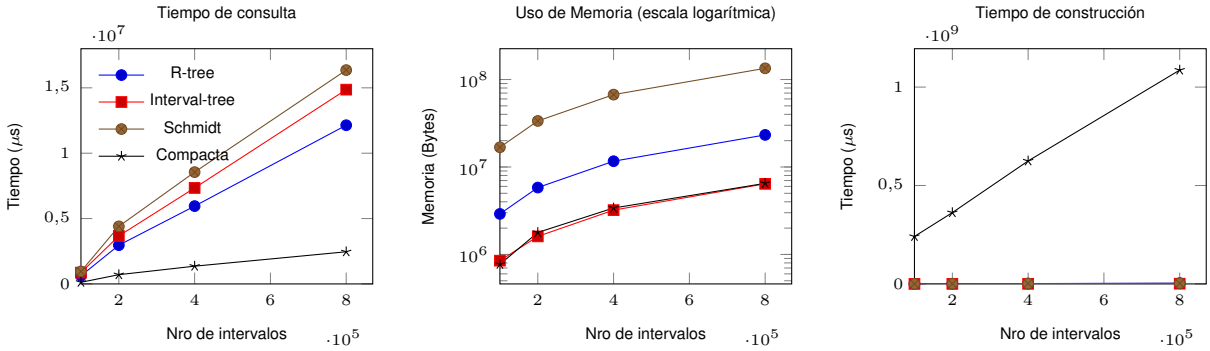


Figura 5.2: Intervalos aleatorios. Fuente: Elaboración propia.

Los resultados para el último conjunto de intervalos (ver Figura 5.3), extraídos de

trayectorias, muestran un comportamiento que está entre los 2 escenarios anteriores, mostrando una mayor similitud al primero (tamaños definidos) que al segundo (tamaños aleatorios), esto debido a que el tiempo que demoran distintos vehículos en circular por la misma calle es similar.

Si bien el tiempo de construcción sigue superando con holgura a los de las otras estructuras, la diferencia no es tanta como en el caso anterior (cerca de 7 veces el tiempo de construcción del Interval-tree para 800.000 intervalos), esto debido a que el número de IIS no crece demasiado (29 para 800.000 intervalos).

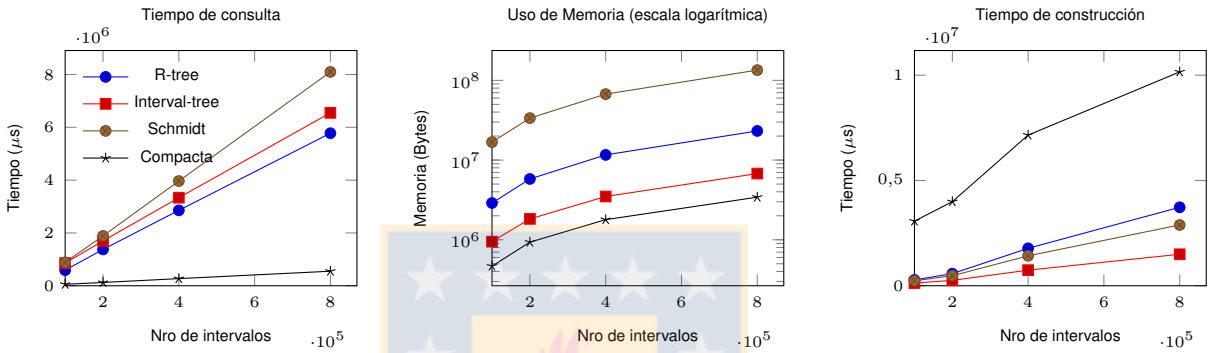


Figura 5.3: Intervalos de trayectorias. Fuente: Elaboración propia.

5.1.2. Comportamiento según la precisión usada

Además de evaluar el comportamiento de las estructuras según la naturaleza de los intervalos se evaluó el comportamiento según el nivel de precisión usado, en éste experimento se utilizaron los mismos 800.000 intervalos extraídos de trayectorias del experimento anterior.

Los resultados (Ver Figura 5.4) muestran que la estructura que demuestra mayor sensibilidad a la precisión con la que se trabajan los intervalos es la compacta. Esto es producto del proceso de discretización del espacio que requiere esta estructura donde, a mayor precisión, más grande es el espacio resultante tras este proceso. Por ejemplo, el uso de 8 decimales implica que el espacio crezca 10^8 veces. Aún así la estructura compacta muestra mejores resultados en tiempo de consulta y uso de memoria que las demás estructuras.

Cabe mencionar que los experimentos de la sección 5.1.1 se realizaron utilizando una precisión de 7 decimales, la máxima cantidad decimales encontrada en los

intervalos sacados de las trayectorías generadas por el generador de Brinkhoff. Por tanto, en aplicaciones donde el usuario esté dispuesto a perder precisión el ahorro de memoria puede ser considerable, mejorando también ligeramente los tiempos de consulta.

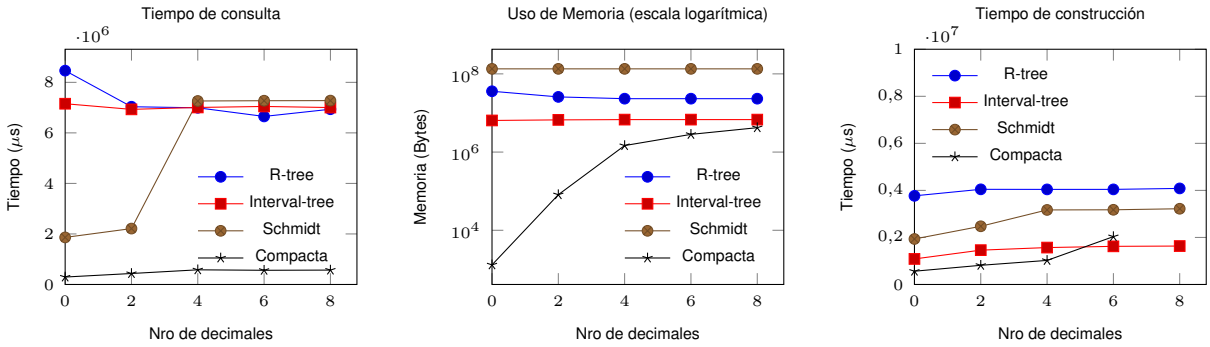


Figura 5.4: Comportamiento según precisión usada. En el tercer gráfico, el último punto de la estructura compacta es omitido debido a un salto exponencial que lo hace demorar 40 veces más que el R-tree. Fuente: Elaboración propia.

5.1.3. Discusión Interval-Intersection

De acuerdo a los resultados anteriores la estructura compacta con descomposición en IIS parece ser la mejor candidata a reemplazar al 1D R-tree en una nueva estructura de datos para el manejo de trayectorias sobre redes.

Recordando las conclusiones del capítulo 3.3, y en especial lo mostrado en la Figura 3.15, sabemos que la carga que recibe cada una de las estructuras de la parte temporal es pequeña en relación a lo que recibe la estructura para trayectorias en sí.

Por ejemplo, para el caso de mayor tamaño analizado, “5.000 objetos en San Francisco durante 100 unidades de tiempo”, con un peso de aproximadamente 300MB, se repartieron 3.986.007 intervalos de tiempo entre 223.606 segmentos de red. Un $41\,\%$ de los segmentos quedaron vacíos, y el $78\,\%$ de los segmentos no recibió más de 5 intervalos.

Por lo tanto se realizó otro experimento, con datasets pequeños de intervalos, con el fin de evaluar el rendimiento de las estructuras trabajando con datasets pequeños

donde además parte importante de las consultas tienen resultados vacíos (Ver Tabla 5.1).

Tiempo de consulta (milisegundos)		
Estructura	10 intervalos	100 intervalos
R-tree	1,77	12,76
Interval-tree	7,42	28,72
Schmidt	10,41	20,50
Compacta	4,20	13,31

Tabla 5.1: Tiempos de consulta para datasets pequeños de intervalos. Fuente: Elaboración propia.

Para ambos datasets se respondieron a las mismas 20.000 consultas, para el primer dataset el 90 % de estas consultas produjo un resultado vacío, mientras que para el segundo un 35 % de estas consultas fueron vacías, otro 35 % de tamaño 1 y un 20 % de tamaño 2.

Se puede ver que el R-tree es el mejor al momento de trabajar con datasets pequeños y responder a consultas vacías, dejando atrás al Interval-tree, lo que da más sentido a los resultados del capítulo 2. Le sigue en desempeño la estructura compacta, la que obtiene buenos resultados, lo que le permite seguir siendo una buena candidata para reemplazar al 1D R-tree del FNR-tree.

5.2. Comparación de la solución completa con FNR-tree y el baseline

Para comparar la nueva estructura para trayectorias sobre redes implementada con el FNR-tree y su modificación, el baseline, se realizaron los mismos experimentos de la sección 3.2, pero agregando esta tercera estructura.

Los dataset que se utilizaron son los mismos, es decir, creados por el generador de trayectorias sobre redes de Brinkhoff [3] para ambas redes, Oldenburg (de 6.105 nodos y 7.305 arcos) y San Francisco (de 175.343 nodos y 223.606 arcos), para 1.000, 2.000, 3.000, 4.000 y 5.000 objetos durante 100 unidades de tiempo.

Las 3 estructuras fueron implementadas con el lenguaje de programación C++ (11),

compiladas con G++ 5.4.0 y probadas en un computador con las mismas especificaciones de la sección 3.2. Para FNR-tree e Interval-tree se utilizaron las implementaciones del trabajo previo, mientras que en la implementación de la estructura propuesta se utilizaron los códigos de R-tree [1] y la biblioteca SDSL [14].

5.2.1. Uso de espacio y tiempo de construcción

Primero se comparó la cantidad de memoria necesaria por cada una de estas estructuras. Como se puede ver en la Figura 5.5, la cantidad requerida por la estructura compacta se mantuvo por debajo de las demás en todo momento, obteniendo una mayor ventaja mientras más objetos circulan por la red. Lo que, reafirmando los resultados obtenidos en la sección 5.1, nos dice que la estructura compacta con descomposición en IIS requiere de una menor cantidad de memoria que el 1D R-tree y el Interval-tree. En este experimento sólo se consideraron las estructuras de la parte temporal que tenían 5, o más, intervalos.

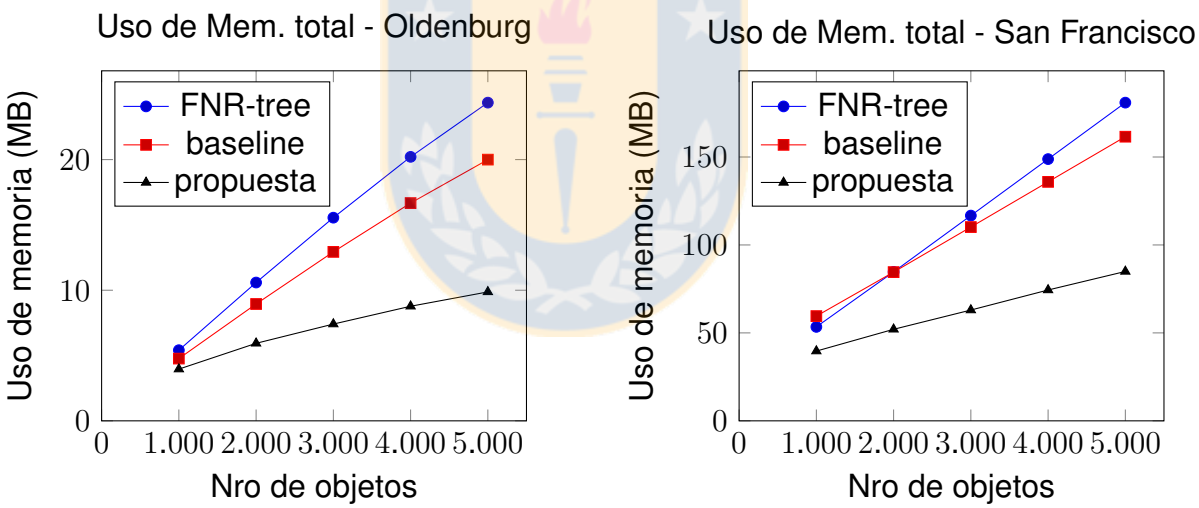


Figura 5.5: Uso de memoria total de las 3 estructuras para ambas ciudades. Fuente: Elaboración propia.

La cantidad aproximada de memoria utilizada por objeto para las 100 unidades de tiempos es mostrada en la Tabla 5.2, y muestra que estructura propuesta requiere de aproximadamente un $70\,\%$ menos memoria que el FNR-tree y un $60\,\%$ menos que la estructura baseline. Para el dataset más grande, “5.000 objetos por San Francisco” de

299 MB, FNR alcanzó un uso de memoria de 181 MB, el baseline un uso de 162 MB y la propuesta 85 MB.

Uso de memoria por objeto		
Estructura	Oldenburg	San Francisco
FNR-tree	∼5 KB	∼32 KB
baseline	∼4 KB	∼26 KB
propuesta	∼1.2 KB	∼11 KB

Tabla 5.2: Uso de memoria por objeto. Fuente: Elaboración propia.

En cuanto al tiempo de construcción, mostrado en la Figura 5.6, se puede apreciar que la estructura propuesta obtiene los peores tiempos con amplia diferencia sobre las otras 2 estructuras, alcanzo incluso a tardar cerca de 3.000 veces el tiempo de construcción del baseline para el caso de 5.000 objetos en San Francisco.

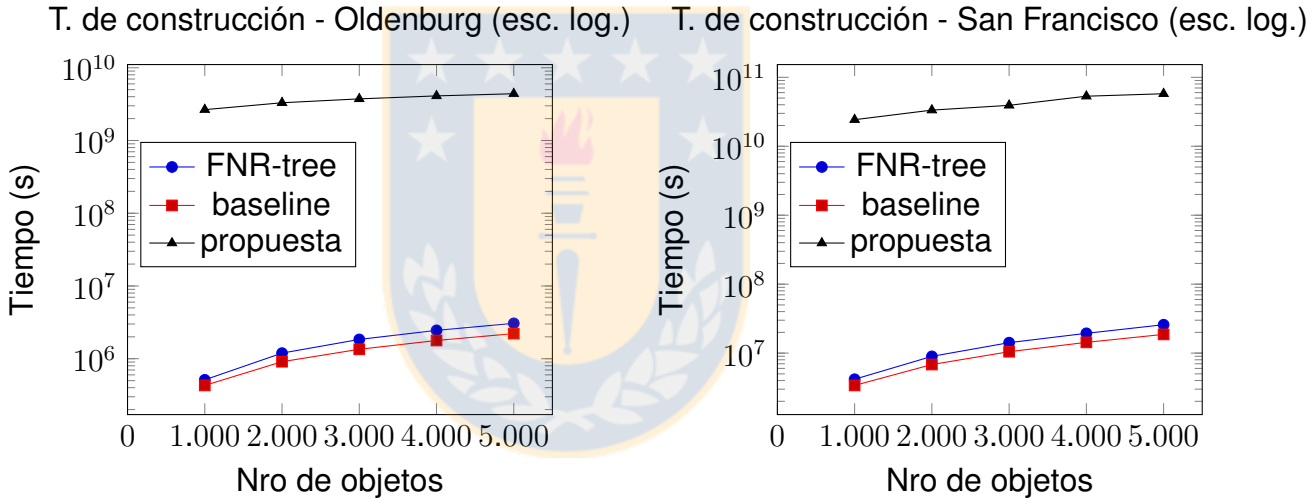


Figura 5.6: Tiempo de construcción para las 3 estructuras (escala logarítmica). Fuente: Elaboración propia.

5.2.2. Tiempo de consulta

Para comparar el rendimiento de las 3 estructuras al resolver consultas se utilizaron los mismos conjuntos de consultas usados la sección 3.2.3. Recordando, se distinguieron 3 tipos de consultas, y para cada uno de estos se crearon 3 conjuntos de 500 consultas:

- En el primer grupo los conjuntos utilizan 1 %, 10 % y 20 %, respectivamente, de cada una de las 3 dimensiones.

- En el segundo grupo, los primeros 2 conjuntos utilizan un 1 % de ambas dimensiones espaciales con un 10 % y 100 % de la dimensión temporal respectivamente, mientras que el tercer conjunto utiliza un 10 % de cada dimensión espacial con un 100 % de la dimensión temporal.

- En el tercer grupo los conjuntos utilizan un 1 %, 10 % y 100 % de ambas dimensiones espaciales respectivamente con un único punto de la dimensión temporal, estas consultas en particular son conocidas como “timeslice queries”.

Grupo 1: Consultas de rango con igual uso de cada dimensión.

En las Figuras 5.7 y 5.8 se muestran los resultados de tiempo de ejecución para las consultas del grupo 1, para las redes de Oldenburg y San Francisco, respectivamente. Se puede notar que para ambas redes la estructura propuesta mantiene los mejores tiempos de consulta, incluyendo un empate con el FNR-tree en el caso de la ventana más pequeña en la red de San Francisco, aunque es importante destacar que en este caso la mayoría de las consultas tienen respuesta vacía.

Como se puede ver, a medida que el tamaño de la ventana de consulta tridimensional crece la estructura diseñada logra aumentar su ventaja, lo mismo para el número de objetos.

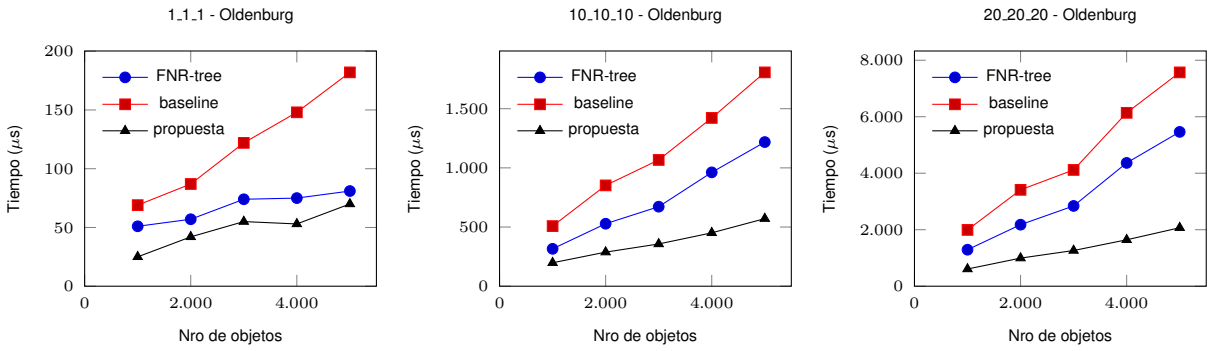


Figura 5.7: Tiempo de consulta: Grupo 1 - Oldenburg. Fuente: Elaboración propia.

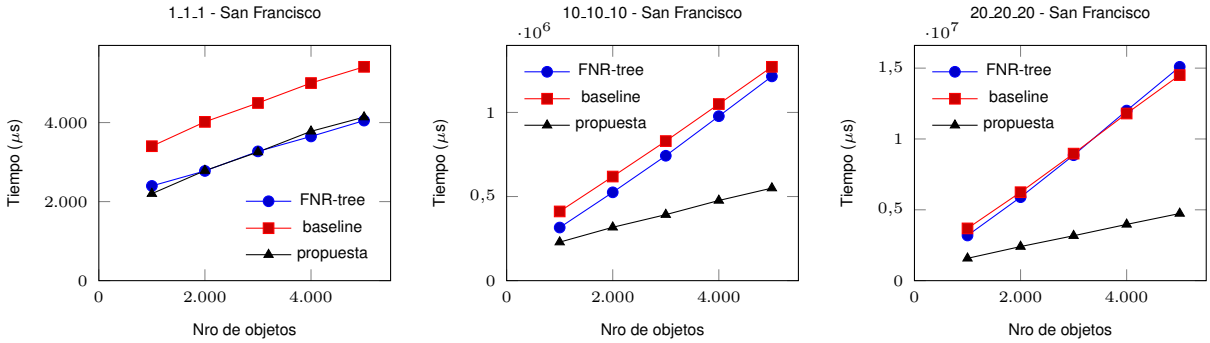


Figura 5.8: Tiempo de consulta: Grupo 1 - San Francisco. Fuente: Elaboración propia.

Grupo 2: Consultas de rango con gran uso de dimensión temporal.

En las Figuras 5.9 y 5.10 se puede ver el resultado del tiempo de consultas para el grupo 2 de consultas, que se caracterizan por utilizar una gran parte de la dimensión temporal (los 2 últimos usando la totalidad de ésta). Al igual que para anterior, la estructura propuesta logra los mejores tiempos de consulta en todo momento.

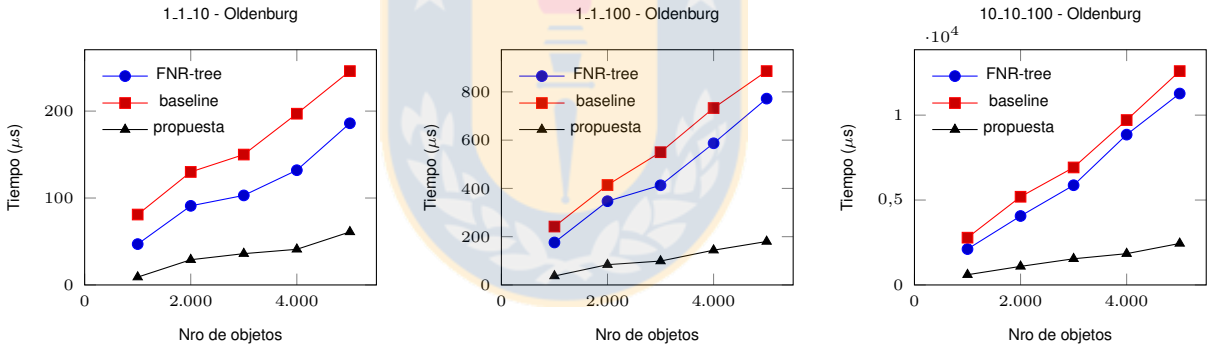


Figura 5.9: Tiempo de consulta: Grupo 2 - Oldenburg. Fuente: Elaboración propia.

Grupo 3: Timeslice queries.

Las Figuras 5.11 y 5.12 muestran los resultados del tiempo de ejecución para las consultas del grupo 3, timeslice queries, las que se se caracterizan por utilizar solo instantes de tiempo en la ventana de consulta. Es visible que en la resolución de este tipo de consultas la estructura propuesta no logró buenos resultados. Si bien para

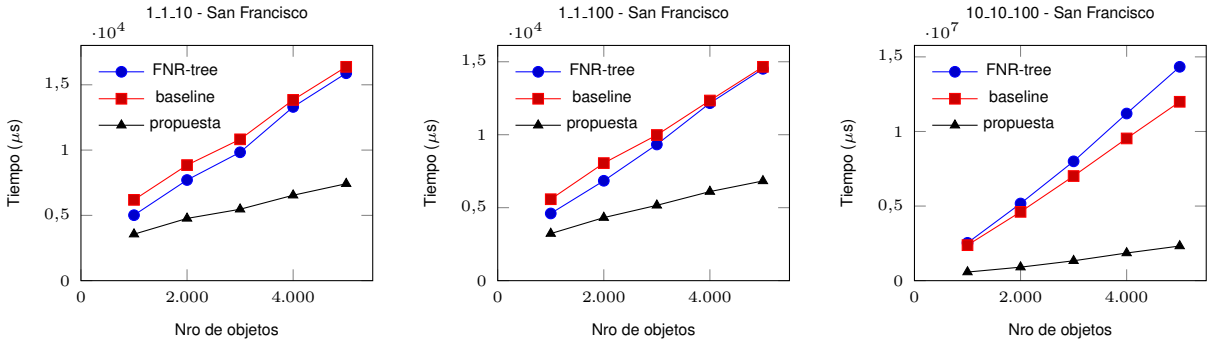


Figura 5.10: Tiempo de consulta: Grupo 2 - San Francisco. Fuente: Elaboración propia.

la red de Oldenburg aún mantiene tiempos competitivos, para la red de San Francisco obtiene incluso peores resultados que el baseline, esto muestra una debilidad al momento de resolver el problema de *Interval-stabbing*, recordando que este es el problema particular donde en el problema de *Interval-intersection* la consulta es un único punto.

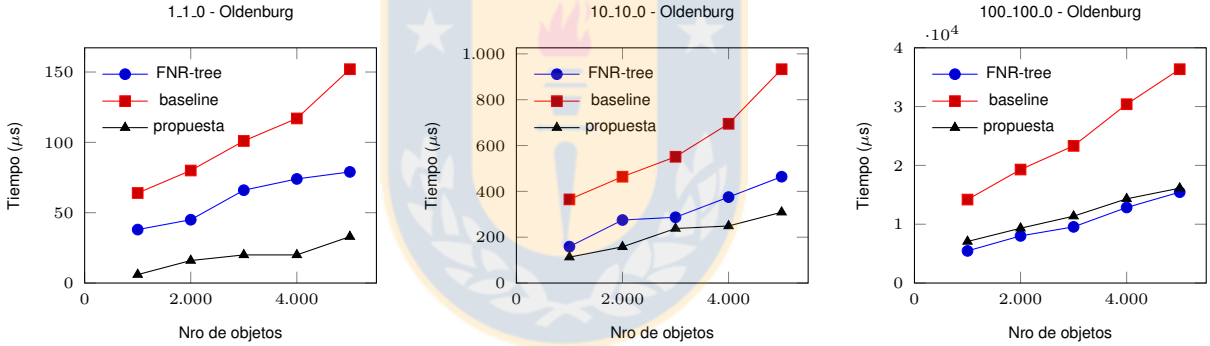


Figura 5.11: Tiempo de consulta: Grupo 3 - Oldenburg. Fuente: Elaboración propia.

El mal rendimiento de la estructura propuesta en este grupo, y en el primer conjunto del grupo 1 (1 % de la dimensión espacial), es atribuible al pequeño tamaño del conjunto solución devuelto por cada una de las estructuras de la parte temporal, lo que hace que la mayor carga de la búsqueda se la lleve el proceso de encontrar el primer intervalo intersecado, esto para las 3 estructuras, lo que la estructura propuesta no debe hacer una única vez por cada segmento como si lo hacen el FNR-tree y el baseline, sino que debe hacerlo una vez por cada IIS, lo que significa una cantidad considerablemente mayor a las demás. Además, la mayor ventaja de la estructura compacta con

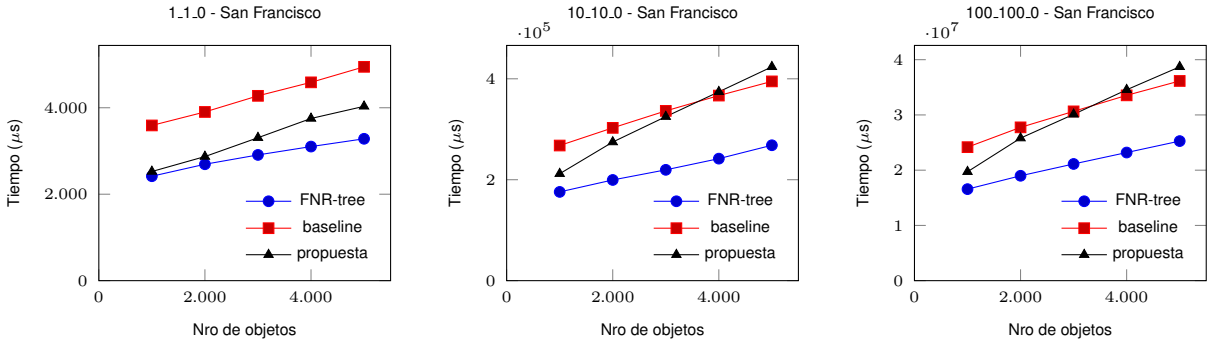


Figura 5.12: Tiempo de consulta: Grupo 3 - San Francisco. Fuente: Elaboración propia.

descomposición en IIS era precisamente la de iterar sobre los intervalos solución sin requerir de hacer comprobaciones, lo que no puede ser aprovechado.

Capítulo 6

Conclusiones y Trabajo Futuro

En la búsqueda de una nueva estructura de datos para el manejo de trayectorias restringidas a redes, y tomando como base la idea de separar el problema espacio-temporal en 2 partes usada por el FNR-tree, se exploraron distintas alternativas al 1D R-tree del nivel temporal para resolver el problema de *Interval-intersection*. Lo anterior puesto que es este nivel temporal el que más memoria requiere, debido al alto nivel de dinamismo de la trayectorias comparado al bajo nivel de la red de transporte.

Estas estructuras fueron evaluadas con el fin de seleccionar aquella con mejor desempeño para formar parte de la nueva estructura para trayectorias, saliendo como mejor candidata una nueva estructura de datos compacta especializada en el problema de *Interval-intersection*, la que obtuvo los mejores niveles en uso de memoria y tiempo de consulta con los grandes datasets y competitivos para pequeños.

Se implementó entonces una nueva estructura haciendo uso de un 2D R-tree como nivel espacial y un conjunto de estas estructuras compactas como nivel temporal, resultando una estructura que, en las pruebas realizadas, requirió de hasta un 70 % menos de memoria por objeto mientras mejoró los tiempos de consulta tanto del FNR-tree como del baseline implementado en la mayoría de los casos, identificándose una debilidad con las consultas de tipo *timeslice*, pero manteniendo su ventaja en general.

Como trabajo futuro queda la búsqueda de un parámetro que indique desde qué tamaño vale la pena la construcción de una estructura de datos especializada en el nivel temporal, por sobre el uso de una búsqueda lineal, ya que, como se vio, el tiempo de construcción de la estructura propuesta es considerablemente grande, y todas estas estructuras requieren de espacio adicional que pudiese ser obviado.

Bibliografía

[1] Yariv Barkan. RTree, 2011. GitHub repository, https://github.com/nushoin/RTree.

[2] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[3] Thomas Brinkhoff. Thomas Brinkhoff: Network-based Generator of Moving Objects. https://iapg.jade-hs.de/personen/brinkhoff/generator/. [Online; accessed 9-May-2018].

[4] Nieves R. Brisaboa, Antonio Fariña, Daniil Galaktionov, and M. Andrea Rodríguez. Compact trip representation over networks. In Shunsuke Inenaga, Kunihiko Sadakane, and Tetsuya Sakai, editors, *String Processing and Information Retrieval*, pages 240–253, Cham, 2016. Springer International Publishing.

[5] Nieves R. Brisaboa, Travis Gagie, Adrián Gómez-Brandón, Gonzalo Navarro, and José R. Paramá. Efficient compression and indexing of trajectories. In Gabriele Fici, Marinella Sciortino, and Rossano Venturini, editors, *String Processing and Information Retrieval*, pages 103–115, Cham, 2017. Springer International Publishing.

[6] Nieves R. Brisaboa, Adrián Gómez-Brandón, Gonzalo Navarro, and José R. Paramá. Gract: A grammar based compressed representation of trajectories. In Shunsuke Inenaga, Kunihiko Sadakane, and Tetsuya Sakai, editors, *String Processing and Information Retrieval*, pages 218–230, Cham, 2016. Springer International Publishing.

[7] Nieves R. Brisaboa, Susana Ladra, and Gonzalo Navarro. k2-trees for compact web graph representation. In Jussi Karlgren, Jorma Tarhio, and Heikki Hyyrö, editors, *String Processing and Information Retrieval*, pages 18–30, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[8] Nieves R. Brisaboa, Miguel R. Luaces, Gonzalo Navarro, and Diego Seco. Space-efficient representations of rectangle datasets supporting orthogonal range querying. *Inf. Syst.*, 38(5):635–655, July 2013.

[9] Victor Teixeira de Almeida and Ralf Hartmut Güting. Indexing the trajectories of moving objects in networks*. *GeoInformatica*, 9(1):33–60, Mar 2005.

[10] Craig Dillabaugh. R-trees. http://cglab.ca/∼cdillaba/comp5409 _project /R_Trees.html. [Online; accessed 9-May-2018].

[11] Z. Ding, B. Yang, R. H. Güting, and Y. Li. Network-matched trajectory-based moving-object database: Models and applications. *IEEE Transactions on Intelligent Transportation Systems*, 16(4):1918–1928, Aug 2015.

[12] Elias Frentzos. Indexing objects moving on fixed networks. In Thanasis Hadzilacos, Yannis Manolopoulos, John Roddick, and Yannis Theodoridis, editors, *Advances in Spatial and Temporal Databases*, pages 289–305, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[13] Erik Garrison. intervaltree, 2011. GitHub repository, https://github.com/ekg/intervaltree.

[14] Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pages 326–337, 2014.

[15] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, June 1984.

[16] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, June 1984.

[17] Yunheng Han, Weiwei Sun, and Baihua Zheng. Compress: A comprehensive framework of trajectory compression in road networks. *ACM Trans. Database Syst.*, 42(2):11:1–11:49, May 2017.

[18] G. Jacobson. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science(FOCS)*, volume 00, pages 549–554, 10 1989.

[19] Georgios Kellaris, Nikos Pelekis, and Yannis Theodoridis. Map-matched trajectory compression. *J. Syst. Softw.*, 86(6):1566–1579, June 2013.

[20] Satoshi Koide, Yukihiro Tadokoro, Chuan Xiao, and Yoshiharu Ishikawa. CiNCT: Compression and retrieval for massive vehicular trajectories via relative movement labeling. 06 2017.

[21] Benjamin Krogh, Nikos Pelekis, Yannis Theodoridis, and Kristian Torp. Path-based queries on trajectory data. SIGSPATIAL '14, pages 341–350, New York, NY, USA, 2014. ACM.

[22] Nirvana Meratnia and Rolf A. de By. Spatiotemporal compression techniques for moving point objects. In Elisa Bertino, Stavros Christodoulakis, Dimitris Plexousakis, Vassilis Christophides, Manolis Koubarakis, Klemens Böhm, and Elena Ferrari, editors, *Advances in Database Technology - EDBT 2004*, pages 765–782, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[23] Gonzalo Navarro. *Compact Data Structures: A Practical Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2016.

[24] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1), April 2007.

[25] Daisuke Okanohara and Kunihiko Sadakane. Practical entropy-compressed rank/select dictionary. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, pages 60–70, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[26] Dieter Pfoser, Christian S. Jensen, and Yannis Theodoridis. Novel approaches in query processing for moving object trajectories. VLDB '00, pages 395–406, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[27] Michalis Potamias, Kostas Patroumpas, and Timos Sellis. Sampling trajectory streams with spatiotemporal criteria. SSDBM '06, pages 275–284, Washington, DC, USA, 2006. IEEE Computer Society.

[28] Iulian Sandu Popa, Karine Zeitouni, Vincent Oria, Dominique Barth, and Sandrine Vial. Indexing in-network trajectory flows. *The VLDB Journal*, 20(5):643, Jun 2011.

[29] Falko Schmid, Kai-Florian Richter, and Patrick Laube. Semantic trajectory compression. In Nikos Mamoulis, Thomas Seidl, Torben Bach Pedersen, Kristian Torp, and Ira Assent, editors, *Advances in Spatial and Temporal Databases*, pages 411–416, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[30] Jens M. Schmidt. Interval stabbing problems in small integer ranges. In Yingfei Dong, Ding-Zhu Du, and Oscar Ibarra, editors, *Algorithms and Computation*, pages 163–172, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[31] Jens M. Schmidt. Publications by Jens M. Schmidt. `http://www4.tu-ilmenau.de/combinatorial-optimization/ShowPub.html`, 2018. [Online; accessed 1-May-2018].

[32] Yufei Tao and Dimitris Papadias. MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. VLDB '01, pages 431–440, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.