UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
Y CIENCIAS DE LA COMPUTACIÓN

REPRESENTACIÓN EFICIENTE DE SECUENCIAS ESPACIALES Y ESPACIO-TEMPORALES

Por: Carlos Quijada Fuentes

Tesis presentada a la Facultad de Ingeniería de la Universidad de Concepción
para optar al grado académico de Doctor en Ciencias de la Computación

Agosto 2025
Concepción, Chile

Profesora Guía: María Andrea Rodríguez Tastets
Profesor Guía: Diego Seco Naveiras

A mi hija Antonia, y también a Susy†, a Chele† y a Moncho†.

AGRADECIMIENTOS

Resulta difícil el recordar a cada persona que mostró su apoyo durante la confección de un trabajo que conlleva tanto tiempo finalizar. Quisiera mencionar al menos a quienes sin lugar a dudas merecen un espacio en esta página. En primer lugar, y antes que todos, agradecer a mi familia: mis padres, mis hermanas, mi hermano, mi hija, y Daniela. Sin su apoyo incondicional no lo habría logrado.

En segundo lugar, a quienes compartieron más de cerca el largo trabajo, Dr. Diego Seco y Dra. M. Andrea Rodríguez. Ambos con una paciencia enorme, y una tremenda disposición en tiempo y enfoque para llevar a puerto este barco que a ratos fondeaba.

Siguen en la lista las amistades cercanas, tanto las que hice en el programa como las que me acompañaron en los momentos de distensión: Ignacio, Mabel, Nacho, Guille, Vale, Mauri, Pablo, Jean, Xino, Yerko, Pame, Jaime, Miguel P. y Adrián.

Finalmente quiero agradecer a algunos docentes, aquellos del programa de doctorado con quienes volví a sentir el placer de estar en un aula como alumno, y por último a mi mentor en la etapa previa, Dr. Gilberto Gutiérrez quien me motivó, y ayudó, a continuar los estudios.

Los estudios de doctorado fueron financiados por la Agencia Nacional de Investigación y Desarrollo (ANID) por medio de la beca de Doctorado Nacional CONICYT-PFCHA/Doctorado Nacional/2020-21200101.

Resumen

El volúmen actual de información espacial y espacio-temporal que se genera se ha visto incrementado por la masividad de dispositivos que permiten registros constantes, ya sea para determinar la ubicación de determinados objetos en movimiento, o para obtener mediciones sobre un espacio o punto de interés determinado. Estos pequeños dispositivos actualmente se encargan de recopilar la información, que luego debe ser pre-procesada, para limpiar sus datos, corregirla en ciertos casos, para finalmente almacenar y responder consultas sobre la información. Este trabajo de tesis se orienta en desarrollar propuestas novedosas para el registro de secuencias espaciales, y secuencias espacio-temporales que permitan almacenar de manera eficiente los datos, y responder consultas en tiempo competitivo.

Este compendio contiene el desarrollo de dos líneas de trabajo enfocadas en datos espacio-temporales. La primera propuesta es una estructura de datos compacta para la representación eficiente de trayectorias sobre una representación en red como caminos, con un fuerte enfoque en responder consultas de relaciones topológicas en los datos contenidos, permitiendo responder todas aquellas secuencias que cumplen una determinada relación. Esta estructura resulta en una representación eficiente en términos de espacio, y por sobre todo eficiente en los tiempos de respuesta para la mayoría de las operaciones implementadas. La segunda investigación corresponde a dos estructuras de datos compactas para la representación de series de tiempo relacionadas a una ubicación espacial. La formulación de la estructura estuvo guiada por índices de autocorrelación de los conjuntos de datos, estableciendo dos situaciones diferentes según las características de los datos. Esta representación resulta especialmente eficiente en grillas de datos espacio-temporales densas, y se puede identificar una diferencia entre los valores de autocorrelación entre los conjuntos utilizados que permiten decidir el uso de la estructura, o de su variante.

Keywords – Árbol de sufijos generalizado, Estructuras de datos compactas, datos espacio-temporales, series de tiempo

Abstract

The current volume of spatial and spatio-temporal information being generated has increased significantly due to the widespread availability of devices that enable continuous recording—whether for determining the location of moving objects or for collecting measurements over a space or specific point of interest. These small devices are now responsible for gathering the data, which must then be preprocessed to clean and, in some cases, correct the information, before it can be stored and queried efficiently. This thesis focuses on developing novel proposals for the representation of spatial sequences and spatio-temporal sequences that enable efficient data storage and competitive query performance.

This compendium presents the development of two research directions centered on spatio-temporal data. The first contribution is a compact data structure for the efficient representation of trajectories over a network-based model, such as roads, with a strong emphasis on answering topological relation queries. It supports retrieving all sequences that satisfy a given relation. This structure achieves an efficient space representation and, most notably, fast query times for most of the implemented operations. The second research contribution involves two compact data structures for the representation of time series associated with spatial locations. The formulation of these structures was guided by autocorrelation indices derived from the datasets, leading to two different approaches depending on the data characteristics. This representation proves particularly effective for dense spatio-temporal grids, and a clear distinction in autocorrelation values among the datasets enables a well-founded decision between the proposed structure and its variant.

Keywords – Generalized suffix tree, compact data structures, spatio-temporal data, time series

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación

El movimiento es una acción que tiene una gran importancia en nuestro mundo. Algunos de los ejemplos son los desplazamientos que realizan los animales en diferentes horas del día al salir de su guarida en busca de bebida o alimento, sus migraciones en distintas estaciones del año al buscar mejores condiciones. De la misma forma las plantas se esparcen por un territorio con sus semillas llevadas por medio del viento, por las aves que comen sus frutos, y por los flujos de agua. Por supuesto el movimiento está presente en diferentes escalas, como los astros moviéndose por grandes distancias en distintas ubicaciones dependiendo del sistema, galaxia o cúmulo en el que se encuentren. Incluso a un nivel microscópico existen movimientos que resultan de gran interés para la investigación. No hace falta mencionar que para los seres humanos, los desplazamientos son algo que está intrínsecamente relacionado desde sus primeras migraciones como homínidos hacia nuestra actual especie. Tanto es así que tenemos guías respecto de los movimientos que con mayor frecuencia realizamos, representados en mapas con diversos contextos: carreteras entre ciudades, calles dentro de ciudades, rutas de transporte público interurbano, rutas de vuelos, la red de metro, las líneas férreas, etc.

Para nuestra ciencia actual ha sido relevante descubrir los desplazamientos realizados por las especies previas para entender el cómo estos describen parte de nuestra evolución. Actualmente ya no existe la necesidad de “buscar” nuestros

desplazamientos, debido a que la mayoría de las personas porta dispositivos con los que se registran en detalle estos movimientos gracias al gran desarrollo en la tecnología GPS y otros sensores. De la misma forma, el desarrollo tecnológico en el área de sensores ha permitido realizar mediciones regulares con pequeños dispositivos que permiten registrar la evolución de distintos parámetros en el punto en el que se encuetran instalados. Esto genera una gran cantidad de información que puede ser aprovechada para distintos fines y estudios, con distinta granularidad, diferentes restricciones dependiendo de su contexto, distinta información que complementa o enriquece los registros base, etc.

Gracias a las nuevas tecnologías es posible capturar múltiples variables y su cambio a lo largo del tiempo, como el nivel de ruido ambiental, la intensidad del tráfico en un punto determinado, las variaciones de temperatura en un territorio, o la actividad eléctrica de las zonas del cerebro. Todos estos registros corresponden a datos espacio-temporales, y comparten una característica fundamental: evolucionan en el tiempo y presentan una dimensión asociada al espacio, ya sea una coordenada geográfica, un sensor ubicado en una región o una celda de una grilla regular.

Con el volumen de información que se produce desde los numerosos dispositivos que la generan, cobra vital importancia el almacenar dichos registros de manera eficiente, problema que ya ha sido ampliamente abordado dando lugar a muchas técnicas de compresión de datos. Sin embargo, no sólo es importante el que su almacenamiento sea eficiente en términos de espacio, sino que también se permita un manejo eficiente de las consultas sobre la información. Aquí entran en juego las estructuras de datos compactas, que permiten almacenar la misma información que otras estructuras de datos en un espacio mucho menor, permitiendo que se consulten porciones de ésta sin necesidad de desempaquetarla completamente. Esto permite que grandes cantidades de información estén disponibles en dispositivos de menor capacidad, almacenar mayor cantidad de información en el mismo espacio, e incluso mejorar el procesamiento de la información por la mayor cantidad de datos en los niveles altos de la jerarquía de memoria de una máquina.

En este trabajo se exploran dos enfoques distintos pero relacionados para abordar el problema de la representación eficiente de datos espacio-temporales. El primero de ellos se centra en el estudio de las relaciones topológicas entre trayectorias representadas sobre una red. Las relaciones topológicas son útiles en el contexto de integración de datos provenientes de distintas fuentes o con distintos modelos de

representación, donde pueden existir inconsistencias debido a errores o diferencias en los mecanismos de captura, y donde detectar relaciones entre ellas puede ser clave para alinear o complementar la información. Se presenta una propuesta de estructura basada en árboles de sufijos generalizados que permite representar múltiples trayectorias sobre una red de movimiento, detectar relaciones entre ellas y responder consultas de manera eficiente.

El segundo enfoque nace a partir de la observación de que no todos los datos espacio-temporales se presentan como trayectorias sobre redes. Existen numerosos conjuntos de datos que corresponden a secuencias de valores, donde cada uno representa una observación en un punto en el tiempo y espacio. En este caso, la eficiencia de representación se puede mejorar si se considera el grado de dependencia o relación que existe entre los datos, tanto en el tiempo como en el espacio. En este trabajo se estudian distintos conjuntos de datos reales, se analiza su nivel de correlación temporal y espacio-temporal, y se evalúa cómo esta correlación puede ser aprovechada para reducir el tamaño de representación usando estructuras de datos compactas especializadas.

Ambas líneas de trabajo tienen como motivación el diseño e implementación de estructuras eficientes para representar datos espacio-temporales, sin perder la capacidad de realizar consultas relevantes sobre los datos.

1.2. Contribución

Todo trabajo desarrollado de manera estructurada tiene resultados interesantes que se deben compartir con la comunidad. Esta tesis centra sus aportes en el área de datos espacio-temporales, tanto en el ámbito de secuencias que describen ubicaciones espaciales en el tiempo como también en el análisis y representación de secuencias de mediciones en el tiempo asociadas a ubicaciones espaciales fijas. A continuación se mencionan las principales contribuciones de esta tesis, contribuciones que se reportan dos artículos científicos, uno publicado [109], y el otro enviado a revista [110]:

- Estudio de las relaciones topológicas de trayectorias, con énfasis en aquellas que se pueden observar sobre redes de movimiento. Se identificaron relaciones relevantes para la consulta de trayectorias y se establecieron condiciones formales para su definición y detección eficiente.

- Propuesta de una estructura de datos para la representación de caminos sobre una red y la consulta de relaciones topológicas binarias, considerando eficiencia tanto en almacenamiento como en tiempo de consulta. Esta estructura permite representar múltiples trayectorias y verificar relaciones como la intersección, inclusión y disyunción de manera directa entre dos trayectorias del conjunto.
- Diseño e implementación de la estructura *TRGST* (Topological Relations Generalized Suffix Tree), basada en el árbol de sufijos generalizado, adaptado para representar trayectorias restringidas a una red y consultar relaciones topológicas sobre el conjunto completo de trayectorias almacenadas. Esta estructura permite responder eficientemente consultas topológicas para la recuperación de un subconjunto de trayectorias que cumple alguna relación topológica con alguna de las trayectorias del conjunto representado.
- Caracterización de conjuntos de datos espacio-temporales compuestos por secuencias numéricas asociadas a ubicaciones espaciales. Se analiza la auto-correlación temporal y espacio-temporal en tres conjuntos de datos; registros de atributos climáticos de un territorio, exámenes de electroencéfalogramas, y sensores de tráfico de una autopista de Madrid. Este análisis permite identificar propiedades para guiar estrategias de representación eficiente.
- Propuesta de dos estructuras de datos compactas para la representación de secuencias espacio-temporales: una orientada a datos densos dispuestos en grilla regular, y otra orientada a datos más esparsos o distribuidos de forma irregular. Ambas estructuras fueron evaluadas en términos de eficiencia de almacenamiento y su comportamiento fue relacionado con las propiedades estadísticas del conjunto de datos.

1.3. Estructura de la Tesis

Este documento de tesis tiene un enfoque de compendio, es decir que su contenido central corresponde a los dos artículos de revista escritos durante el desarrollo del programa de doctorado [1]. En el Capítulo 2 se desarrollan conceptos previos que guían la comprensión del trabajo que aquí se presenta, comenzando por

[1] De acuerdo al reglamento del programa , Artículo 13, "Opcionalmente, el documento de tesis de grado podrá estar constituido por un compendio de (al menos dos) publicaciones científicas en inglés, agregando una descripción inicial y conclusiones finales en español".

la base de estrucutras de datos compactas, luego la definición de modelos de representación de datos espaciales, siguiendo con la evolución natural de estos datos hacia trayectorias (donde se incorpora la dimensión temporal), donde se analizan además las relaciones topológicas entre trayectorias, y entre segmentos de línea, y finalizando con algunos ejemplos de representación eficiente de trayectorias. Los desplazamientos espaciales pueden ser representados como segmentos de línea, por lo que el interés en esta etapa fue formar un conocimiento sobre cuáles eran las diferentes relaciones topológicas definidas en la literatura. En el Capítulo 3 se presenta la estructura de datos *TRGST*, estructura que extiende del Árbol de Sufijos Generalizado, y fue diseñada para la representación de un conjunto de trayectorias representadas como caminos. Ahí también se presentan algoritmos utilizados para dar respuesta a consultas sobre relaciones topológicas. En el Capítulo 4 se presenta el artículo desarrollado sobre secuencias temporales de datos asociadas a una ubicación en el espacio. Este artículo muestra la caracterización de los datos, y las dos estructuras presentadas para las distintas categorías de tipos de datos que se identifican, junto con los resultados de su experimentación. Para finalizar el Capítulo 6 desarrolla la discusión de las contribuciones del trabajo y algunas líneas posibles de investigación que se desprenden del trabajo realizado.

Capítulo 2

Conceptos Previos

Este trabajo se enfoca en la integración de dos grandes conceptos: datos espacio-temporales y estructuras de datos compactas. En ambos casos existen conceptos previos importantes de conocer para facilitar la comprensión de las propuestas que se realizan en este documento. Aún cuando esta tesis corresponde a un compendio de artículos, se desarrollan aquí algunas secciones para complementar la información del trabajo en formato de artículos. Principalmente se busca complementar los antecedentes del trabajo descrito en el Capítulo 3, desarrollando en más detalles conceptos asociados a datos espacio-temporales y estructuras de datos compactas.

2.1. Estructuras de Datos Compactas

La gran cantidad de información que existe sobre trayectorias de diferente índole magnifica la necesidad que la información se maneje de modo eficiente tanto en espacio como en tiempo de procesamiento. Esto se vuelve relevante tanto para almacenar los datos como también para la recuperación de los mismos.

En este escenario resultan de particular interés las Estructuras de Datos Compactas (EDC), que constituyen un tipo de estructuras de datos que almacenan información en un espacio menor al utilizado por una representación común. Por ejemplo, los bitmaps que se explican en la subsección siguiente utilizan $n + o(n)$ bits para representar n bits. A diferencia de lo que sucede al comprimir los datos, donde es necesario descomprimir para acceder a la información, las EDC permiten realizar

consultas de interés sobre la información que contienen de manera eficiente sin la necesidad de descomprimir los datos. En general se dice que una EDC usa $O(OPT)$ bits de espacio donde OPT es el número óptimo de bits necesarios para almacenar los datos según la teoría de la información propuesta por Shannon [121]. Otra de las ventajas de una EDC, además del ahorro de espacio para la representación de la información, es que permite introducir más información en los niveles más rápidos de la jerarquía de memoria.

A continuación se describen dos estructuras de datos, Bitmap, o arreglos de bits, y el Compressed Suffix Tree. La primera de estas estructuras es ampliamente conocida como un estructura de datos compacta que a su vez se utilizar como un componente para el diseño de otras estructuras de datos compactas. La segunda estructura de datos es de especial interés para facilitar la comprensión de la propuesta descrita en el Capítulo 3.

2.1.1. Bitmap o Bitvectors

Un bitmap es una EDC ampliamente utilizada, y parte fundamental de otras EDC. Esta estructura corresponde a una secuencia de valores 0,1 que permite responder las consultas: *access*, *rank*, y *select*. Para dar respuesta eficiente a estas operaciones requiere de estructuras de datos adicionales. Las consultas de *rank* y *select* permiten responder cuántos bits en 1 existen desde el inicio hasta la i-ésima posición y cuál es la posición del i-ésimo bit en 1, respectivamente. También existe, para ambas operaciones, las mismas consultas sobre los valores 0 en la búsqueda. La consulta *access* permite recuperar el valor del i-ésimo bit en la secuencia. En la Figura 2.1.1 se puede ver un ejemplo de un bitmap y de sus operaciones.

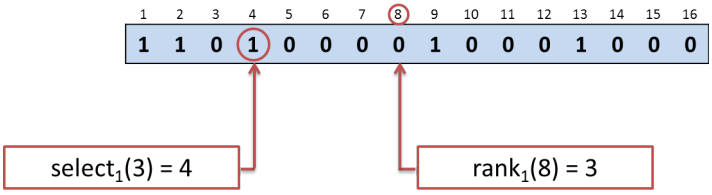


Figura 2.1.1: Ejemplo de bitmap y sus operaciones.

Para responder las consultas de *rank* y *select* en tiempo constante se sigue la siguiente idea base: se segmenta el arreglo de bits en trozos de tamaño constante, y por cada trozo se almacena la suma de bits en 1 que contiene hasta su última casilla. Por ejemplo, si el bitmap de la Figura 2.1.1 se divide en segmentos de 4

elementos, se requerirá de un arreglo adicional que almacene 4 valores, los cuales serían 3, 3, 4 y 5.

Una implementación en $n + o(n)$ bits y respuesta en tiempo constante considera un esquema jerárquico donde el primer nivel son superbloques y cada uno se divide en bloques de menor tamaño. Los superbloques poseen la suma acumulada de valores en 1 al inicio y luego para los bloques se guarda la cantidad de unos al inicio pero relativos al superbloque. Luego para determinar los elementos en el bloque se utiliza la instrucción conocida como *popcount* que gracias a tablas precomputadas permite obtener la cantidad de bits en 1 en una palabra. De esta manera las operaciones de *rank* se puede responder en tiempo constante. Para la operacion *select* existe diversas propuestas que permiten responder en tiempo constante [26, 99, 134].

Un ejemplo de implementación de esta estructura base, y varias otras, es la realizada por Gog *et al.* [51], publicada en GitHub[1]. Esta implementación de un bitmap (o bitvector) es la base para muchas de las otras estructuras de datos disponibles en el mismo proyecto.

2.1.2. Compressed Suffix Tree

Un Suffix-Tree (`ST`) corresponde a una estructura de datos que permite indexar texto para responder eficientemente consultas sobre secuencias comunes o coincidencias en sus elementos. Un `ST` se construye a partir de todos los sufijos del texto en cuestión. Cada sufijo se inserta en el árbol mediante el recorrido de sus caracteres, insertando por cada carácter un nodo en el árbol desde la raíz. Si el primer carácter no está relacionado a ninguno de los arcos del nodo raíz, se crea un nuevo nodo cuyo arco desde la raíz se relaciona con el primer carácter y se mueve hacia el nuevo nodo para continuar con los demás caracteres. Si el carácter existe en el nodo raíz, se sigue el enlace y se continúa con los demás caracteres. Una vez que se han procesado todos los caracteres se genera el nodo hoja asociando en su arco el carácter de fin de la secuencia. El `ST` es un árbol cardinal, por lo que se cumple para todos los nodos que no existen arcos repetidos, y los arcos de cada nodo se ordenan según el símbolo que representan.

El árbol de sufijos comprimido (`CST`) es una propuesta de implementación

[1] https://github.com/simongog/sdsl-lite

que se divide, de manera general, en 3 componentes: un arreglo de sufijos para almacenar el orden lexicográfico de todos los sufijos del texto, un arreglo para almacenar los prefijos comunes más largos de aquellos sufijos que son contiguos lexicográficamente, y algunas estructuras adicionales para operaciones de navegación sobre la estructura [98].

Algunas de las operaciones soportadas se mencionan a continuación (considere que un nodo del `CST` representa un texto):

- **LCA**: Lowest Common Ancestor, que permite recuperar el nodo común más alejado a la raíz entre dos nodos dados.
- **slink** de un nodo: corresponde al nodo que representa la misma cadena del nodo entregado, eliminando el primer carácter.
- **wlink** de un nodo dado un carácter: devuelve el nodo que representa la concatenación del carácter y el texto del nodo. El nodo de la concatenación podría no estar presente en la estructura.

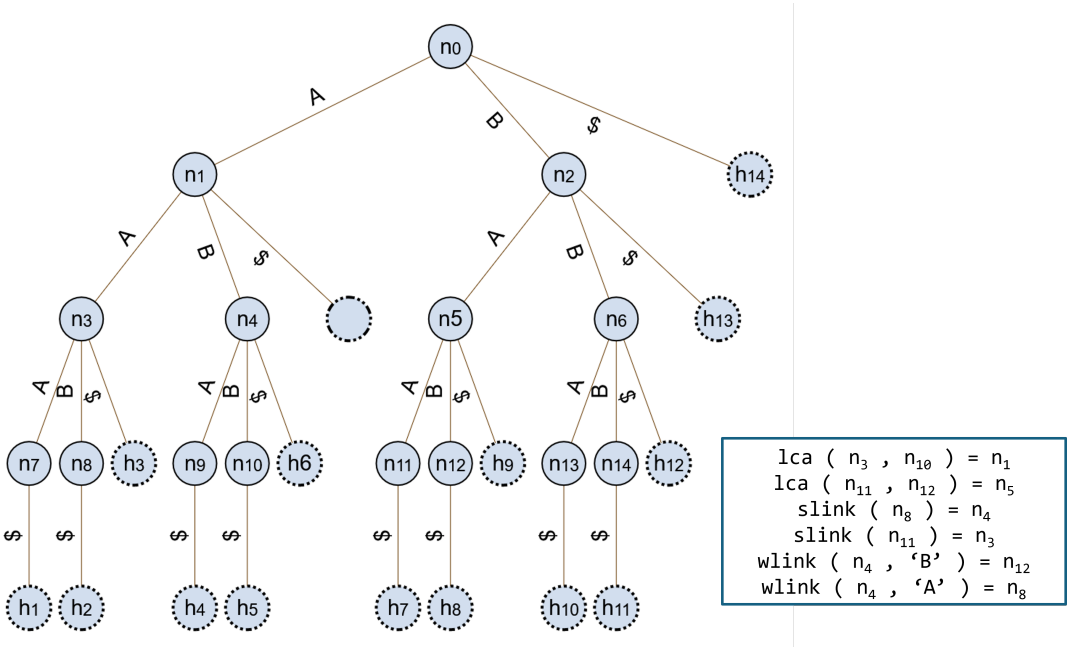


Figura 2.1.2: Esquema de árbol de sufijos generalizado y ejemplos de consulta sobre la estructura.

Las operaciones anteriores se ejemplifican en la Figura 2.1.2, donde se muestra el esquema de un árbol de sufijos construido con todas las combinaciones posibles de texto de 3 carácteres con un alfabeto $\{A, B, \$\}$. Por ejemplo, la operación **LCA** entre los textos AA (nodo n_3) y ABB (nodo n_{10}) corresponde al texto A

(nodo n_1). En el caso de **slink** con el texto BAA (nodo n_{11}), el resultado es AA (nodo n_3). Terminando con los ejemplos, en la operación **wlink** desde el texto AB (nodo n_4) con el carácter B, se obtiene el nodo n_{12} que representa el texto BAB (recordar que esta operación está sujeta a que exista nodo que representa el texto resultante).

2.2. Representación y Procesamiento de Datos Espacio-Temporales

Los datos espaciales son aquellos vinculados a ubicaciones geográficas específicas, y suelen representarse en una o más dimensiones. Comúnmente se proyectan en dos dimensiones, como en mapas digitales o modelos territoriales. Estos datos permiten representar diferentes fenómenos, temperaturas en un territorio, altitudes, densidad de vegetación, hidrografía, ubicaciones de interés, etc. Existen dos modelos ampliamente utilizados para su representación: el modelo ráster, y el modelo vectorial [103].

El modelo ráster se caracteriza por realizar divisiones del terreno sobre el que se va a representar la información, donde la unidad de información está asociada a una de estas divisiones. usualmente de forma regular como cuadrados, aunque también pueden usarse otras formas geométricas como hexágonos o triángulos. A cada celda se le asocia un valor que representa una propiedad, como puede ser temperatura promedio, elevación, nivel de vegetación, uso de suelo, etc.

El modelo de representación vectorial utiliza como elementos base puntos y líneas, a partir de los cuales se pueden formar polígonos más complejos. Esta representación permite describir con mayor precisión objetos espaciales como carreteras, ríos, límites administrativos o edificaciones. Su estructura no depende de una grilla fija, lo que le permite representar formas irregulares o trayectorias de movimiento con mayor detalle, sin depender de la granularidad de una grilla.

Una extensión natural de estos modelos surge al incorporar la dimensión temporal, dando lugar a los datos espacio-temporales. Un caso relevante es el de las trayectorias, que representan el movimiento de un objeto a través del espacio a lo largo del tiempo. Este movimiento en la realidad es continuo a través del tiempo y del espacio, sin embargo los registros que se realizan mediante instrumentos

no lo son. Además de la ubicación, estos registros permiten calcular propiedades como la velocidad, dirección, área cubierta, distancia total recorrida y duración del movimiento [9].

La trayectoria que sigue un objeto se puede representar como una secuencia de las ubicaciones espacio-temporales del mismo, es decir, como pares de instantes de tiempo y ubicación espacial [102]. Un ejemplo de trayectoria $T = (p_1, t_1), (p_2, t_2), ..., (p_n, t_n)$ con n el número de registros define el desplazamiento de un objeto en movimiento, donde se cumple que $t_i < t_j$ para cualquier $1 \leq i < j \leq n$. Se tiene que p_i corresponde a registros en la dimensión espacial y t_i son los registros en la dimensión temporal.

Existen dos formas principales de representar los datos de movimientos: sobre un espacio libre en el que los objetos de interés pueden estar ubicados en cualquiera de las posibles locaciones en un rango de valores continuos y otra representación donde se restringe la posible ubicación de los objetos de interés a una red representada por un grafo.

2.2.1. Modelos para la Representación de Trayectorias

Las trayectorias resultan interesantes en distintos ámbitos de estudio, por lo que es relevante la forma en la que se pueden representar. Existen distintos enfoques que han evolucionado en el tiempo tomando porciones de propuestas previas y agregando elementos novedosos para mejorar su representación según el contexto, el rendimiento del procesamiento o la flexibilidad para la representación de ciertas restricciones que existen entre los elementos a almacenar. Esta sección se enfoca en la descripción de aquellas propuestas sobre los modelos de representación para trayectorias.

2.2.1.1. Representación en Espacio Libre

La representación de una trayectoria sobre el modelo de espacio libre permite que la ubicación asociada a un objeto en un instante de tiempo esté en cualquier punto del espacio. Como ya se mencionó antes, los dispositivos de medición realizan muestras a intervalos de tiempo, por lo que una trayectoria se puede definir por una secuencia de caminos ordenada en el tiempo. Un camino se define por dos puntos, uno de inicio y otro de fin, los cuales definen el recorrido del objeto en ese intervalo

de tiempo. Así, la secuencia de caminos que describe la trayectoria de un objeto en movimiento tiene como restricción que el punto final de un camino coincide con el punto inicial del siguiente en la secuencia. La Figura 2.2.1 muestra una trayectoria donde el desplazamiento espacial en dos dimensiones es representado en los ejes x e y y el componente temporal se representa en el eje t (vertical). En esta representación los registros de trayectorias se dan por una secuencia de pares de ubicación y tiempo. Recuperar la ubicación de un elemento en un instante de tiempo cuando no hay un registro en el mismo instante se puede hacer utilizando la ubicación del instante de tiempo anterior y del siguiente. A partir de estos dos registros se puede asumir que el objeto en movimiento siguió un recorrido recto entre los dos puntos a una velocidad constante y así determinar por interpolación una aproximación de la ubicación en el instante consultado [132].

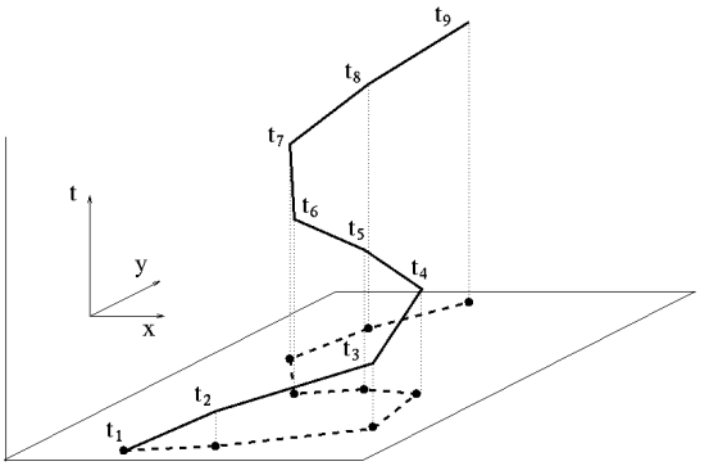


Figura 2.2.1: Ejemplo de trayectoria en espacio libre [104].

Adicional a la representación en espacio libre tradicional, existen algunas propuestas que agregan características especiales que pueden resultar útiles en ciertos contextos. Por ejemplo, si el objeto en movimiento no se mueve directamente desde un punto de la muestra al siguiente no existe certeza de su ubicación en la ventana de tiempo intermedio. Aquí resulta interesante la propuesta de incorporar incertidumbre, y en este contexto responder consultas sobre áreas en las que es seguro que el objeto está dentro, o áreas en las que hay un grado de incertidumbre [128, 71].

Otra alternativa para manejar representaciones de objetos en movimiento en un espacio libre es transformar el espacio de dos dimensiones en un espacio de una

dimensión mediante una curva de llenado (como puede ser z-order, el valor Hilbert[2] u otra), luego se procesan las trayectorias para representar sus ubicaciones en el espacio transformado de una dimensión [104]. La desventaja de esto es el costo en el procesamiento en sistemas de bases de datos comerciales tradicional, donde existen algunas alternativas de pre-procesamiento para volver más eficiente el almacenamiento y consulta de los datos espacio-temporales [9].

Finalmente, mencionar que existen diversas propuestas de modelos de representación de trayectorias en espacio libre que abordan distintas dificultades; optimizar los tiempos de respuesta y el espacio requeridos para explorar distintos niveles de detalle al momento de desplegar la información [35], representar el movimiento de los objetos a través de ciertas regiones [88, 90], o con enfoque en ciertos puntos de interés del espacio de movimiento [2, 6].

2.2.1.2. Representación en Red

Dado que los datos de trayectoria en un modelo de espacio libre vuelven compleja la implementación de las operaciones, resulta necesario contar con otra alternativa para su representación. Aquí surge el modelo de representación de trayectorias sobre una red de movimiento, donde se restringe las posibilidades de movimiento de un objeto desde un espacio libre hacia una red de caminos por los cuales se podría desplazar. Esta limitación se puede basar en el contexto o en el objetivo de la aplicación. La restriccción en la dimensión espacial permite reducir el espacio requerido para la representación de los datos, y también reduce las posibles operaciones, por lo que vuelve más eficiente el soporte para indexación y procesamiento de consultas [56] que en el espacio libre.

Un ejemplo de mapeo de trayectorias se muestra en la Figura 2.2.2, donde movimientos en un espacio de dos dimensiones que siguen un camino se pueden mapear a una red de nodos de una dimensión. En esta representación en red la componente espacial se almacena en referencia a los nodos, pudiendo definirse en base a las trayectorias del conjunto, a puntos de interés en el espacio libre, puntos de intersección en un sistema de calles, celdas del espacio libre, u otros. Luego las ubicaciones espaciales se registran en base al recorrido por los arcos de la red en un intervalo de tiempo, o directamente en base a los nodos. Para conocer

[2]El valor Hilbert corresponde a una generalización del espacio Euclideano, que en este caso se calcula en el punto central de cada arco de la red.

la ubicación en un instante de tiempo distinto de los registrados se puede hacer por interpolación de las ubicaciones en los instantes inmediatamente siguiente y anterior a la consulta, de manera similar a como se explicó para las representaciones en espacio libre.

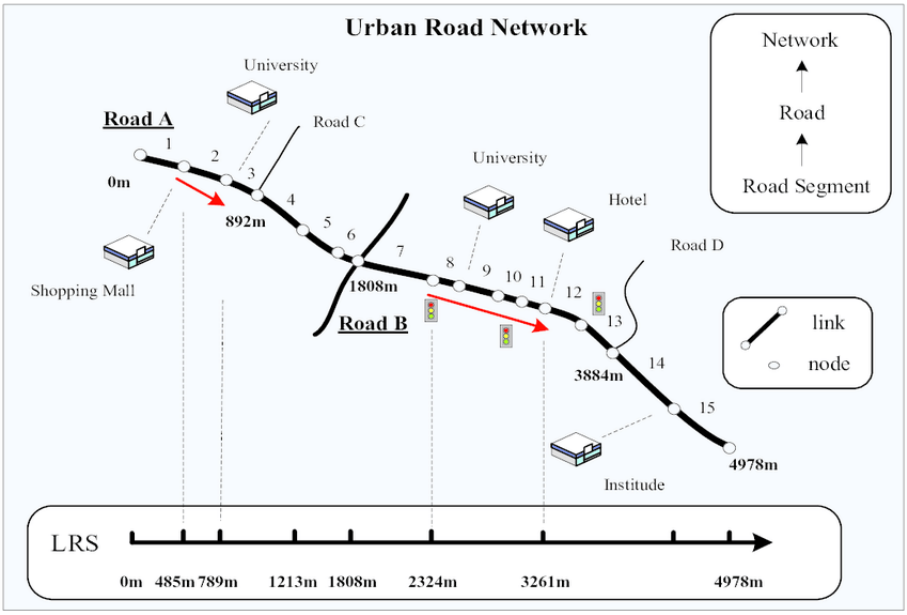


Figura 2.2.2: Ejemplo de mapeo a una trayectoria en red [138]

Otro enfoque para reducir el espacio de almacenamiento de trayectorias sobre una red propone modificar la red mediante la reducción del número de nodos, luego se actualizan los caminos en base a las modificaciones de la red, obteniendo caminos más cortos [64]. El objetivo es mantener el mayor grado de similitud con la estructura original de la red, lo que se evalúa mediante una función de similitud [24, 23]. Una vez mapeada una red de transporte es posible utilizar estructuras para el procesamiento de ubicaciones y así conocer ubicaciones pasadas, actuales y estimar ubicaciones futuras [34].

2.2.2. Relaciones sobre trayectorias

En esta sección se describen algunas propuestas que abordan la forma en que se relacionan las trayectorias entre sí o con otros objetos espaciales o regiones. Es importante destacar que las trayectorias consideran dos elementos clave en su definición: una componente temporal que representa el intervalo de tiempo del movimiento asociado y una componente espacial que corresponde a la ubicación del objeto en movimiento.

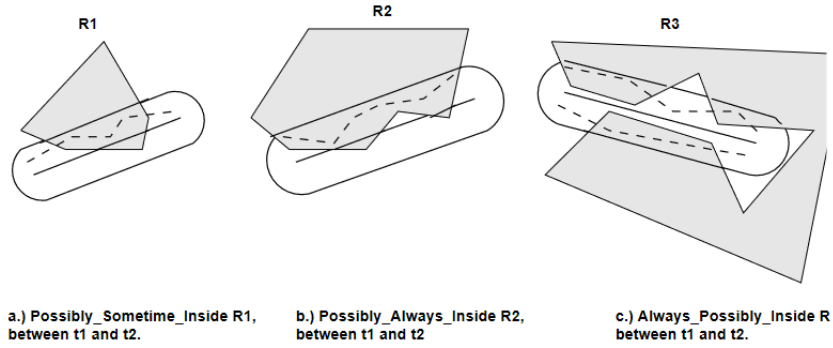
Existen relaciones entre los objetos espaciales que también se aplican y resultan interesantes en el ámbito de las trayectorias. Una de las aplicaciones donde se aprovechan las relaciones entre trayectorias corresponden a técnicas de clustering que permiten identificar patrones o similitud entre trayectorias de diversas formas: a diferentes niveles de detalle en la dimensión espacial [35], grupos en base a su densidad [93], división en sub-trayectorias [54], caminos más densos [65], según cantidad de vecinos [80] e incluso desde fuentes de video [111, 135].

Una de las posibles relaciones que existen para trayectorias permiten vincular una trayectoria con un espacio o área determinada. Por ejemplo, sobre la ubicación de un elemento respecto de una región es posible responder, con un cierto grado de incertidumbre definida, la posible ubicación de la trayectoria en términos de: *possiblySometimeInside*, *sometimePossiblyInside*, *possiblyAlwaysInside*, *alwaysPossiblyInside*, *alwaysDefinitelyInside*, *definitelySometimInside* y *sometimeDefinitelyInside* [128]. La Figura 2.2.3 muestra ejemplos de los predicados, donde t_1 y t_2 los instantes que definen un intervalo de tiempo, la zona gris representa el área sobre la que se consulta la relación, la línea contínua representa la trayectoria y su límite de incertidumbre, y la línea discontinua representa una curva que podría satisfacer el predicado. Similarmente se puede indentificar relaciones entre las trayectorias y su interacción con una región del tipo: *stayWithin*, *bypass*, *leave*, *enter* y *cross* [9].

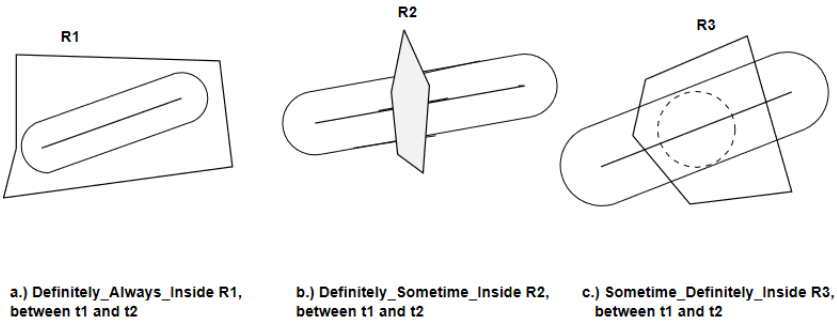
Respecto de relaciones entre trayectorias existen algunas que se basan en relaciones topológicas como: *intersect*, *meet*, *equal*, *near* y *far* (ver Figura 2.2.4). Es importante considerar que el contexto influye sobre las últimas dos relaciones. La escala de los objetos representados hará que en algunos casos algo cercano sea considerado lejano, como por ejemplo en el movimiento entre peatones la cercanía entre ellos es completamente distinta a lo que se esperaría cercano entre dos aviones.

Las relaciones anteriores corresponden a una manera sencilla de describir cómo se comportan, pero carecen de detalles que permitan identificar más concretamente por ejemplo, en qué momentos es que se produce alguna intersección entre las trayectorias, omitiendo completamente la dimensión temporal.

Una forma mucho más detallada de describir las relaciones topológicas, que sí considera la dimensión temporal, corresponde a QTC_c (Qualitative Trajectory



(a) possible operations



(b) definite positions

Figura 2.2.3: Ejemplos de relaciones en [128]

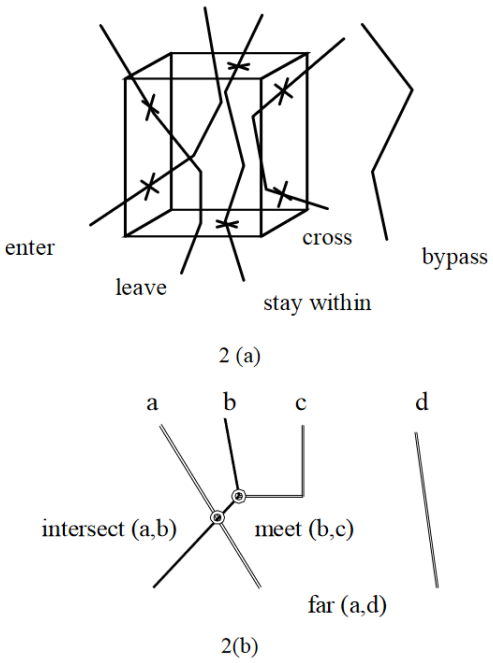


Figura 2.2.4: Ejemplos de relaciones en trayectorias [9]

Calculus Double-Cross), donde se utilizan tablas de composición para generar 81 relaciones que describen los posibles movimientos entre 2 objetos [130]. Es

importante destacar que en este caso las relaciones reflejan el movimiento que hay de un elemento respecto del otro [129].

Una tabla de composición codifica todas las posibles composiciones que existen entre relaciones, es posible representar estas relaciones válidas en reemplazo de las demostraciones por teoremas complejos [133]. Este trabajo detalla reglas que se pueden aplicar para generar de manera automática una tabla de composición entre las relaciones de QTC_c. Esto debido a que existen 81 relaciones posibles (ver todos los iconos en los Anexos), dando un total de 6.561 (81^2) composiciones. Las representaciones de la Figura 2.2.5 corresponden a 3 de los movimientos posibles entre dos objetos definidos en el modelo QTC_c. Cada objeto es representado por un punto, donde un punto vacío indica que si existe movimiento, y un punto pintado indica que no hay movimiento del objeto representado (como muestra la relación 44 de la imagen). Las líneas punteadas que acompañan los puntos blancos indican la dirección del posible movimiento (como el ejemplo de la relación 16 y 44), y en el caso de los cuartos de círculo corresponde al área a la que se dirige el movimiento del objeto (como muestra el esquema de la relación 1 y 16). Los movimientos se dan en un espacio de dos dimensiones, y se describen por el código que se muestra en la parte inferior de cada movimiento de los ejemplos en la Figura 2.2.5. Los códigos se definen por medio de los signos $+, -$ y 0, que indican las posibles variaciones en las siguientes 4 condiciones:

- Movimiento del primer objeto respecto del segundo objeto: $-$ el objeto se aproxima, $+$ cuando el objeto se aleja, y 0 indica que el objeto no cambia su posición.
- Movimiento del segundo objeto respecto del primer objeto: los mismos casos anteriores.
- La dirección del movimiento del primer objeto en referencia al segundo: $-$ el objeto se mueve hacia la izquierda del segundo objeto, $+$ cuando el primer objeto se mueve hacia la derecha del segundo objeto, 0 cuando no hay orientación en el movimiento del primer objeto.
- La dirección del movimiento del segundo objeto en referencia al primero: los mismos casos anteriores.

Las ventajas de poseer una buena definición de relaciones entre trayectorias, es que éstas relaciones permiten la definición de restricciones de integridad semánticas de los elementos almacenados en un modelo relacional extendido [10]. En esta

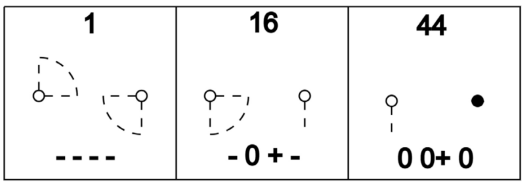


Figura 2.2.5: Ejemplos de relaciones definidas en QTC_c [130]

propuesta se describen Restricciones de Integridad Semántica Espacial (SSIR) separadas en 5 categorías: Dependencias Funcionales (FD), Dependencias de Inclusión (IND), Dependencias Topológicas (TD), Dependencias Referenciales Espaciales (RD) y Restricciones de Comprobación (CC). Esto permite especificar relaciones topológicas entre objetos espaciales, y permite imponer restricciones sobre atributos numéricos asociados a geometrías.

2.2.3. Relaciones topológicas entre segmentos de línea

Las trayectorias se pueden representar de diversas formas, y una opción que permite simplificar la dimensión temporal de éstas es representar una trayectoria como una secuencia de puntos, los que forman un segmento de línea. Dada esta simplificación se torna importante conocer los modelos de representación de relaciones topológicas en segmentos de línea, los cuales serán brevemente explicados a continuación.

Los modelos de esta sección se caracterízan por el uso de una matriz para la representación del tipo de intersección que está presente entre los segmentos, conocida como matriz de intersección. Las matrices de intersección difieren en dimensiones según los elementos que utiliza el modelo para caracterizar la relación topológica, y según la comparación que se hace.

El primero de los modelos de intersección se conoce como 9-Intersection Model (*9-IM*) propuesto por Egenhofer en 1990 [45] y establece una caracterización de las relaciones topológicas entre objetos geométricos (regiones, líneas y puntos). Para los segmentos de línea se establece que los puntos extremos forman el *borde*, el *interior* corresponde a todo el segmento de línea exceptuando los puntos extremos, y el *exterior* es todo aquello que no es *interior* ni *borde* del segmento de línea.

El *9-IM* utiliza una matriz de 3×3 para representar las relaciones topológicas entre dos objetos espaciales, donde cada celda indica si existe intersección entre sus elementos. La Tabla 2.2.1 muestra una definición de la matriz de intersección para

los objetos A (columna izquierda) y B (fila superior). Por ejemplo, la intersección entre el interior de A y el borde de B se representa en la celda con $A_i \cap B_b$. Es importante mencionar que, a pesar que existen 2^9 combinaciones posibles, existen varias que no son válidas; como la intersección vacía entre el exterior de ambos objetos. Es así que se llega a un total de 33 relaciones posibles entre segmentos de línea simple [45]. La Figura 2.2.6 muestra un ejemplo de intersección entre dos segmentos de línea por medio de un esquema, y su correspondiente matriz de intersección según el *9-IM*.

	B_i	B_b	B_e
A_i	$A_i \cap B_i$	$A_i \cap B_b$	$A_i \cap B_e$
A_b	$A_b \cap B_i$	$A_b \cap B_b$	$A_b \cap B_e$
A_e	$A_e \cap B_i$	$A_e \cap B_b$	$A_e \cap B_e$

Tabla 2.2.1: Definición de matriz de intersección propuesta por Egenhofer [45].

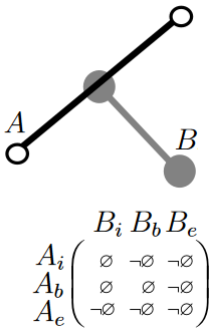
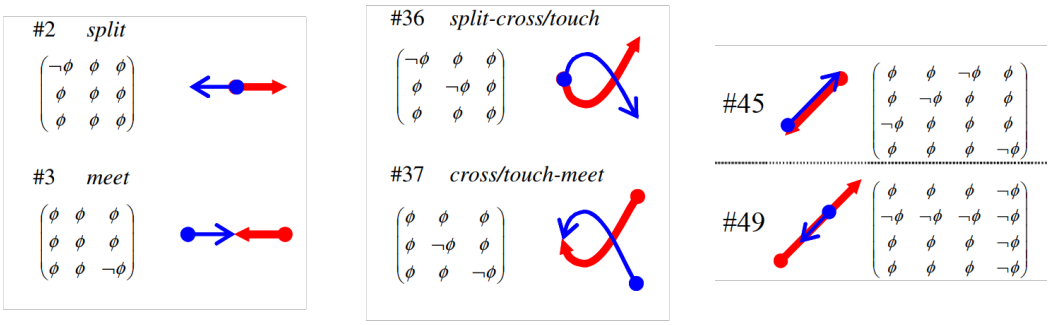


Figura 2.2.6: Ejemplo de 9IM entre dos líneas simples [45].

En el *9-IM* se consideran segmentos de línea donde los extremos no se diferencian entre sí, impidiendo la representación de líneas dirigidas. El Head Body Tail Intersection Model (*hbt*) es una extensión del *9-IM* que representa relaciones topológicas entre segmentos de línea dirigidos [72]. En primera instancia el *hbt* elimina el componente *exterior* de los segmentos de línea, y divide los bordes en *head* y *tail*. Así que también utiliza una matriz de 3×3 para representar las intersecciones.

El modelo *hbt* identifica 80 relaciones, 68 desde el modelo original y 12 desde la extensión hbt^+. La Figura 2.2.7 muestra algunos ejemplos de relaciones topológicas entre segmentos de línea dirigidos representados por el modelo *hbt* y hbt^+.

Existen otros modelos de representación de relaciones topológicas entre segmentos



(a) *hbt* int-int (b) *hbt* no int-int (c) hbt^{+}

Figura 2.2.7: Muestra de algunas de las relaciones topológicas propuestas por Kurata en [72]. Las relaciones se agrupan de 3 maneras: (a) *hbt* de relaciones entre segmentos donde no hay intersección entre los elementos interior de cada elemento, (b) *hbt* relaciones donde el interior de ambos elementos se interseca, y (c) hbt^{+} intersecciones donde el interior de un elemento interactúa con el exterior del otro.

de línea que también hacen uso de matrices de intersección para definir las interacciones entre los elementos. Por ejemplo el modelo de intersección para segmentos de línea en espacio cíclico [122]. En este contexto los segmentos de línea son elementos contenidos en un espacio con forma de anillo, de una dimensión. Esto es similiar a los recorridos de transporte público, donde los medios de transporte sólo pueden moverse dentro de los límites de una ruta cíclica.

En todos estos modelos de representación de relaciones topológicas las matrices de intersección permiten identificar si existe o no una interacción entre los elementos que representa cada celda, pero no se tiene una cuantificación de la intersección. El Modelo de Relaciones Topológicas con Detalles Métricos (*TRM-MD*) permite determinar el grado de intersección que hay entre los elementos, o incluso la cantidad de intersecciones que hay para aquellos casos en los que es posible contarlos. Esto lo logra por medio de una matriz de intersección donde, en vez de sólo indicar si existe o no la intersección, genera índices que permiten saber, por ejemplo, qué porcentaje de un segmento de línea A está contenido en el segmento de línea B.

2.2.4. Trayectorias Semánticas

La información sobre trayectorias por lo general está dentro de un contexto para el cual se utilizan los datos y las consultas que se espera ser capaz de responder. La combinación de la información en bruto con estos datos relacionados al contexto

permiten enriquecer los registros de trayectorias, lo que se conoce como trayectorias semánticas [102]. Normalmente esta información adicional no es capturada por los dispositivos de posicionamiento, pero son el paso fundamental para iniciar el análisis de datos de trayectorias en aplicaciones reales [2].

El análisis de datos sobre trayectorias se compone de la integración de datos espaciales, no espaciales y de trayectoria dependiendo del contexto y la aplicación, donde el usuario es quien define las características relevantes para el análisis de las trayectorias [9]. Otro ejemplo es el registro de migración de aves donde la alimentación, el clima y obstáculos en el espacio son datos que influyen en las trayectorias y pueden ser integrados explícitamente en la base de datos [125].

Entre los datos que complementan la información de una trayectoria es posible registrar el código postal, velocidad, tiempo de conducción, entre otros [132]. Si específicamente se trata de trayectorias en red se puede complementar la información con la cantidad de veces que una trayectoria recorre un arco, la dirección del objeto cuando pasa por un vértice y la cantidad de veces que pasa por un nodo [116].

Una de las ventajas de la integración de información geográfica que permite caracterizar los sitios de mayor importancia, según el contexto, permite reducir significativamente la complejidad de las consultas y facilita el análisis de datos de trayectorias [2, 132], de otra forma, el trabajo que involucra construir consultas sobre los datos espaciales integrando la información contextual su vuelve más costoso.

Un ejemplo sobre trayectorias con información adicional se muestra en la Figura 2.2.8 que corresponde a trayectorias con información adicional respecto de los sitios de interés identificados como hoteles (H), centro de conferencia (CC) y lugares turísticos (TP).

En el contexto de medios de transporte es posible incluir datos sobre la región en la que se transita, el vehículo y el camino, con el fin de generar rutas alternativas cuando existe congestión en el tráfico [9, 23].

Es posible determinar la información geográfica que resulta relevante para la aplicación utilizando un algoritmo de recomendación que hace pre-procesamiento para reducir la complejidad temporal del análisis semántico, siendo el usuario quien finalmente establece las características relevantes para el análisis de las

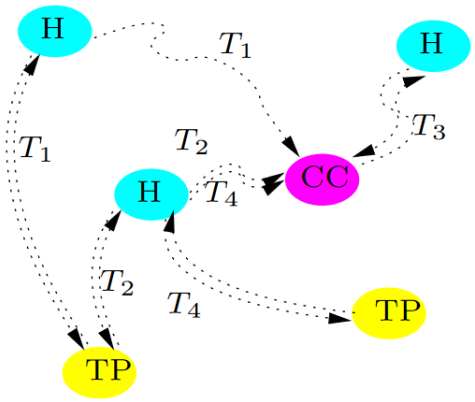


Figura 2.2.8: Trayectoria con semántica de los sitios de interés [2]

trayectorias [2]. Una vez teniendo puntos de interés es posible definir la ruta de la red para posteriormente representar las consultas asociadas a los datos espaciales, no espaciales y de los objetos en movimiento [52].

Cuando existe información semántica adicional a las trayectorias se agrega rendimiento y se mejora la calidad de la información que se puede extraer del sistema. Por ejemplo, si ya existen puntos de interés de la red es posible obtener patrones de las trayectorias considerando esta información de modo que simplifique las consultas semánticas al usuario [6]. También aplicado en campos como la privacidad de datos [102], las redes de transporte donde se etiqueta la actividad asociada al desplazamiento [19] y otros.

Es importante tener en cuenta que los modelos orientados a tipos de datos no son suficientes para soportar la semántica de las trayectorias en una aplicación específica, así como aquellos modelos basados en patrones de diseño son menos específicos aún, ya que no se restringen a un tipo en particular [1].

2.2.5. Índices para Representación de Trayectorias

Algunas propuestas incluyen estructuras de datos que facilitan el procesamiento de las consultas tomando ventaja de la forma en la que estos se almacenan. Consultas como la ubicación de un objeto en movimiento en un instante de tiempo dado, si una trayectoria entra en un área determinada durante algún instante de tiempo, si se mantiene siempre dentro de un área, si sale de un área, son algunos ejemplos de las consultas que responden los índices que se describen a continuación, los cuales se dividen en dos tipos: índices en memoria secundaria que están relacionados a

DBMS y aquellos que trabajan en memoria principal de manera compacta.

2.2.5.1. Índices sobre Memoria Secundaria

SETI [22] (Scalable and Efficient Trajectory Index) utiliza una estructura de indexación de 2 niveles para las dimensiones espacial y temporal, lo que permite búsquedas e inserciones de manera eficiente. Para la indexación espacial utiliza la estructura de datos R-tree, lo que permite su implementación en cualquier DBMS que soporte esta estructura. Un R-tree de 3 dimensiones permite representar un segmento de línea de manera eficiente, pero al momento de utilizar la tercera dimensión para la representación del tiempo, el índice para la dimensión temporal se vuelve ineficiente ya que crece continuamente, a diferencia del índice en las dos dimensiones espaciales que tiende a mantenerse dentro de ciertos límites [22]. Esto permite que el espacio se puede dividir en "celdas" de tamaño fijo donde cada celda almacena aquellos segmentos que están totalmente contenidos dentro. Si un segmento de línea cruza de una celda a otra, se divide y almacena en ambas. Luego, la dimensión temporal se indexa en base a las celdas, comprendiendo el intervalo de tiempo que contiene todos los elementos que pertenecen a la misma "celda". Por esto es que los índices asociados a la estructura se mantienen clusterizados, permitiendo mejoras en las consultas e inserciones.

El PA-tree [96], es una estructura parecida al R-tree, donde cada entrada corresponde a coeficientes polinomiales en vez de MBRs. Se propone la representación de trayectorias usando polinomios de Chebyshev, los que son fáciles de calcular y tienen una buena aproximación. La Figura 2.2.9 muestra un ejemplo de la aproximación, donde las Figuras 2.2.9.c y 2.2.9.d corresponden a las propuestas por el PA-tree. Esta estructura resulta más eficiente en la entrada y salida de datos que SETI, mejorando el cuello de botella presente en el procesamiento online de datos, por lo que el PA-tree se puede aplicar en el procesamiento on-line y off-line. Es más rápido que SETI en las consultas de rango espaciotemporales, no así el rendimiento en la construcción de los índices y las consultas de instantes de tiempo donde SETI es más rápida. El nodo raíz posee un PA-Tree por cada intervalo de tiempo definido, y cada uno almacena los segmentos de trayectorias que estén en el segmento de tiempo que representan.

Otra forma de aplicar el R-tree es la propuesta ANR-Tree (Adaptive Network R-tree) utilizada para responder consultas sobre predicción de vehículos en

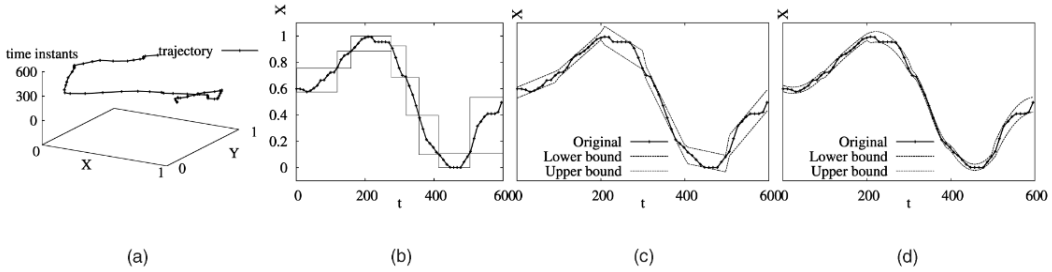


Figura 2.2.9: Aproximaciones de trayectoria en PA-tree [96]. (a) Trayectoria del vehículo, (b) Aproximación con MBRs, (c) Aproximación con PA-tree, (d) Aproximación con más coeficientes en el PA-tree

movimiento [23]. Esta estructura extiende un R-tree de dos dimensiones que indexa información espacial de la red de transporte, aprovechando las restricciones de la red y el comportamiento estocástico para el rendimiento en la consultas y actualizaciones. La estructura es de dos niveles, donde el nivel inferior almacena las representaciones de segmentos de Cellular Automata y el nivel superior la estructura de la red que se mantiene fija a menos que la red presente cambios. Cada arco del grafo que representa la red corresponde a un segmento de camino sin intersecciones y se define el movimiento en base a un autómata celular, que permite determinar el estado de un elemento del camino en base a el estado de los elementos circundantes. Así es posible determinar si una trayectoria esta sobre nodo/arco del grafo de red. El tiempo se modela como unidades de tiempo y cada objeto en movimiento en la red se desplaza un cierto número de celdas durante esta unidad de tiempo según su velocidad.

Otra estructura que también se basa en el R-tree para trayectorias sobre red es el UTR-tree [34]. La parte superior del UTR-tree corresponde a un índice basado en las aristas de la red, con lo cual la intersección entre los MBR se reduce. Las hojas de este R-tree superior llevan a la sección inferior que corresponde a R-trees que almacenan rutas con mayor granularidad donde se registran las rutas con incertidumbre. La propuesta incluye también un método para actualizar eficientemente las ubicaciones.

El Trayectory Pattern Tree (TPT) [62] corresponde a una variante del Signature tree [84] utilizado para indexar trayectorias. Cada nodo de la estructura almacena la clave del patrón (ejemplo en Figura 2.2.10), la confianza y un puntero a la región clave que representa la consecuencia del patrón. Se definen una serie de operaciones utilizadas por el TPT en su búsqueda y construcción: union, size,

contain, difference e intersect.

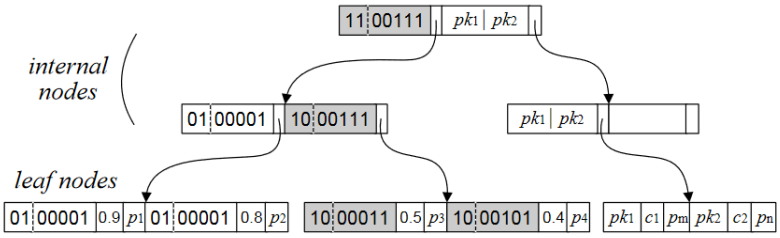


Figura 2.2.10: Ejemplos de Trayectory Pattern Tree [62]

TrajStore [30] corresponde a una propuesta para memoria secundaria diseñada para segmentar trayectorias y registrar de manera cercana los segmentos de trayectorias que son próximos geográfica y temporalmente. Este esquema de almacenamiento adaptativo se basa en el índice quadtree y permite determinar el tamaño de las celdas basado en el tamaño de las consultas, la densidad de los datos y el tamaño de las páginas del sistema. La Figura 2.2.11 muestra un ejemplo del esquema propuesto.

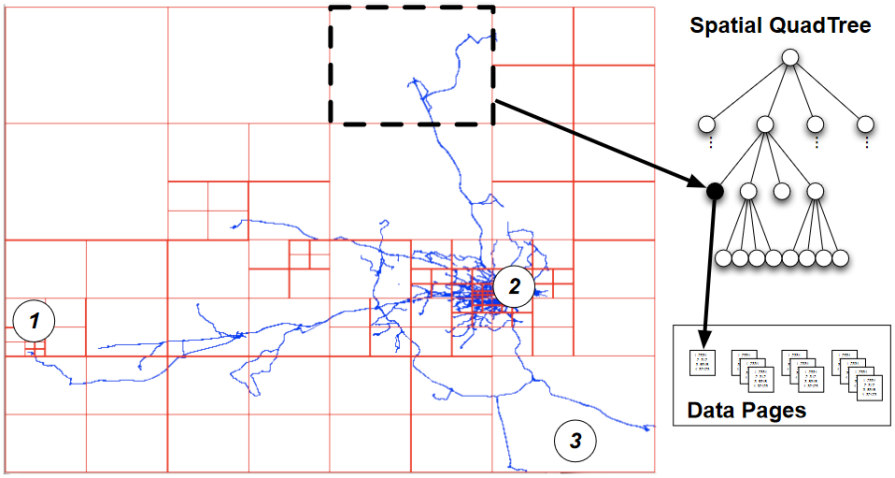


Figura 2.2.11: Ejemplos de TrajStore [30]

GCOTraj [139] es una propuesta para indexar y almacenar información espacio-temporal en una grilla de celdas multidimensionales en memoria secundaria. Estas grillas se pueden ordenar por un enfoque de ordenamiento basado en grafos o curvas de llenado de espacio. Una vez ordenadas, se almacenan de manera consecutiva aquellas sub-trayectorias que están ubicadas en la misma celda, almacenándose en el mismo bloque del disco. El enfoque de este trabajo apunta a recuperación eficiente de información antes que a la inserción o actualización. En la Figura 2.2.12

se puede ver un esquema del procedimiento.

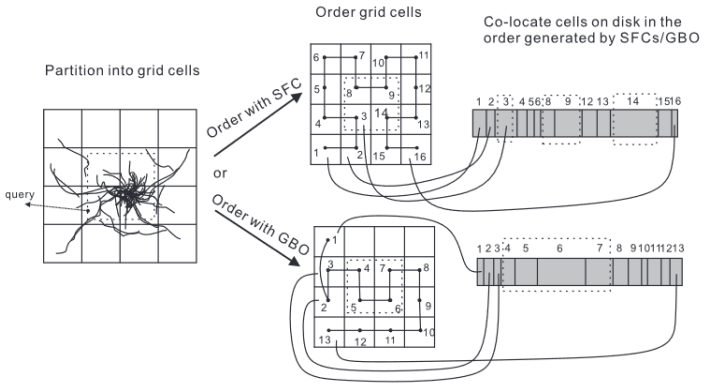


Figura 2.2.12: Ejemplo de GCOTraj [139]

Recientemente se ha propuesto una mejora que permite reducir el tiempo de respuesta en consultas mediante una estructura que funciona como un índice de baja precisión en memoria principal [20] utilizando la estructura GraCT [16] (que se describe en la siguiente sección). Esta idea permite simplificar los índices que se utilizan en memoria secundaria y mejorar los tiempos de respuesta para consultas de intervalos de tiempo.

2.2.5.2. Índices en Estructuras Compactas

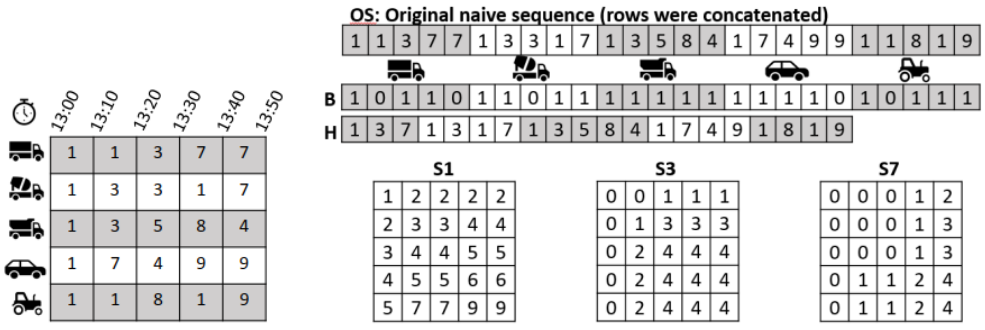
El uso de estructuras que manejen eficientemente la información en términos del espacio requerido y el tiempo de respuesta se vuelven cruciales para manejar gran volumen de información y responder consultas eficientemente. Es por ello que esta sección describe algunos trabajos que utilizan Estructuras de Datos Compactas para la representación de datos de trayectorias y las principales consultas que son capaces de responder de manera eficiente.

Brisaboa *et al.* [14] proponen una estructura de datos compacta denominada CTR (Compact Trip Representation) para la representación de los viajes realizados por usuarios del transporte público. Esta estructura utiliza un Suffix Array para representar el componente espacial de las trayectorias y una variante del Wavelet-Tree o un Wavelet Matrix para la representación de la componente temporal. Definen 2 tipos de consultas, una del *tipo número de viajes* y otra de *top-k*. La primera corresponde a consultar por la cantidad de viajes que hay entre los datos donde pueden haber restricciones temporales (una ventana de tiempo para el inicio, fin o todo el viaje), espaciales (punto de inicio, punto fin o ambos) y una

combinación de los dos. La segunda responde a aquellos nodos relacionados a la mayor cantidad de viajes, también con restricciones espaciales o temporales. En sus resultados muestran reducción de un 75 % en el uso del espacio en comparación con el baseline y las consultas son resueltas en un rango de tiempo de 1-1000 [μs].

Posteriormente se extiende el trabajo anterior para enfocarse en la representación de los viajes de usuarios de una red de transporte. En este contexto es necesario contar con más información que sólo las trayectorias de los propios usuarios ya que es deseable conocer, por ejemplo, qué usuarios abordan un determinado vehículo, lo que corresponde a elementos relacionados a la red de transporte y que se deben almacenar también en la estructura. La propuesta se denomina TTCTR (Topology&Trip-aware Compact Trip Representation) [13] y es capaz de responder consultas sobre patrones de movimiento de los usuarios utilizando la mitad del espacio requerido por una representación plana. Adicionalmente también se propone XCTR (eXtended Compact Trip Representation) orientada para abordar los puntos débiles de la estructura anterior, asociadas a consultas donde hay puntos de inicio y de fin.

Semantrix [19] corresponde a una estructura de datos compacta propuesta para la representación de porciones de trayectorias asociadas con etiquetas. La estructura se compone de un bitvector, un vector de enteros y un vector de matrices. Los dos primeros para la representación de la secuencia de actividades y los intervalos discretos de tiempo y el vector de matrices corresponde a una matriz por cada posible actividad relacionada a la secuencia de actividades. Un ejemplo se puede ver en la Figura 2.2.13.



(a) Información original (b) Esquema de la estructura

Figura 2.2.13: Ejemplo de estructura semantrix [19]

Por otra parte existe MRTS (Multi Resolution Trajectory Sketch) que entrega

una novedosa idea para responder consultas sobre vecinos cercanos, clasificación, distancias entre trayectorias y sub-trayectorias [3]. Esta representación compacta traza de manera aleatoria discos en el espacio y luego utiliza una representación aproximada de las trayectorias respecto de su interacción con los discos, registrando en un bitmap los discos con los que se interseca cada trayectoria como se muestra en la Figura 2.2.14. Dado que trayectorias similares van a cruzar los mismos discos, se aplican técnicas de hashing sensible a localidad con lo que las trayectorias similares quedan en el mismo bucket.

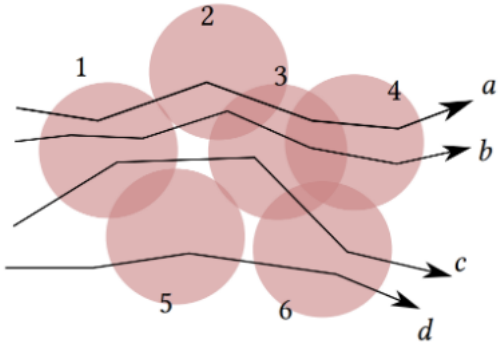


Figura 2.2.14: Discos aleatorios para representar trayectorias [3]

Otra propuesta sobre índices apunta al uso de compresión de gramática para representar trayectorias aprovechando el hecho que en muchas aplicaciones las trayectorias tienden a ser similares a las demás [16]. *GraCT* combina el uso de la compresión de gramática con estructuras adicionales para dar soporte eficiente a consultas sin la necesidad de descomprimir la información. Se utiliza la estructura k2-tree [18] para representar los *snapshots* que corresponde a las posiciones de los objetos en un instante de tiempo. Estos snapshots son a intervalos regulares de tiempo, y los movimientos que se generan entre dos de estas estructuras se representan por una secuencia comprimida por Re-Pair [76].

Los elementos hasta aquí descritos en esta sección corresponden a estructuras de datos compactas, es decir, que permiten la consulta directa de la información sin la necesidad de descompactar la información a su representación original u otra ingenua. Existe un área anterior a la compactación conocida como compresión de datos que también permite mantener la información en un espacio reducido pero, en desventaja, la información se debe descomprimir para ser manipulada o consultada. En ésta área también se abordan aplicaciones sobre datos de trayectorias y, a continuación, se describen dos de ellas.

SQUISH [91] es un método de compresión de trayectorias propuesto en el contexto de la transmisión de información GPS. El algoritmo se basa en el uso de una cola de prioridad para insertar la información y, una vez está lleno, reemplazar los puntos con poca o nada de información por los nuevos que provienen del stream, como muestra la Figura 2.2.15. La priorización se basa en la Distancia Euclidea Sincronizada, que corresponde a una medida entre dos puntos en un mismo instante de tiempo.

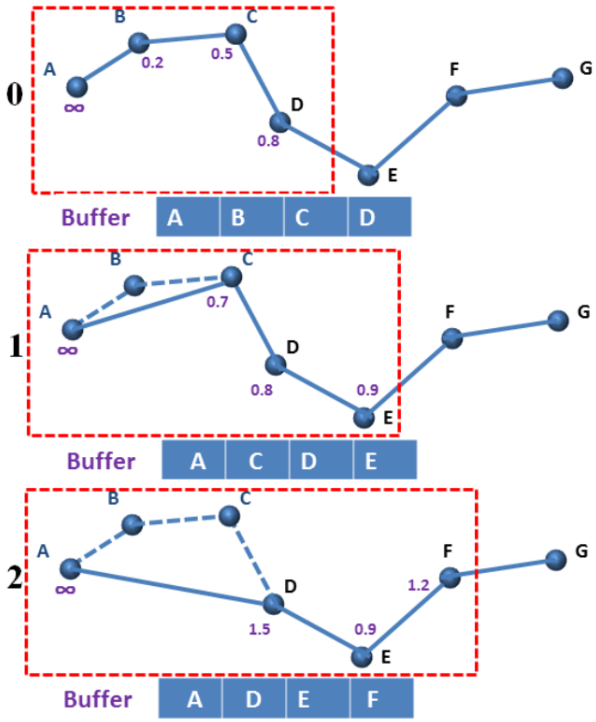


Figura 2.2.15: Ejemplo del algoritmo SQUISH [30]

En la compresión de la información de trayectorias hay ocasiones en las que existe tolerancia a cierto grado de errores o falta de precisión. Trajic [97] es un algoritmo que presenta buenos índices de compresión en escenarios con poca precisión requerida. El algoritmo procesa los puntos adivinando cuál es el la siguiente ubicación, luego esta ubicación se contrasta y se genera un residuo que se utiliza para la generación de la nueva predicción. Luego, la secuencia de puntos de la trayectoria se codifica con Leading Zero y los residuos se almacenan para responder consultas sobre los datos sin necesidad de decodificar toda la secuencia.

Capítulo 3

TRGST: An Enhanced Generalized Suffix Tree for Topological Relations between Paths

En este capítulo se presenta el primer artículo de revista que se trabajó durante la tesis, que se enfoca en el tratamiento de trayectorias representadas como secuencias sobre una red [109]. El artículo se entrega en su versión original sin modificaciones respecto de la publicación. La estructura *TRGST* es fruto de un trabajo anterior en el que se explora el uso del *GST* para consultas binarias de relaciones topológicas entre trayectorias, cuyo detalle se puede revisar en la Sección 6.2 (en los anexos del documento).

Abstract

This paper introduces the *TRGST* data structure, which is designed to handle queries related to topological relations between paths represented as sequences of stops in a network. As an example, these paths could correspond to stops on a public transport network, and a query of interest is to retrieve paths that share at least k consecutive stops. While topological relations among spatial objects have received extensive attention, the efficient processing of these relations in the context of trajectory paths, considering both time and space efficiency, remains a relatively less explored domain. Taking inspiration from pattern matching implementations, the *TRGST* data structure is constructed on the foundation of the Generalized Suffix Tree. Its purpose is to provide a compact representation of a set of paths and

to efficiently handle topological relation queries by leveraging the pattern search capabilities inherent in this structure. The paper provides a detailed account of the structure and algorithms of *TRGST*, followed by a performance analysis utilizing both real and synthetic data. The results underscore the remarkable scalability of the *TRGST* in terms of both query time and space utilization.

3.1. Introduction

The rapid advancements in mobile device technology have facilitated the collection of extensive spatial data sets. An illustrative example is the ubiquitous mobile phone, which empowers individuals to gather data regarding their movements, resulting in the accumulation of substantial spatial data. This development has also led to an increasing interest to represent trajectories that describe the movements of objects through space and time, as well as to study data structures to efficiently store and manage this type of data [58, 69, 37, 68, 19, 15, 20, 16, 14, 34, 23, 104].

When discussing trajectories, it is essential to recognize that they consist of two primary components: the temporal dimension, which represents the time interval of the movement, and the spatial dimension, which denotes the changes in the position of the moving object within a physical space. In this paper, we concentrate on the spatial component of a trajectory, referred to as a *path* and defined by a sequence of spatial references. With regards to the representation of paths, two models are commonly utilized: free-space representation and network-restricted representation. In a free-space representation, the spatial references are geometric coordinates in a space (e.g. the Euclidean space), whereas the network-restricted representation constraints the movement of objects to elements of a network, such as in the case of public transportation systems. In the latter approach, which is the one adopted in our work, the representation of paths is typically a sequence of elements on the network, where these elements belong to a finite set and can be represented by labels or identifiers.

In the context of integrating and querying moving object data, it is crucial to support queries that involve relations between paths, such as intersections, overlaps, and so on. In addition to paths that can fulfill some selection criteria, relations could be useful in the context of improving data quality and reducing inconsistencies. An example of the applications could be the public transportation

domain, where vehicles travel through a transportation network, and where it is interesting to establish routes of services that intersect. Moreover, in the presence of records detailing the movement of each vehicle, it becomes possible to detect instances where vehicles deviate from their prescribed routes. This could help in detecting unexpected incidents in the public transportation system, leading to a better management and monitoring of the provided services.

The definition and efficient support of aforementioned relations, in the particular case of spatial objects, have been the focus of much research, and even flagship database systems incorporate support for them [43, 101, 112, 108]. In the case of trajectories, however, the situation is quite different. When trajectories are defined in free space, paths can be described as line segments with direction, and several works have explored adapting existing solutions for spatial objects [46, 72, 122]. However, for trajectories restricted to a network, as studied in this work, there are no implementations that provide support for such relations. On the other hand, space-efficient representations exist for both free-space trajectories and trajectories restricted to a network [58, 69, 68, 19, 15, 20, 16, 14]. However, the support for topological relations on such representations has not been studied thus far. Therefore, our data structure, referred to as the *TRGST*, emerges as the first efficient solution for topological relations between paths of trajectories restricted to a network. In other words, the work in this paper concentrates on the representation of paths on networks and proposes a data structure for their space-efficient representation, along with time-efficient algorithms to answer topological relations between such paths. These relations involve their connectivity, such as containment, inclusion, and intersection.

The *TRGST* adapts and improves upon previous research on (text) sequence representations, with a specific emphasis on the spatial domain of topological relations. In particular, the solution proposed in this work utilizes a Generalized Suffix Tree (GST) to represent a set of paths that are sequences of elements that identify positions on a network. These sequences are concatenated to construct the structure and leverage the benefits of the GST in responding to topological relation queries. Furthermore, the GST is complemented with some additional data structures to provide efficient support for the studied operations. The GST employed is based on a Compressed Suffix Tree, enabling the space-efficient representation of repetitive data and eliminating the redundancy present in

non-compact versions of the same structure. It is important to notice that, in many contexts related to trajectories on networks, the repetitiveness of the data is expected to be high as moving objects tend to travel similar paths several times. Not to mention that the volume of data in such domains tends to be enormous. In such a context, compact data structures (CDS) can harness their potential for efficient memory utilization [14, 16, 15, 13, 63, 68, 64]. Our experimental evaluation shows that the *TRGST* can effectively leverage this redundancy, providing an efficient space solution that also supports the relations of interest in very competitive query times.

The organization of the paper is as follows. Section 3.2 reviews related works on trajectories, spatial relations, and some CDS relevant to this work. Then, Section 3.3 formalizes the problem, describes the topological relations of interest as well as the representation and algorithms used to answer the queries. Section 3.4 describes the data for the experimental evaluation, followed by the experimental comparison with respect to a baseline in terms of query time and space usage. Finally, Section 3.5 presents the conclusions and potential future research directions.

3.2. Background and Related Work

This section covers various topics related to trajectories, including their representation, relations, and CDS that form the foundation of this paper. Firstly, it is essential to explore existing trajectory representations. In this context, efficient space utilization is particularly relevant. Subsequently, in Section 3.2.1, several CDS proposed for indexing trajectory data are explored. Following that, Section 3.2.2 discusses relations between trajectories, with a specific focus on understanding the relations concerning trajectory paths. Lastly, Section 3.2.3 provides an explanation of the Generalized Suffix Tree, as it serves as the main building block for our proposal.

3.2.1. Trajectory Representation

This section covers two trajectory representation models: free space and network-constrained. For each of them, various data structures exist, and more recently, efforts for not only having time but also space efficient structures lead to apply

compression techniques and CDS.

In the realm of approaches to representing free-space trajectories, the majority of data structures are adaptations of the R-tree [57], a data structure used for indexing spatial information. The R-tree efficiently handles the representation of the spatial dimension, and the proposals for trajectories adapt it to accommodate both temporal and spatial dimensions or employ a different structure for storing the temporal dimension. Notable examples include the STR-Tree and TB-Tree [105], MV3R-Tree [126], and SETI [22]. Usually, these indexes answer window queries (objects within a specified spatial region), time slice or timestamp queries (objects that intersect a window at a specific instant), and time interval queries (objects that intersect a window over a range of timestamps).

To reduce space, an effective approach for compressing the spatial dimension is through delta compression, wherein a set of closely related numbers is encoded as the difference from their respective predecessors. This approach proves to be useful for compressing points of the spatial dimension of trajectories because these points tend to be close in space. As an example of this approach, TrajStore [30] divides trajectories into cells and subsequently applies delta compression to represent trajectories within each group.

Regarding CDS for free-space trajectories, ContaCT [15] is an example of a CDS that offers constant-time access to compact trajectory data through delta compression. It efficiently manages queries pertaining to position, trajectory, spatial range, nearest neighbors, as well as range minimum/maximum. An alternative is the use of grammar compression, which emerges as an innovative technique of significant relevance. This technique has been applied in GraCT [16], using snapshots of the positions of moving objects. It employs the compact k^2-tree data structure [17] for position snapshots and grammar compression for the sequences that represent the movements of the objects between two consecutive snapshots. These snapshots significantly enhance the efficiency of range and nearest neighbor queries. The supported queries encompass a wide range of functionalities, including position at a specific time instant, trajectory between two time points, time slice, time interval, and nearest neighbor queries. This use of CDS to provide representations that, in less space than the original plain data, support both the retrieval of the original data and queries on them, has been referred to in the literature as self-indexes [48].

When the movements of an object are restricted to a network, it is possible to reduce the number of dimensions of the spatial data from 2D to 1D by given an identification to positions on the network, and consequently the trajectory data is reduced from 3D to 2D [104]. This is a great advantage when a network-constrained trajectory (NCT) is compared with respect to a free space trajectory representation, as mentioned earlier.

For NCT, Graph of Cellular Automata (GCA) can represent the network. In GCA the edges are depicted by multiple cells through which a moving object can traverse [23]. Once the transport network is mapped, it becomes possible to utilize data structures to index the trajectory data over the network, such as the UTR-Tree [34]. In this approach, queries related to historical, present, and near future possible locations of moving objects are supported.

One way to perform trajectory compression on a network is to focus on their paths and attempt to reduce their length. SPNET [69], an in-memory index, employs Shortest Path Encoding based on the observation that trajectory paths often conform to the shortest route. Consequently, it needs storing only a subset of the path elements comprising the trajectory, enabling efficient handling of comparison, range, and path queries. Additionally, Kellaris *et al.* [64] proposes a way to compress data trajectory without modifying the core structure of the moving object database. This by substituting some paths with shorter alternatives with the highest quality measure defined as a factor of the compression and the similarity with respect to the original path.

Koide *et al.* introduced CiNCT [67], a compression method designed for NCT represented as sequences of road edges. This is built upon the FM-index [49] and introduces the concept of Relative Movement Labeling (RML) to enhance space efficiency. In this framework, the alphabet corresponds to the number of road segments within a road network. RML transforms trajectory strings into strings over a smaller alphabet. This transformation is based on the fundamental observation that NCTs can only transit between physically connected road segments. Consequently, this approach facilitates pattern matching and enables the system to respond to a variety of queries, including range, path, and exact-path queries.

A self-index based on CDS for trajectories on a network is the *Compact Trip*

Representation (CTR) [14]. In this structure, the spatial dimension is represented as a sequence in a *Compressed Suffix Array*, and the temporal dimension can be represented using a *Hu-Tucker-shaped Wavelet tree* (uses the Hu-Tucker [60] code that preserves the order of the input vocabulary) or a *Wavelet Matrix*. CTR uses between 50 % and 70 % of the space required by a non-indexed compact representation. It responds *counting queries* categorized into *number-of-trips queries* and *top-k queries*. These categories can be further sub-classified based on whether they involve the spatial, temporal, or both dimensions. Examples of these queries include the number of trajectories starting at a certain stop of the network, the number of trajectories passing through a stop within a given time interval, the *top-k* most used stops, the *top-k* most used stops to initiate a trip within a time interval, and more. Later on, TTCTR and XCTR [13] further extend the functionality of CTR, enabling queries related to network load and specific patterns along the travel path. These structures efficiently manage information related to trips and the transportation network.

In the context of representing trajectories constrained to networks, it is important to highlight that various approaches have been developed to address this challenge. The most relevant ones have been mentioned above; however, to the best of our knowledge, none of these structures addresses relations between paths of trajectories.

3.2.2. Trajectory Relations

This section begins by reviewing various studies that explore relations between trajectories, with a specific emphasis on relations between paths. It starts by examining the concept of trajectory similarity, a flexible and less stringent approach to establishing relations between trajectories based on the underlying path concept. Subsequently, it introduces a well-established model for describing topological relations between spatial objects, setting the stage for a deeper exploration of path-to-path relations. These relations are defined for lines with direction.

There are various clustering techniques that can be employed to identify patterns or similarities among trajectories across different elements. These techniques operate at different levels of detail within the spatial dimension and utilize different criteria for clustering [35]. One approach is to perform density-based clustering [93, 65], where trajectories are grouped based on their proximity and density in the spatial

domain. This helps to identify areas of high trajectory density, clusters with similar movement patterns, or specific transportation routes. Another technique involves dividing trajectories into sub-trajectories based on specific criteria [54], such as time intervals or changes in direction. This allows for a more detailed analysis of the trajectory segments and the identification of similar sub-patterns. Furthermore, clustering can take into account the neighbors of trajectories [80], considering the spatial proximity and interactions between nearby trajectories. This enables the identification of groups or clusters of trajectories that exhibit similar spatial behaviors.

The techniques for clustering described above rely on assessing the similarity between trajectories, which implies more flexible relations, accommodating scenarios where an exact match in their spatial dimension does not exist. Unlike clustering, the work in this paper concerns relations between paths of trajectories, which explore different ways that paths share common elements and are useful for retrieval, integration and consistency.

A trajectory refers to the change of location of an object and, in this context, there are some common spatial relations based on topological reasoning: *intersect*, *meet*, *equal*, *near* and *far* [9]. On this, it is relevant the context of the scale, because the *far* and *near* relations between two walking people do not use the same scale as those applied to two airborne airplanes. Furthermore, considering the possibility of relations between distinct elements, it becomes evident that movement-based interactions can also occur between a region and a line segment. These interactions encompass concepts such as *stayWithin*, *bypass*, *leave*, *enter*, and *cross*, as proposed by Brakatsoulas *et al.* [9]. It is worth noting that these last five relations include directions along the line.

As trajectories inherently have a spatial component, it becomes important to delve into the relations between spatial elements. Among relations between spatial objects, topological relations have gained special attention and are part of query languages of flagship spatial databases [43, 112, 108]. A well-known model of spatial objects is the 9-Intersection model [43, 101], which defines relations between non-empty spatial objects based on the intersection of their boundaries, interiors, and exteriors. In consequence, the 9-Intersection Model (9-IM) establishes that a topological relation between two spatial objects in $\mathbb{R}^2$ can be represented by a 3×3 matrix to describe the intersection between the parts of each spatial element.

Given that there are 9 cells, and each cell has 2 possible values, there are $2^9 = 512$ combinations, many of which are not feasible in a real representation of spatial elements. For example, it is not possible for the exterior of one object to have no intersection with the exterior of the other, so the cell in row 3 and column 3 must always indicate a non-empty intersection. Focusing on non-empty areas with connected boundaries, this model identifies eight basic topological relations: *Equals, Covered By, Covers, Inside, Includes or Contains, Overlap, Touches or Meet,* and *Disjoint*. The same relations were also identified by a logic-based model of topological relations [112].

In addition to these eight basic relations between areas, there are other relations that result from aggregations or combinations of some of the basic ones. Figure 3.2.1 illustrates a hierarchy of the eight fundamental relations: Overlaps (O), CoveredBy (CB), Inside (IS), Equals (EQ), Covered (CV), Includes (IC), Touches (TO), and Disjoint (D). Additionally, it showcases four supplementary relations that are aggregations of the basic ones: Within (W), Contains (C), Intersects (IT), and InternallyDisjoint (IDC). In Figure 3.2.1, the relations represented with thick borders are currently included in spatial extensions to SQL, as they are the named relations in the definition by the Open Geospatial Consortium (OGC) [59].

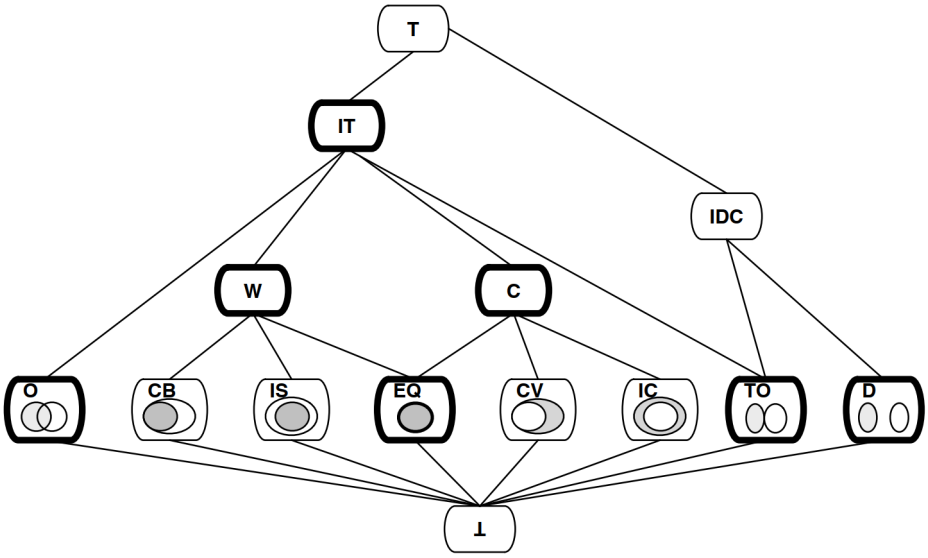
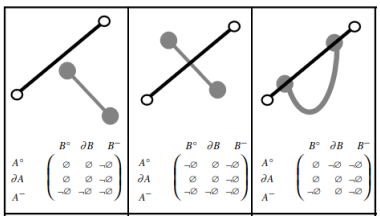


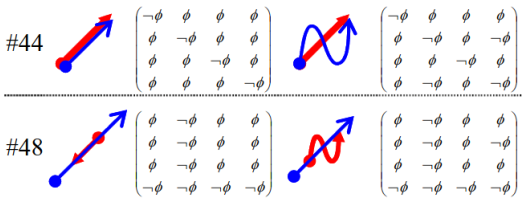
Figura 3.2.1: Binary relations between non-empty regions (Taken from [10]).

When considering the path of trajectories, it is necessary to focus on relations between lines. The study of topological relations over line segments has also been well researched by Egenhofer *et al.* [38, 39, 40, 41, 42, 43, 44, 45, 46]. By applying

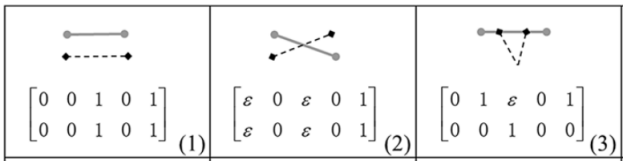
constraints on the 3 × 3 intersection matrix between two simple line segments, Egenhofer *et al.* identified 33 possible relations [45]. Some examples are shown in Figure 3.2.2(a).



(a) Example of Egenhofer 9-IM [45]



(b) Example of Kurata HBT-IM [72]



(c) Example of Shen TRM-MD [123]

Figura 3.2.2: Some of the intersection matrices proposed to represent topological relations between line segments.

Several alternatives to the 9-IM have been proposed. One of them extends it to consider the dimension of the intersection between the elements of a spatial object, what is called the dimensionally extended 9-intersection model (DE-9IM) [27]. This extension aims to improve the expressiveness of the topological relations already found by specifying the dimension of the intersection between the elements. When the intersection corresponds to a point, and not to a line segment or an area, it uses a value of 0. If the intersection contains at least one line segment and no area, it uses the value 1. If the intersection corresponds to at least one area, the value used is 2. Finally, the empty intersection is represented by the symbol −. This model is used on the Resource Description Framework (RDF) of the OGC [1], as well as in SQL cross-relation, where the intersection of the interiors of two lines has a dimension value of 0.

If the intention is to apply these models to represent movement, it becomes important the direction of motion. In this context, the 9-IM has also been extended by Kurata [72]. In this extension, the direction of the line segments is taken into account. As a result, the *boundary* is no longer treated as merely two endpoints; instead, a distinction is made between the *head* and *tail* of a line segment, considering their respective directions. With this division, a total of 80 possible

[1] https://www.ogc.org/

relations have been found, 68 of which are titled *hbt-intersection*, and the remaining 12 complement the former with the title *hbt$^+$-intersection* (some examples are shown on Figure 3.2.2(b)). Additionally, there is a proposal that considers the direction of segments within a cyclic space. In this context, a total of 38 distinct topological relations have been identified: 28 relations where the starting or ending points do not intersect, and 10 relations where there is an intersection between these points [122].

The previous proposals allow representing topological relations between line segments by identifying the intersection between their elements. However, even when knowing that the dimension of the intersection is a line, it is not possible to determine the magnitude of such intersections. The Topological Relations Model with Metric Details (TRM-MD) [123] uses a 2-rows and 5-columns matrix to represent the intersection between elements of two spatial objects. In contrast to previous models, the cells of this matrix do not only indicate the presence, absence, or dimension of the intersection, but they use values in the range $[0, 1]$ to represent the extension of the intersection between line segments.

Previous models could be used to relate paths of trajectories, by considering the time as a perpendicular and independent dimension for representing trajectories. A different approach is the Qualitative Trajectory Calculus (QTC) [130], which considers how space and time interplay in the definition of the relation between trajectories. In addition to considering time to define relations, QTC represents various situations of the disjoint relation, which in previous proposals is defined as the elements of one object relating only to the exterior of the other. Hence, the QTC_c model takes into account the direction in which an object moves concerning the line segment connecting it to a second object. A practical example of its application is when two people are close, and their movements do not intersect. It becomes interesting to know if one of them is moving away from the imaginary line connecting them, if they are getting closer in the same direction, or if they turn. The QTC_c model identifies 81 relations. Despite the additional semantics provided by the QTC_c model, these relations are not yet standardized and have not been widely implemented in current systems.

Therefore, in the realm of relations between line segments, numerous proposals have emerged to address the representation of potential relations between the elements of two segments. However, in the context of trajectories, the focus has

primarily been on employing similarity measures to identify common patterns, rather than explicitly addressing the relations between trajectory elements. This divergence underscores the significance of the *TRGST* proposal, which aims to provide a data structure that effectively handles topological relations between trajectory paths within a network context.

3.2.3. The Generalized Suffix Tree (GST)

The path of a trajectory over a network can be represented as a sequence of stops that the moving object passes through during its movement. This sequence can be thought of as a text, with the symbols representing the spatial reference from the network. Therefore, the vocabulary consists of all distinct identifiers of these references, such as the stops in a public transportation network. Hence, it is interesting to explore how a data structure can efficiently index text sequences and answer queries related to prefixes and suffixes of the text. The Generalized Suffix Tree (GST) stands out as one of the most prominent approaches for this purpose.

The GST is a data structure based on the Suffix Tree (ST), which in turn, is a PATRICIA trie [4] constructed from all the suffixes of a sequence. This data structure enables efficient indexing of text and answering queries related to common prefixes or matches among its elements. GST is commonly used in environments with very long text sequences where data repetitiveness is prevalent, such as in bioinformatics [5, 55]. A ST is constructed by considering all the suffixes of a given text. Each suffix is inserted into the Suffix Tree by traversing its characters and inserting a node for each character from the root. This process builds the hierarchical structure of the tree, where each leaf represents a suffix of the original text and the internal nodes represent substrings of the text. The children of each node are sorted alphabetically and unary nodes are eliminated, allowing a label between a parent node and its child to contain more than one symbol. The ST facilitates efficient searching for patterns, subsequences, suffixes, and palindromes within a text, alongside various other operations.

There are two important sub-structures that compose the ST: the Suffix Array (SA) and the Lowest Common Prefix Array (LCP). The SA is an array that stores the starting positions of all the suffixes of the sequence used to construct the ST. Each entry in the SA points to the starting position of a suffix within the

original sequence. The values of the SA are arranged in the same order as a DFS that retrieves leaves from the ST. The LCP array, on the other hand, stores the length of the largest common prefix between leaf i and leaf $i-1$ of the ST. A non-zero value in the LCP array indicates the presence of a prefix match between two consecutive suffixes in the SA.

Some of the primitive operations supported by a ST include:

- **lca**: Given two nodes, it returns their Lowest Common Ancestor.
- **slink**: Given a node, it returns another node representing the same subsequence without the first character.
- **wlink**: Given a node and a character, it returns another node representing the same subsequence with the given character added as a prefix.

These operations allow the traversal between nodes that represent prefixes with common subsequences. If there is no match, there are no results for the *lca* or *wlink* queries, so both queries return the root node. Some possible problems that can be solved using a ST include finding the longest repeated substring in a sequence or finding all occurrences of a pattern in the text.

A Generalized Suffix Tree (GST) is a ST constructed from the concatenation of multiple sequences. By definition, in a GST, each sequence is assigned an *end symbol*, and therefore, each sequence ends at a leaf labeled with its respective *end symbol*. However, the use of different *end symbols* for each sequence is not actually done in practice [94]. Then, the GST can be effectively used to identify similarities or coincidences between sequences that represent paths.

Some important applications of GST include finding matches in biological data sequences [5], sequence alignment [92, 119], computing similarity measures in sequential data [114], and indexing XML files to answer indirect containment queries more efficiently than the tree traversal method [140]. Also in the DNA sequence context, it can be used to identify all substrings with no more than k mismatches, a problem known as inexact matching. Another problem is, given a set of subsequences, find a sequence S that contains all the subsequences (which is called a super-string). Furthermore, it is employed in tasks such as circular string linearization, identifying maximal repeats, and discovering super-maximal repeats [131]. Given the aforementioned applications, the GST is a promising data structure for detecting coincidences among paths, facilitating the establishment of

topological relations.

3.3. Our Proposal

In this section, we formalize the problem of representing the paths of trajectories restricted to a network and supporting topological relation queries on them, proposing a solution based on the GST described above.

As a running example, Figure 3.3.1 illustrates the path representation of 4 trajectories on a network of stops, which will guide the examples that follow in this document. In this figure the path $P1$ corresponds to the stops with IDs A, B, C, and D. Note that the representation of path $P1$ includes all stops traversed by the path (i.e. $ABCD$) and not only its origin and destination stops (i.e. AD).

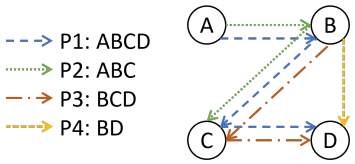


Figura 3.3.1: Paths represented as a sequence of stop IDs over a network of stops.

Figure 3.3.2 shows the corresponding GST constructed from the sequences of paths in Figure 3.3.1. In this construction, the character X has been used as the end character for all the individual sequences, while the \$ character represents the end of the concatenated sequence.

3.3.1. Formalization

This section provides the formalization of the problem, the relations between paths under consideration, and key terms that underpin the development of this study.

Definition 1. A Trajectory x is a sequence of spatial references (either points or identifiers of locations) and corresponding time instances. Spatial references, denoted as s_i, are elements of a spatial domain S and the time instances, denoted as t_i, are elements of a time domain T. In the context of trajectories restricted to a network, the spatial references correspond to elements of a network. The temporal dimension of the sequence follows the restriction that $t_1 < t_2 < t_3 < \ldots$, implying that the time instances are strictly increasing. Formally, a trajectory X

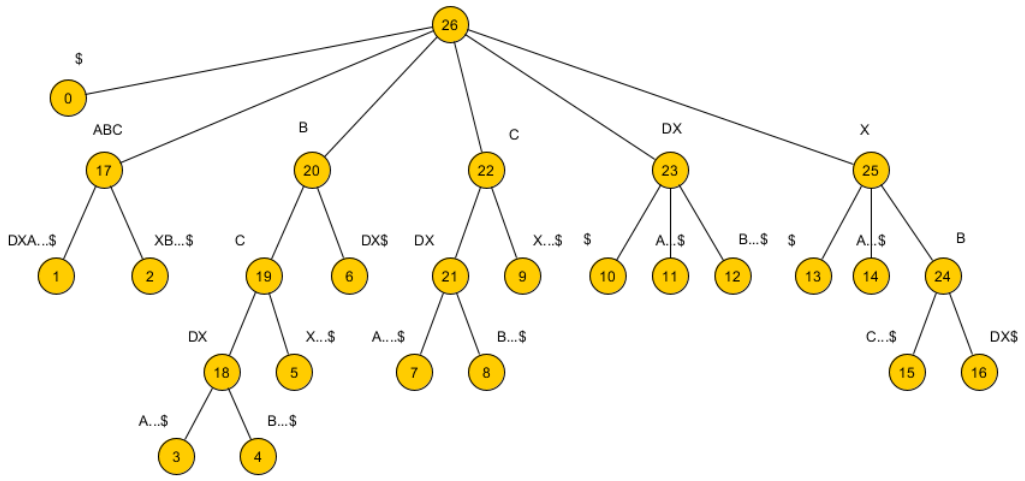


Figura 3.3.2: GST for sequences from Figure 3.3.1, which is equivalent to the sequence *ABCDXABCXBCDXBDX$* in its concatenation form.

can be defined as:

$$x = \{(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)\}$$

where $(s_i, t_i) \in S \times T$ represents a spatial reference s_i at time instant t_i.

Definition 2. A Trajectory Path (or Path) p_x, is the sequence of the spatial references from the trajectory x. Formally, a path p_x of a trajectory x can be defined as:

$$p_x = \{s_i, s_{i+1}, \ldots, s_n\}$$

where $s_i, s_{i+1}, \ldots, s_n \in S$ are spatial references.

In this work, we concentrate on the spatial dimension of trajectories, namely the paths defined above, with a specific emphasis on queries concerning the topological relations existing between such paths.

Definition 3. A Topological Path Relation Query (TPRQ) is a query operation defined over a set of paths P. Given a topological relation T, a set P of paths, and a path $p \in P$,

$$TPRQ(P, T, p) \rightarrow P'$$

retrieves a set $P' \subseteq P$ such that all paths $p' \in P'$ and the relation T between p and p' holds.

It is noteworthy that the topological relation in a $TPRQ$ query can be any of the relations illustrated in Figure 3.2.1. However, in the context of this

study, we focus on coarser topological relations that are commonly used in query languages: `Equals` (Definition 4), `Within` (Definition 5), `Contains` (Definition 6), `Intersects` (Definition 7), and, as the complement of `Intersects`, the `Disjoint` relation (Definition 8). The definition of the `Intersects` relation utilizes a value k, which represents the number of consecutive elements that must match to satisfy the relation.

Definition 4. The `Equals` relation between two paths p and q is denoted as `Equals`(p, q). This relation holds true if and only if the lengths of paths p and q are equal, and $\forall i$, $p_i = q_i$.

Definition 5. The `Within` relation between two paths p and q is denoted as `Within`(p, q). This relation holds true if and only if the length of path p is equal to or less than the length of path q (i.e. $n \leq m$, where n is the length of p and m is the length of q) and there exists a non-negative integer value c such that $\forall i$, $1 \leq i \leq n$, $p_i = q_{i+c}$, where $0 \leq c \leq m - n$.

Definition 6. The `Contains` relation between two paths p and q is denoted as `Contains`(p, q). This relation holds true if and only if `Within`(q, p).

Definition 7. The `Intersects` relation between two paths p and q is denoted as `Intersects`(p, q, k). This relation holds true if there exists two non-negative integers values i and j such that $i + k \leq n$ and $j + k \leq m$ (n is the length of p and m is the length of q), where $\forall l$, $1 \leq l \leq k$, $p_{i+l} = q_{j+l}$.

Definition 8. The `Disjoint` relation between two paths p and q is denoted as `Disjoint`(p, q). This relation holds true if and only if $\forall i$, $1 \leq i \leq n$, $\forall j$, $1 \leq j \leq m$, $p_i \neq q_j$, where n is the length of p and m is the length of q.

Since the topological relations to be solved in this work are focused on the sequence representing the paths, it is possible to represent each path as an ordered sequence of stop IDs from the transportation network through which the trajectory passes. Moreover, considering that the set of paths is represented as a set of sequences, efficient string processing algorithms can be applied to answer the desired topological relations. This approach provides the *Baseline* for comparison used in this work, which is described in Section 3.4.1.

As a summary, the representation proposed in this work has the following scope:

- The representation focuses on paths, maintaining the time-ordered sequence of spatial references while excluding the recording of precise timestamps for movements. Consequently, each path is presented as a chronological sequence of identifiers, denoting the locations that the trajectory traverses, and where geometries are not a part of this representation.
- The relations defined above are based on exact matching between spatial references, which in this case are elements' identifiers on a network. That is, distances between the symbols denoting spatial references are not defined. Wildcards in searches, such as sequences of missing symbols between two matches, are also not allowed.
- The representation does not incorporate other factors such as velocity, acceleration, or environmental conditions.
- The segments of the network do not intersect. In other words, each segment connecting one stop to another does not cross or intersect any other edge within the network.
- A sequence is constructed using the identifiers (IDs) of the spatial references. Two IDs in the sequence are considered contiguous if and only if there is a segment connecting the corresponding stops associated with those IDs.

3.3.2. Description of the Data Structure

The Topological Relation Generalized Suffix Tree (*TRGST*) is the data structure proposed in this work, which represents the paths of trajectories as sequences stored in a GST and enables efficient querying of topological relations over these sequences. In addition to a GST, this data structure utilizes additional elements that are described as follows:

- *mapRoute2Node*: Each path is associated with a single node of the GST that represents its entire sequence. Since the identifiers of the paths are assigned sequentially starting from 0, it is possible to implement this structure as an array where position i records the ID of the node in the GST that represents the i-th path. This component eliminates the necessity to traverse the tree to access these nodes, significantly improving the performance of some operations on the *TRGST*. Note that the node pointed by this structure represents the path sequence including the end-marker (concatenation separator).

- *mapNode2Route*: The nodes of the GST that represent entire sequences are linked to their corresponding path. This structure corresponds to a multi-map, which allows associating one key to one or more values. In this case, a node can represent the sequence of more than one path, and there can also be nodes that do not represent any path. However, it is important to clarify that only the former are stored in this structure (i.e. nodes that do not represent any path are not stored in this structure).
- *markBranch* and *markNode*: The former allows identifying those branches of the GST that have at least one node representing a path, and the latter is used to identify the nodes that represent an entire sequence of a path. These markers are implemented using two bitmaps of the size of the number of nodes in the GST.
- *markEndSequence*: It corresponds to a bitmap that marks the positions of the last element of each path in the concatenation sequence used to construct the GST. This bitmap enables the identification of the path corresponding to a specific position within the concatenation of path sequences. Additionally, it helps in determining the path associated with a suffix of the concatenation, which is obtained from the leaves of the tree.

Figure 3.3.3 illustrates an example of the additional elements described above to complement the GST shown in Figure 3.3.2. This example represents the mappings of *mapRoute2Node* and *mapNode2Route* using the labels *P1, P2, P3*, and *P4* near the GST nodes $1, 2, 18$, and 6, respectively. The bitmaps *markNode* and *markBranch* are displayed at the bottom and, as an example, the position 2 of *markNode* has a value of 1 indicating that this node represents an entire path. Similarly, at position 17 of *markBranch*, there is a mark with a value of 1, indicating that the subtree rooted at this node has at least one descendant node representing an entire path. In this case, node 17 corresponds to the parent of the nodes 1 and 2, representing $P1$ and $P2$, respectively.

The *markEndSequence* is depicted at the top of the illustration with the concatenation of all sequences, using the character X as a separator. The marks on this bitmap are located in the same positions as the X characters in the concatenated sequence. Additionally, Figure 3.3.3 displays the values of the *SA* (stored as a CSA) and *LCP* array, that are below the GST (see Section 3.2). For instance, at position 6 of the *CSA*, there is a value of 13, indicating that the suffix

BDX ($P4$), represented by node 6, starts at position 13 in the concatenation of the sequences. The LCP has a value of 3 at position 2, indicating that node 2, which represents the sequence $ABCX$..., has a prefix in common of length 3 with the sequence represented by the previous node (node 1), which is $ABCDX$.... This also implies that paths $P1$ and $P2$ share a common prefix of length 3.

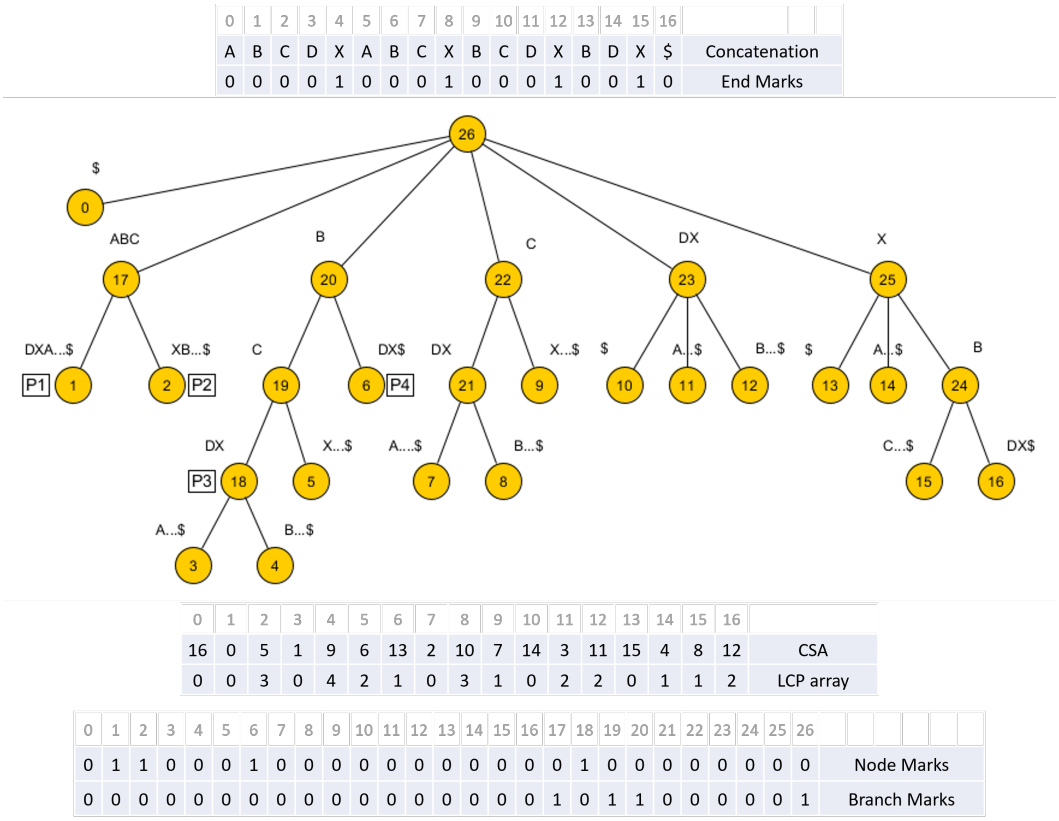


Figura 3.3.3: Example of the additional structures of the $TRGST$.

3.3.3. Description of the Query Algorithms

In this section, the methods designed to solve topological relation queries are explained, providing descriptions, examples, and algorithms for performing these operations on the $TRGST$. All the queries are implemented following the definitions of each relation described in Section 3.3.1, using the same names but differentiated by the parameters. The parameters in the definitions correspond to binary operations over two sequences. However, in the case of the implementation, the parameters correspond to the ID of a sequence from the set of paths, named a, and the set of paths, denoted by an uppercase letter X. Each query is designed to verify whether the topological relation between the sequence a and each $x \in X$

holds. The outcome of each query is the subset of path identifiers within the set that hold the specified relation. Algorithms for all the aggregated relations depicted in Figure 3.2.1 are provided, with the exception of the Internally Disjoint relation, which can be computed using the complement of the intersection.

To solve these queries about topological relations, it is important to consider that the GST node representing a path includes the end character (concatenation separator) because, in some cases, moving up to the parent leads to a node that contains the entire sequence, as in the case of path $P2$ in Figure 3.3.2. On the other hand, there are sequences where the edge between the node representing the path and its parent contains more elements than just the end of the sequence, as in $P1$ where node 1 represents the sequence $ABCDXA...$ and node 17 represents the sequence ABC. In these cases, moving up to the parent from the node that contains the sequence associated with a path implies losing elements of the sequence and, therefore, not fully representing the path. Hence, in some cases, the algorithms move up to the parent node only if it represents the complete sequence. This will be explicitly stated in each of the algorithms.

3.3.3.1. `Equals`(a,X)

The simplest operation is the `Equals` relation, which returns all $x \in X$ where `Equals`(a, x) holds (by Definition 4).

In the *TRGST*, the node representing the sequence a is obtained from *mapRoute2Node*, and then all the identifiers of sequences that are equal to a are obtained from *mapNode2Route*. The most time-consuming operation in computing the `Equals` relation involves retrieving elements from an `unordered_multimap` (*mapRoute2Node*). This retrieval is proportional to the number of sequences (elements in the map) with the same node ID (the searched key) in the map structure and, in the worst case, it becomes linear with respect to the total number of sequences in the set (total number of elements in the map). However, because retrieving elements from an `unordered_multimap` is generally considered to have constant time complexity for key retrieval, the `Equals` operation is also considered to be constant, with a high probability. As a result, the time complexity of this operation is $\mathcal{O}(w)$, where w corresponds to the number of paths that satisfy the relation.

3.3.3.2. `Within`(a,X)

The relation `Within` returns all the identifiers of the paths that entirely contain a, meaning that a is a subsequence of each of them. Using Definition 5, this operation returns all $x \in X$ that satisfy `Within`(a, x). This relation is established when one of the following relations exists between a and x: *CoveredBy*, *Inside*, or *Equals*, which are equivalent to the aggregation a *within* x, as shown in Figure 3.2.1.

In the GST, given any node, all the leaves in its subtree correspond to prefixes that contain the sequence represented by such a node. Therefore, in the *TRGST*, it is sufficient to identify the node that fully represents the path, and then all the leaves in the subtree induced by such a node represent paths that satisfy the relation.

Algorithm 1 describes the `Within` implementation. First, it obtains the value of the *nodeA* from *mapRoute2Node* (Line 1), where *nodeA* corresponds to the node associated with the sequence representing path a. From such a node, we need to evaluate if its parent node represents the complete sequence without the end element, and to start from there in such a case (Line 2); otherwise, it starts from the recovered node. It is important to note that if we were to work directly with the node obtained earlier (containing the end element), we would interpret that the leaves of this subtree correspond to sequences that contain a at the end, without considering those that contain it inside their path. Then, the leaves of this subtree are explored (Line 5), using the operations *leafL* and *leafR*, which return the identifiers of the leftmost and rightmost leaf nodes in the subtree rooted at *nodeA*. Each of these leaves corresponds to a sequence that entirely contains a, but only a few of them correspond to paths. Therefore, for each one, it is necessary to check if it corresponds to a sequence representing a path. The path identifiers can be obtained based on the positions in the *CSA* and the *markEndSequence*, where the function *idFromPos* returns the identifier of the path based on the position in the concatenated sequence (Line 7).

The `Within` operation is executed by traversing the leaves within the subtree that encompasses sequence a. If u represents the number of leaves in a subtree, the complexity of the `Within` operation is proportional to this value, expressed as $\mathcal{O}(u)$. Depending on the application domain, various scenarios may arise. However, it is possible to assume certain values for some cases, like a network of the public

Algorithm 1 `Within`$(a, TRGST)$

```
1: nodeA ← TRGST.mapRoute2Node(a)
2: if nodeA.parent().depth() == a.length() then
3:    nodeA = nodeA.parent()
4: end if
5: for i ← nodeA.leafL() to nodeA.leafR() do
6:    posSuffix ← TRGST.csa(i)
7:    idRoute ← TRGST.idfromPos(posSuffix)
8:    if NOT result.find(idRoute) then
9:       result.add(idRoute)
10:   end if
11: end for
12: return result
```

transportation system defined by stops. If the network stops are relatively evenly utilized, we can assume an average degree for each GST node, denoted as v, which is interpreted as the number of possible stops one can reach from any given stop. If the length of the concatenation of paths is l, and the length of the path a is defined as m_a, then the number of leaves in the subtree can then be defined as $u = l/(v^{m_a})$. This is because each GST child node of the root will have approximately l/v leaves in its subtree, and in the same way, every time we descend to a child node, the number of leaves is divided by v. Given the foregoing, the time complexity of the proposed algorithm can be expressed as $\mathcal{O}(l/v^{m_a})$ on average.

3.3.3.3. `Contains(a,X)`

The relation `Contains` retrieves all the paths from the set that are completely contained within a. According to Definition 6, it corresponds to all $x \in X$ such that `Contains`(a, x). In other words, it identifies all the paths that are subsequences of a. This relation is established when one of the following relations exists between a and another sequence: *Covers*, *Includes*, or *Equal*. This corresponds to the aggregated relation *Contains* in Figure 3.2.1.

Any node x in a GST represents a sequence. Navigating towards the root through the ancestors of this node allows us to traverse nodes representing prefixes of such a sequence. Therefore, having the ancestor y of x, each of the leaves in its subtree shares a prefix with the sequence represented by node x, which may be larger. If there exists an edge from y that starts with the end-of-sequence marker, leading to a node z, it indicates that there is a sequence that contains the prefix represented

by y as a suffix. If the suffix of a leaf in z starts right after an end-of-sequence character in the GST, it is possible to assert that the sequence represented by node y is a sequence that is entirely contained at the beginning of the sequence represented by node x. This way, it is possible to find all sequences that correspond to prefixes of the sequence represented by x. To find other sequences that are completely contained in x, the same procedure is used, starting from the node obtained through the suffix-link (explained in Section 3.2.3). This operation can be limited by the length of the shortest path in the set. This way, it is not necessary to explore nodes representing sequences with a shorter length. For example, in Figure 3.3.3, applying this operation from node 1 representing path $P1 = ABCD$ allows finding node 2 representing path $P2 = ABC$. Then, with the suffix-link operation from node 1, we arrive at node 3, where navigating through the ancestors leads to node 18, representing path $P3 = BCD$. Note that during this process, we do not pass through node 6 representing $P4$ because the edge between nodes 20 and 6 does not start with an end-of-sequence marker.

Algorithm 2 shows the instructions to solve the query `Contains`. From the *TRGST*, the node representing a is retrieved from *mapRoute2Node*, and the traversal goes towards the root as long as the represented sequence is longer than the shortest path in the set (Line 5 and line 14). During this traversal, the algorithm checks whether there are paths represented in the subtree using *markBranch* (Line 6) and *markNode* (Line 8) when end-of-sequence markers are encountered. If there are candidate results, they can be obtained from *mapNode2Route* (Line 9), including all these paths in the result.

This navigation is repeated for consecutive suffix-link operations starting at the node representing a (Line 16), as long as the length of its sequence is greater than the shortest path in the set (Line 3 and Line 16).

Algorithm 2 traverses the leaves using the suffix link operation, a number of times proportional to the length of the sequence a, previously denoted as m_a. For each sequence, the traversal proceeds to the root through its parent, which is also proportional to m_a. The parent node navigation occurs in constant time, while the time for the suffix link operation is denoted as SL and it depends on the actual implementation of the GST. In cases where a node represents one or more paths, the identifiers are extracted from *mapNode2Route* with a logarithmic complexity in

Algorithm 2 `Contains`$(a, TRGST)$

```
1: nodeA ← TRGST.mapRoute2Node(a)
2: limit ← nodeA.length() - TRGST.shortestSeqLength
3: for i ← 0 to limit do
4:   nodeB ← nodeA
5:   while nodeB.depth ≥ TRGST.shortestSeqLength do
6:     if TRGST.markBranch[nodeB] then
7:       nodeC ← nodeA.child(endSec)
8:       if TRGST.markNode[nodeC] then
9:         for all n from TRGST.mapNode2Route(nodeC) do
10:          result.add(n)
11:        end for
12:      end if
13:    end if
14:    nodeB ← nodeB.parent()
15:  end while
16:  nodeA ← TRGST.suffixLink(nodeA)
17: end for
18: return result
```

the size of the container[2], resulting in a complexity of $\mathcal{O}(\log n)$, where n represents the number of paths in the set. The size of the output is denoted as w, considering that extracting a path from *mapNode2Route* is performed a proportional number of times relative to the result. Consequently, the `Contains` operation demonstrates a worst-case time complexity of $\mathcal{O}({m_a}^2 + m_a \times SL + w \times \log n)$, where ${m_a}^2$ represents all the parent traversals, $m_a \times SL$ corresponds to the suffix link traversal, and $w \times \log n$ is the recovery of the paths in the output.

3.3.3.4. `Intersects(a,X,k)`

The `Intersects` query retrieves the identifiers of all the paths in the set that intersect with a in at least k consecutive elements. For each path in the query result, the length of the longest match between the sequence of such path and a is also returned. This topological relation is formalized in Definition 7 and returns all $x \in X$ for which `Intersects`(a, x, k) holds. This is similar to the problem of computing the $q-$ Maximal Exact Matches ($q - MEMs$) in a Suffix Tree [**?**]. It differs in that the search for MEMs does not need searching for shorter sequences once the maximal ones have been found.

[2]In practice, the *equal_range* operation is employed to determine elements associated with a key in a *multimap* (https://en.cppreference.com/w/cpp/container/multimap/equal_range).

To obtain the result of the longest common intersection between two elements in a GST, it is possible to use the information provided by the Longest Common Prefix (LCP) array. This array indicates the longest common prefix that exists between a leaf node and its immediate predecessor, which is the leaf node at the immediately preceding position. For example, in Figure 3.3.3, position 2 in the LCP array has a value of 3, indicating that there is a common prefix of length 3 between leaf node 2 and leaf node 1, which corresponds to the prefix ABC.

Then, given a node x of a GST, it is possible to find all suffixes that have a prefix of at least k elements in common with the sequence it represents through the navigation of the LCP array. Next, to identify all suffixes in this GST that share elements with the sequence of node x, the same idea described for the `Contains` operation can be applied, navigating the tree through successive suffix-links from node x while ensuring that the depth of the sequence satisfies the condition of being greater than or equal to k. At each of these nodes, the search through LCP navigation is applied. Recall that the leaf node i is also represented by the element at position i in the CSA. Then, $lcp[i]$ also represents the length of the prefix shared by suffixes $csa[i]$ and $csa[i-1]$.

Algorithm 3 starts from the node representing path a in the $TRGST$ and traverses towards one of the leaves in the subtree. It is again necessary to check if the parent of *nodeA* contains the complete sequence; if so, we work with this node (Line 2). This navigation is repeated for each suffix-link that can be followed starting from the node representing a, as long as the length of its sequence is greater than or equal to k, controlled by the loop on Line 9, where the value of i is updated based on the sequence length and the value of k. In each iteration, we switch branches using the suffix link operation (Line 25).

We include in the result those leaves for which the length of the common prefix is greater than or equal to k, using sequential traversals both to the right (corresponding to the *Rightward traversal*) and left (Line 13). The length of the intersection is obtained from the variable *matchLen*, which is created based on the depth of the GST node with which each loop starts (Line 12). Note that the match length cannot be directly obtained from the values in the LCP array because those values are associated with the sequences in the nodes, and they could be greater than the actual intersection length between the paths (Line 20).

Algorithm 3 `Intersects`$(a, TRGST, k)$

```
1: nodeA ← TRGST.mapRoute2Node(a)
2: if nodeA.parent().depth() == a.length() then
3:    nodeA ← nodeA.parent()
4: end if
5: ls ← nodeA.depth()
6: if ls < k then
7:    return null
8: end if
9: for i ← 0 to ls - k do
10:   leafId ← TRGST.getLeafId(nodeA)
11:   navigateId ← leafId
12:   matchLen ← ls - i
      {Leftward traversal}
13:   while matchLen ≥ k do
14:      suffixPos ← TRGST.csa(navigateId)
15:      routeId ← TRGST.idfromPos(suffixPos)
16:      if result[routeId] < matchLen then
17:         result[routeId] ← matchLen
18:      end if
19:      if TRGST.lcp[navigateId] < matchLen then
20:         matchLen ← TRGST.lcp[navigateId]
21:      end if
22:      navigateId ← navigateId - 1
23:   end while
      {Rightward traversal}
24:   ...
25:   nodeA ← TRGST.suffixLink(nodeA)
26: end for
27: return result
```

The drawback of this navigation is that it does not discard subsequences found in future visits to the suffixes. As a result, each of the already included results will be "found" again in the next branch that is verified. Hence, the results of this operation are stored in an *unordered_ map* structure, where the key corresponds to the IDs of the paths, and the value is the longest match with path a. With this approach, it is possible to filter out the duplicate results by checking the path ID before insertion (Line 17).

Algorithm 3 provides detailed instructions for traversing from the leftmost leaf of the reference node (Line 10). This traversal is repeated similarly to check towards the right side of the leaf from the reference node (Line 24). The same traversal could not be done sequentially since the length of the maximum intersection is not directly obtained from the LCP array. Instead, it is necessary to keep track of the maximum intersection (*matchLen*) from the reference node towards the endpoints in a non-increasing manner. It is not possible to perform the traversal from one end to the other even with knowledge of the range of leaves of the reference node, as the values of the LCP array may exhibit growth that does not necessarily represent the values of coincidence with the subsequence of a under evaluation.

The `Intersects` operation, similar to the previous one, traverses the subtrees of the tree by employing the suffix link operation a number of times proportional to the length of the sequence defined as m_a. For each branch, it examines each of the leaves that fulfill the condition of the k value. Previously, the number of leaves in a subtree has been defined as u. It is reasonable to assume that there are scenarios where the distribution of leaves is relatively proportional across each subtree within the structure. In such cases, there exists a value v corresponding to the average degree for each node. It is possible to indicate that in this scenario, the number of leaves does not depend on the path length m_a, but it depends on the value of k. Therefore, the number of leaves explored by the `Intersects` operation for each subtree is defined as $u = l/v^k$, where l is the length of the concatenation of paths. As a result, the time complexity of this operation is $\mathcal{O}((m_a - k) \cdot (l/v^k))$. This result indicates that with a higher value of k, the time required to solve the operation is considerably reduced. In the next section, we will see how this is also reflected in practice.

3.4. Experimental Evaluation

In this section we describe the experimental evaluation of the proposed data structure to answer topological relations between paths. The evaluation takes into account both the response time for answering queries about relations and the space usage of the proposed structure. In this assessment, three different datasets are employed to identify the impact of different parameters such as the number of paths in the set or the number of stops in the underlying network. As a baseline for comparison, we use an approximation based on well-known pattern matching algorithms, which we describe below.

3.4.1. *Baseline*

Since paths are represented as sequences, it is possible to answer the queries by using efficient string processing algorithms. The *Baseline* uses different algorithms depending on the relation of interest for the query. All queries yield a list with the identifiers of all paths in a set S that satisfy the relation with respect to a reference path denoted as a, as described in Section 3.3.3. Due to this, all queries follow a global approach where the specific algorithm for the studied relation is executed for each path $b \in S$ with respect to the target path a. The specific algorithms for each query are described below:

- *Equals*(a, X): Equality between two sequences is defined by the condition that both sequences have the same length ($|a| = |b|$) and that each corresponding element of sequence a is equal to the corresponding element of sequence b ($a_i = b_i$, where $1 \leq i \leq |a|$). This is solved by sequentially scanning both paths simultaneously until a mismatch is encountered. If there is no mismatch and the lengths are the same, then an *Equals* relation exists.
- *Within*(a, X): In this operation, the expected result comprises all paths that completely contain the path a. Thus, a first filter is applied based on the path length: a sequence with a length shorter than a cannot contain it. After this filter, the containment verification is equivalent to performing a pattern matching search in a text, and the algorithm used is *Knuth, Morris & Pratt*[3] (KMP) [136].

  The KMP algorithm searches for a pattern with a pre-processing time

[3] https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/

complexity of $\mathcal{O}(m)$, where m is the length of the pattern. During this preprocessing step, information about pattern sequence matches is computed, which is later utilized to efficiently handle partial matches between the text and the pattern, avoiding unnecessary re-examination of previous positions in the text.

As a result, the pattern search has a time complexity of $\mathcal{O}(n+m)$, where n is the length of the text, and m is the size of the pattern. For this operation, the path b is treated as the text, and the path a as the pattern, resulting in determining whether path b contains path a.

- *Contains*(a, X): It is equivalent to the previous operation, but a and b are exchanged, making b the pattern and a the text.
- *Intersects*(a, X, k): To answer this query, the *Baseline* implementation uses the Longest Common Subsequence (LCS) dynamic programming algorithm[4] [29]. This algorithm is based on the fact that given two sequences of length n and m, there are $n \times m$ subproblems. If it is known that there is a match between element i and element j, this match can be included in some immediately previous match. Thus, this problem is solved in $\mathcal{O}(nm)$.

In order to save some space, the implementation of the *Baseline* structure uses the `int_vector` structure available in the `sdsl` library (see Section 3.4.2) to represent each path in the set. This structure allows storing a vector of integers, where each integer uses $\lceil \log_2 p \rceil$ bits, where p is the largest number in the set.

3.4.2. Experimental Environment

All the implementations reported in this section were developed using the C++ programming language, compiled with GCC 11.3.1, and executed on the Rocky Linux operating system[5] version 9.1 (Blue Onyx). The server used for the experiments is equipped with an Intel®Xeon®Silver 4316 CPU @ 2.30GHz processor and 251 GB of RAM.

The reported times were obtained using the `clock()` function from the `ctime` library[6], which represents the processor time consumed by the running program. All reported times only consider the interval required for the execution of the

[4] https://www.geeksforgeeks.org/longest-common-substring-dp-29
[5] https://rockylinux.org/
[6] https://cplusplus.com/reference/ctime/clock/

operation that produces the result. The loading of input data and reading of query data files are not included in these times.

Both the proposed data structure and algorithms in this implementation – including the *Baseline* described in Section 3.4.1– make use of elements from the Succinct Data Structure Library (SDSL) by Simon Gog[7]. The source code and instructions to reproduce our experiments are available on GitHub[8].

Implementation Details of the *TRGST*

The implementation of the *TRGST* utilizes the following elements, some of which, as mentioned earlier, are from the *sdsl* library:

- `cst_sada`: Implementation of the compressed suffix tree [117].
- *mapRoute2Node*: An array of integers where the i-th position indicates the ID of the node that represents the i-th path.
- *mapNode2Route*: A `multi-map` structure that stores pairs (x, y), where x corresponds to the ID of a node and the value y corresponds to the ID of a path.
- *markEndSequence*: A `sd-vector` [99, 47] that corresponds to a compressed bitmap unary encoding the sizes of the sequences in the same order they were concatenated.
- *markNode*: A `sd-vector` of the size of the number of nodes in the GST. Each node is identified by a position in the `sd-vector` that corresponds with its identifier in the GST. The value stored in such position is a 1 if the node represents an entire path and a 0, otherwise.
- *markBranch*: A `sd-vector` equivalent to the previous one, but the value 1 indicates that there is at least one path in the subtree induced by the node.

3.4.3. Experimental Data

In this section, we describe the three datasets used in our experiments. The largest dataset, `trips_madrid`, has paths generated using the real transportation network of Madrid, and it was used in the experimental evaluation of [14, 13]. The second dataset, named `stops_trips_madrid`, was derived from the former by restricting the number of stops on the network to evaluate the impact of this parameter.

[7] https://github.com/simongog/sdsl-lite
[8] https://github.com/cquijadafuentes/TGRST_paper_pub

Finally, the third dataset, `taxi_shang`, is a real-world dataset that was used in the evaluation of the COMPRESS framework [58], which is a method for trajectory compression.

In this work, each dataset is a derived subset of the original dataset, adhering to the following rule: sequences in the dataset can be represented by a simple line segment that does not self-intersect, in accordance with the definition of simple lines in the 9-IM of Egenhofer [45].

3.4.3.1. `trips_madrid` Dataset

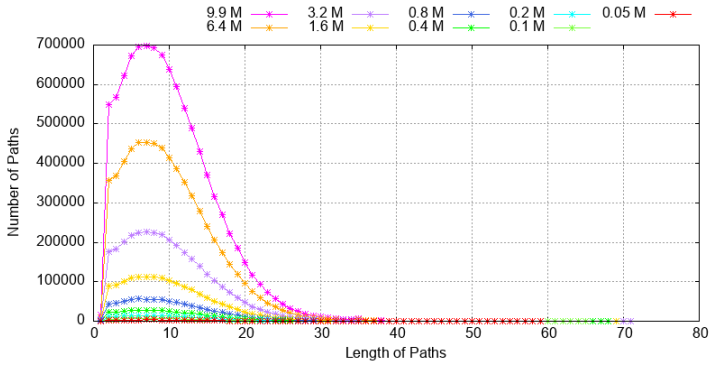
The `trips_madrid` dataset used in the experimentation corresponds to a subset of data generated on the transportation network of Madrid[9], which was also used in the experimental evaluation performed in [14, 13]. The real-world transportation network consists of 11,021 stops and 1,048 lines, with an average of 27.07 stops per line. The original dataset consists of almost 10 million trips generated synthetically on this network, considering some movement patterns based on real records, as well as schedules, duration, days of the week, etc. The sizes reported in our graphs correspond to approximations of the number of paths in the dataset. For instance, the data file labeled as 9,9M contains $9,922,566$ paths, which are all the paths in the dataset that satisfy the simple lines definition in [45].

This dataset is used to evaluate the impact of variations in the number of paths in the experimentation. The dataset consists of 9 files with 50K, 100K, 200K, 400K, 800K, 1.6M, 3.2M, 6.4M and 9.9M paths. The paths in all the files, except for the last one, were randomly selected from the original data file. Figure 3.4.1 (a) shows the distribution of the sequences based on the lengths of the paths in the dataset. It is possible to notice that all the files in the `trips_madrid` dataset contain numerous short paths, concentrated within the range of 1 to 12 stops. On the other hand, Figure 3.4.1 (b) illustrates that all the stops are uniformly utilized, albeit with a few outliers.

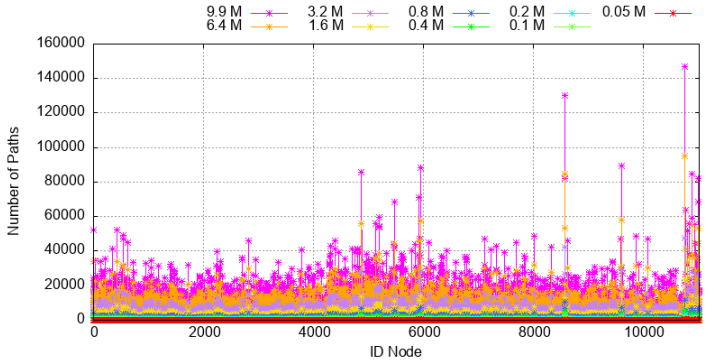
3.4.3.2. `stops_trips_madrid` Dataset

This dataset was created to evaluate the impact of the number of stops in the network and it is a variation of the previous one, where the number of paths in

[9] https://lbd.udc.es/research/XCTR/xctrdata.7z



(a) Number of paths per path length



(b) Number of paths per network stop

Figura 3.4.1: Characterization of the `trips_madrid` dataset.

the set is fixed, and the variable is the number of stops in the network. There are five files, each containing approximately 0,8 million paths. In each file, each sequence is created using the modulo operation on one of the original sequences in the dataset `trips_madrid`. The modulo operation was chosen because the IDs of each individual path are typically similar. Hence, the modulo operation results in few collisions between IDs of the same sequence, while also adhering to the rule of non-self-intersecting paths. The number of stops in each file is $2K$, $4K$, $6K$, $8K$ and $10K$.

3.4.3.3. `taxi_shang` Dataset

This dataset is composed of real world trajectories data from the largest taxi company in Shangai at it was used in the evaluation of the COMPRESS framework [58]. The framework takes GPS data and maps the trajectories onto a given road network. Subsequently, the matched trajectories are decomposed into spatial paths and temporal sequences, which are compressed separately using various compression techniques. In our evaluation we use the spatial paths obtained after

the decomposition of the matched trajectories.

This dataset is composed by only one file with $100K$ paths, but after the *simple line* rule is applied, it becomes a subset of $64,574$ paths where the number of stops is $62,584$. The longest path is a sequence of length 348, and the length distribution of the paths in the dataset is similar to the curve shown in Figure 3.4.1(a) for the `trips_madrid`. Although it is a small dataset, it allows us to observe that the results obtained with the `trips_madrid` dataset are consistent in this real-world dataset, which was also used in the evaluation of a trajectory compression method.

3.4.4. Time Evaluation by Number of Paths

In this section, we present a time performance evaluation of our proposal that shows its scalability with respect to the size of the dataset (defined by the number of paths) using the `trips_madrid` dataset. Subsequently, the average time for each of the previously described operations is reported. Query-sets were generated by randomly selecting $1,000$ paths ids from each data file. The average time reported in each graph is obtained by querying the corresponding file three times and calculating the average.

3.4.4.1. `Equals`

Figure 3.4.2 presents the results for the `Equals` operation, where the *TRGST* implementation outperforms the *Baseline* in all cases. Notably, the *Baseline* implementation exhibits a significantly greater increase in processing time with larger data files, particularly when comparing the first file with 50 thousand elements to the last file with 3.2 million elements. This increase is approximately two orders of magnitude for the *Baseline*, while it is about one order of magnitude for the *TRGST*. This is mainly due to the way it is solved, as it only needs to query the results directly from the additional structures of the *TRGST*. On the other hand, the *Baseline* implementation needs to traverse and compare all sequences in the set and it takes more time, even when the initial filter for finding an equal relation is based on the length of the path, which is very fast.

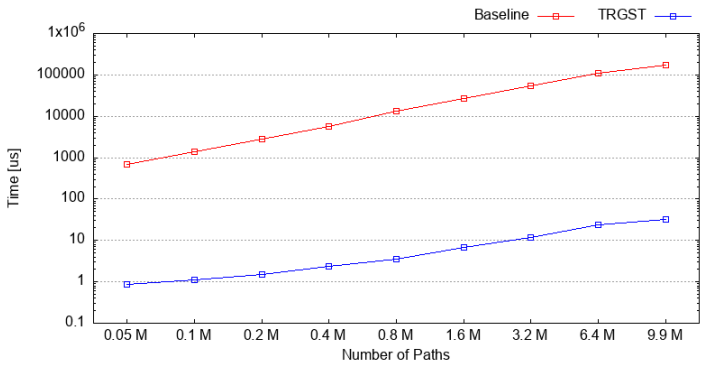


Figura 3.4.2: Time performance of `Equals` on the `trips_madrid` dataset with *random* queries.

3.4.4.2. `Within`

Figure 3.4.3 displays the execution time of the `Within` operation. The proposed data structure consistently outperforms the *Baseline* approach, maintaining lower execution times across all reported cases. Interestingly, the difference between these two lines remains relatively constant, as both increase over time while consistently maintaining a distance from each other. The factor between them ranges from 2.48 to 3.42, with an average of 3.08.

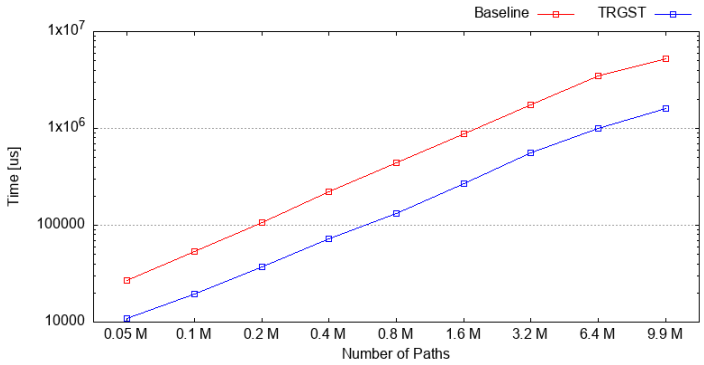


Figura 3.4.3: Time performance of `Within` on the `trips_madrid`dataset with *random* queries.

3.4.4.3. `Contains`

In Figure 3.4.4, it is possible to see the results of the `Contains` operation. The *TRGST* implementation begins with lower performance compared to the *Baseline*, exhibiting a difference of approximately one order of magnitude. However, the *Baseline* approach demonstrates a rapid increase in execution time as the number of paths increases. On the other hand, the *TRGST* implementation maintains a

relatively constant performance that appears to be unaffected by the growth in data size.

Notably, as the number of paths reaches between 200,000 and 400,000, the performance lines cross each other, and the *TRGST* consistently outperforms the *Baseline* approach in terms of execution time for the larger data files. Towards the end of the graph, the *TRGST* exhibits a significant advantage of nearly one order of magnitude over the *Baseline*, highlighting its superiority in handling larger datasets.

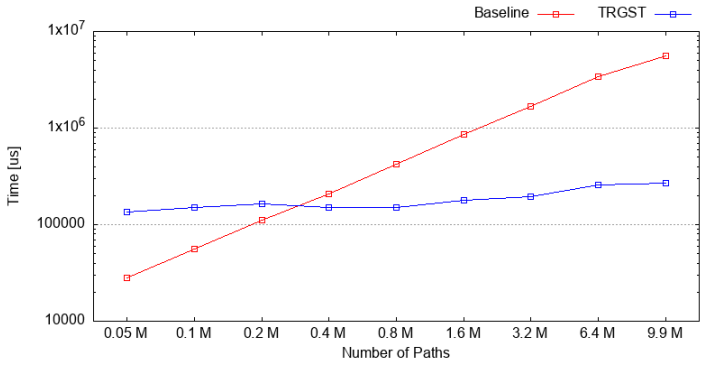


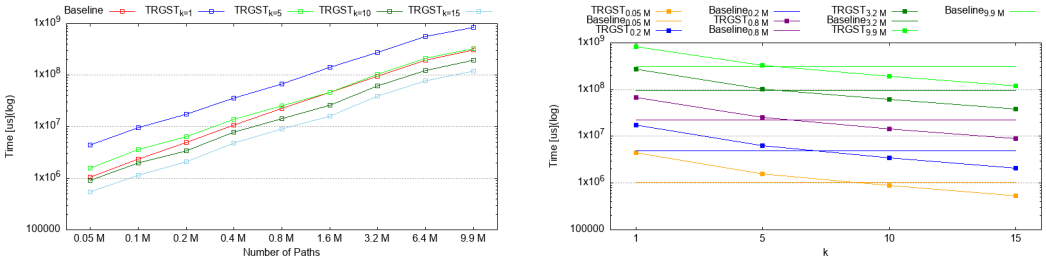
Figura 3.4.4: Time performance of `Contains` on the `trips_madrid` dataset with *random* queries.

3.4.4.4. `Intersects`

The execution times for the `Intersects` operation are displayed in Figure 3.4.5 with two graphs, each of which focuses on a specific aspect of the algorithm's behavior. For this operation, a smaller number of queries and repetitions were used due to the time required to obtain the experimental results, with 50 queries and only one repetition. This reduction in the number of executions should not have a negative impact on the accuracy of the reported times, as the proportion of error that is sought to be corrected with high numbers of queries and repetitions is small compared to the total time taken by the queries. Since queries in this operation are sensitive to the length of the input paths, each query file contains the longest paths, allowing the algorithms to obtain more results.

Figure 3.4.5(a) presents the results for different dataset sizes, where the value of k of the different lines represents the minimum number of consecutive matching elements between paths. Note that the *Baseline* was executed with $k = 1$, as it is not influenced by this parameter. The *TRGST* implementation with $k = 1$

always takes longer than the *Baseline* implementation, and it can be observed that the difference tends to persist as the dataset size increases. For $k = 5$, after the 1,6M paths, the points of *TRGST* are nearly identical to those of the *Baseline*. Meanwhile for $k = 10$ and $k = 15$, the proposed data structure consistently outperforms the *Baseline* across all dataset sizes.



(a) Time by dataset size (b) Time by value of k

Figura 3.4.5: Time performance of `Intersects` on the `trips_madrid` dataset with *random queries*. (a) A comparison of time between the *Baseline* (with $k = 1$) and the *TRGST* (with $k = [1, 5, 10, 15]$) based on the size of the paths set. (b) A comparison of time between the *TRGST* and the *Baseline* across various dataset sizes concerning the minimum length of intersection, denoted as k.

Figure 3.4.5(b) shows horizontal lines to represent the time for the *Baseline* instances, as the reported time for $k = 1$ in the *Baseline* implementation is used for all values of k. On the other hand, the *TRGST* implementation does benefit from an increase in the value of k, as it reduces the exploration of subtrees. This behavior is consistent with the description in Section 3.3.3.4, where the time complexity depends on the value of k and is expressed as $\mathcal{O}((m_a - k) \cdot (l/v^k))$.

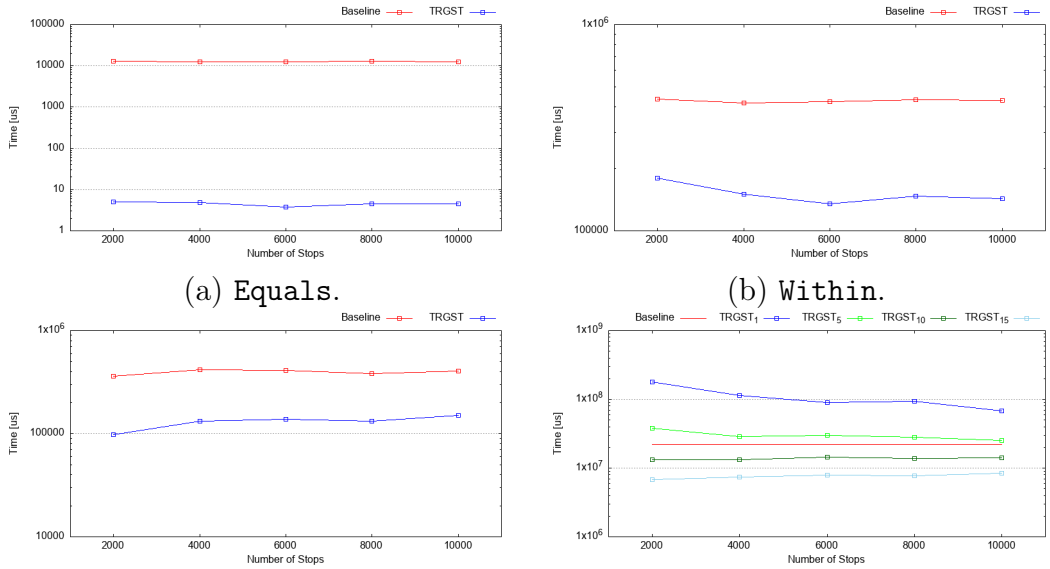
Figure 3.4.5(b) demonstrates that for values of $k \leq 5$, the time used by *TRGST* is greater than the time of the *Baseline* implementation for all the dataset sizes. However, for values of $k \geq 10$, the time of *TRGST* is lower than the time of the *Baseline* implementation, also across all dataset sizes. In other words, as the value of k increases, the *TRGST* requires less time to answer `Intersects` queries. It is also worth mentioning that the value of k for which the proposed structure starts to outperform the *Baseline* decreases as the dataset size increases.

3.4.5. Time Evaluation by Number of Stops

In this section, we evaluate the influence of the number of stops of the underlying network on the query time using dataset `stops_trips_madrid` described in the

Section 3.4.3.2. Figure 3.4.6 shows the results for the four relation. In all the cases, the *Baseline* is slower than the *TRGST* time, except for the `Intersects`, where the *Baseline* performs better than the *TRGST* for values of $k = \{1, 5\}$.

These results are consistent with those presented in Section 3.4.4, as the curves in graph maintain a similar order of magnitude for the same data file size. These graphs exhibit an almost constant trend in both implementations, indicating that the number of stops does not have a significant impact on the time performance of both the *TRGST* and the *Baseline*.



(a) `Equals`. (b) `Within`.

(c) `Contains`. (d) `Intersects`.

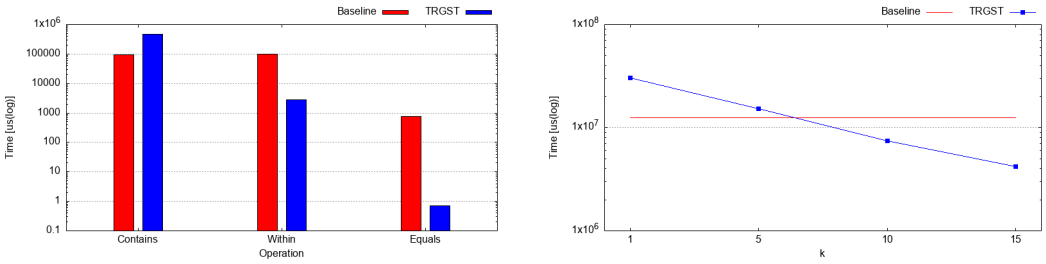
Figura 3.4.6: Time performance on the $800K$ trips data file when varying the number of stops.

3.4.6. Time Evaluation on the `taxi_shang` dataset

In this section, we present the query time evaluation on dataset `taxi_shang` (described in Section 3.4.3.3), which, although smaller than `trips_madrid`, provides real data on both the network and the paths. Figure 3.4.7 shows the results of this evaluation.

Figure 3.4.7(a) indicates that the `Contains` operation is slightly faster using the *Baseline*, while for the `Within` and `Equals` operations the *TRGST* performs better. Regarding the result for `Contains`, it is important to note that in dataset `trips_madrid`, for small file sizes (equivalent to the size of `taxi_shang`), the *Baseline* was also faster than the proposed structure. Therefore, for a larger

number of paths in this dataset, it is expected that the structure will be faster than the *Baseline*. On the other hand, in Figure 3.4.7(b) the `Intersects` operation time is compared, where the *TRGST* performs better when the value of k is greater than 5; otherwise, the *Baseline* is faster. A similar behavior to what was observed for this operation with dataset `trips_madrid`.



(a) `Contains`, `Within`, and `Equals`. (b) `Intersects`.

Figura 3.4.7: Time performance of relation queries on the `taxi_shang` dataset.

In general, the behavior of the operations on this dataset is similar to that demonstrated in dataset `trips_madrid`, specifically for a file size of 0,05M paths. For the `Equals` operation, there are no differences; however, the other operations show a change in the order of magnitude of the times. The `Within` operation reduces the times by almost an order of magnitude and increases the difference between the two approaches, while `Contains` slightly reduces its execution time, maintaining its behavior. Finally, the `Intersects` operation maintains the trend between both approaches, but also exhibits an increase by an order of magnitude in the execution time. These changes in the execution times are associated with the length of paths in the datasets. The `trips_madrid` dataset, for example, contains path lengths ranging between 1 and 80, while the paths in the `taxi_shang` dataset span lengths from 1 to 348.

3.4.7. Space Evaluation

Next, we present an evaluation of the space required by the TRGST. It is important to note that when utilizing CDS to construct self-indexes, the objective is to obtain representations that occupy less space than the plain original data (typically a space proportional to that data but in compressed form) and respond to queries in a time similar to that of classical indexes. Those classical indexes necessitate additional space on top of the data themselves, which must also be stored.

The two primary elements of comparison in this evaluation are the *TRGST* and the *Baseline*, which are the only two existing alternatives to address the queries studied in this work. In this section, the space usage of both methods is determined using the `size_in_bytes` function provided by the *sdsl* library. Recall that the *Baseline* representation uses a sequence stored in `int_vector` arrays of such library that utilize the *bit_ compress* operation to use $\lceil \log_2 p \rceil$ bits for each identifier, where p is the largest stop ID.

To provide context for the space required by the *TRGST*, we provide some additional elements of comparison. First, a naive solution could be to precompute the answers to any possible query. Obviously, this approach would be very fast to solve the queries, but its space requirements make it unpractical. To illustrate this, we computed the space required to store the answers to the four relations in the smallest file of the `trips_madrid` dataset. Considering only the $50,000$ paths in such file, the space usage is already 2,3 [GB]. Hence, this approach is unfeasible. As we mentioned above, a self-index using CDS usually requires space in between the plain data and the compressed data. We compressed our paths with gzip to illustrate this comparison[10]. Finally, a classical non compact index equivalent to our approach could be a Suffix Array (SA) combined with a Longest Common Prefix (LCP) array. Such alternative requires two arrays of the size of the concatenated paths, in addition to the paths themselves. Figure 3.4.8 illustrates the total space used by the *TRGST* structure compared to the *Baseline* representation, gizp, and a SA+LCP+Seq.

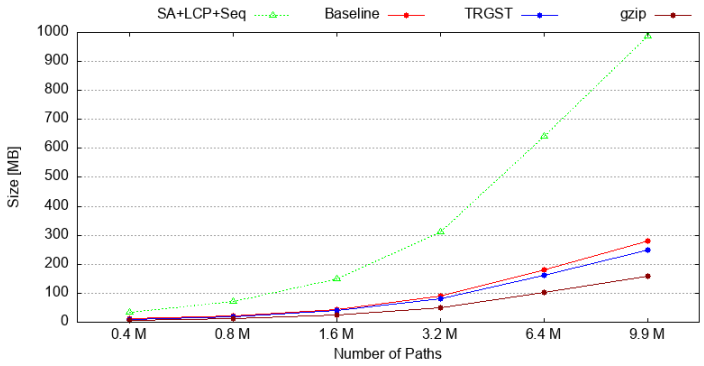


Figura 3.4.8: Size of different alternatives for the `trips_madrid` dataset.

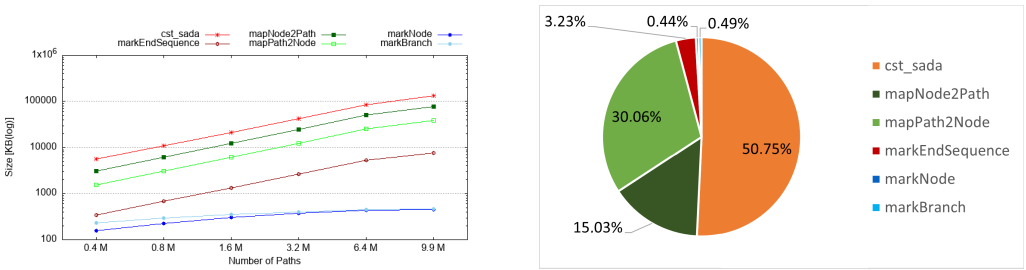
A first conclusion from this graph is that, as expected, the *TRGST* requires space

[10]Note that gzip is just a compressed format and it does not allow any kind of query without decompressing the data.

in between the plain data (which is approximately the space reported by the *Baseline*) and a compressed representation of the data (gzip). In other words, in space proportional to the compressed paths, the *TRGST* provides efficient query capabilities. A non compact version with similar query capabilities, such a SA+LCP, would require almost 5 times more space. Note that such space is allocated solely for the core structure. To support the relations studied in this work, additional data structures would be necessary, thereby adding some extra space.

As the previous descriptions indicate, the *TRGST* is composed of various data sub-structures. Figure 3.4.9 provides a detailed breakdown of the space usage by the proposed *TRGST*, indicating the space utilized by each of its sub-structures. In this figure, it can be observed that the compact suffix-tree representation (cst_sada in the graphs) occupies the largest amount of space, followed by the maps that facilitate navigation between the nodes representing the paths and their identifiers, and vice versa.

The space consumption of the second heaviest sub-structure can be minimized by substituting it with a bitmap that flags the CSA with suffixes representing complete sequences. This collaborative approach with *markEndSequence* could enable the recognition of stop-associated paths through leaf navigation. Nevertheless, opting for this space reduction could affect processing time, which, in this proposal, is given precedence over space efficiency.



(a) Size for each datafile. (b) Size proportion (3.2 million paths).

Figura 3.4.9: Size used by the sub-structures that compose the *TRGST* over `trips_madrid` dataset.

Since the other two datasets are small, the space evaluation is not as relevant. However, for the completeness of the article, we also report them. The *Baseline* for the `taxi_shang` file requires 4,39 [MB], while the *TRGST* uses 4,46 [MB]. In the `stops_trips_madrid`, the *Baseline* reports spaces between 19,94 and 21,71 [MB]

for the different files, while the *TRGST* requires between 20,21 and 20,75 [MB] of space. These space values are consistent with those reported for the small files in `trips_madrid` dataset.

3.5. Conclusions and Future Work

In this work, the *TRGST* data structure is introduced, designed to efficiently handle topological queries within a set of paths. These paths are represented as sequences of stops of a network, and the queries enable the extraction of path subsets exhibiting specific topological relations with respect to a chosen path from the set. Furthermore, detailed descriptions of the proposed algorithms for handling topological relations between paths are provided. These algorithms have been described, and their time complexity has been analyzed.

At the core of *TRGST* is a compact version of the Generalized Suffix Tree (GST), the principal sub-structure encompassing all the sequences representing the paths set. This central element collaborates with auxiliary sub-structures, such as bitmaps, multi-maps, and arrays, to enhance GST performance by facilitating rapid node retrieval and navigation. The synergy among structures empowers *TRGST* to adeptly manage topological relations among paths.

The experimental evaluation includes an analysis of execution time with three different datasets, where the *TRGST* is benchmarked against a *Baseline* implementation that utilizes established text processing algorithms. As for space, in addition to comparing the space used by the proposed structure with respect to the *Baseline*, we also report the space occupied by the paths compressed with a standard compressor (gzip), and the space that the non-compact versions of the main components of the GST would occupy on top of the paths themselves. Similar to what occurs with self-indexes based on CDS proposed for other domains, such as text, the TRGST is introduced as a structure that, in less space than the original data (and proportional to the size of such data in compress form), efficiently supports operations. It should be noted that this represents a significant advantage over traditional indexes that occupy additional space beyond what the original data already requires.

The results highlight the ability of the *TRGST* in containment queries, an accomplishment attributed to its skill in comparing prefixes and suffixes within

sequences. A detailed analysis of the intersection operation reveals the pivotal role of parameter k (which defines the minimum number of consecutive elements that must intersect to satisfy the query), affecting execution time by regulating leaf node traversal in candidate path branches.

Through our experimental evaluation, the graphs reflect an encouraging trend: larger datasets correspond to improved results for topological relations within the *TRGST* structure. This scalability underscores the GST's ability to capitalize on data repetitiveness to enhance performance. This advantage applies to both the space requirements of the *TRGST* and the time required to solve queries. It can also be deduced from the experimental evaluation that the number of stops does not have a significant impact on the reported times for either of the two evaluated approaches.

Looking ahead, our work calls for an extension that embraces the temporal dimension in both trajectory representation and analysis. This augmentation promises a more comprehensive exploration of spatio-temporal information. Another avenue for future exploration is the development of a structure and algorithms capable of addressing similar queries within trajectories represented in a free-space model. Additionally, this work holds the potential for practical applications in public transportation networks. The efficient handling of paths and their topological relations could aid in optimizing bus routes, tracking vehicle movements, and improving overall transportation efficiency. In this domain, it is also possible to define new types of queries, such as counting and other aggregation queries, that can be supported by our structure. Finally, the inclusion of the temporal dimension in trajectory representation opens up new avenues for enhancing safety and efficiency in transportation systems.

Capítulo 4

Formalization and Scalable Processing of Spatially-Embedded Time Series

A continuación se desarrolla el segundo artículo que trabaja sobre secuencias de datos espacio-temporales, y que actualmente está bajo revisión [110]. Este trabajo tiene fuerte enfoque en la caracterización de los conjuntos de datos que se utilizan para la experimentación. Dichas características guian la confección de las estructuras de datos que se presentan en este trabajo. Actualmente el artículo ha sido enviado para ser sometido a evaluación, y determinar si es apto para su publicación en revista científica.

Abstract

Time series play a crucial role in numerous scientific and technological domains. In many cases, these series do not come from completely independent sensors, but are associated with specific locations in a space—such as geographic space—inducing common spatial patterns. In this work, we study different datasets sharing this characteristic: meteorological records over a spatial grid, traffic sensors distributed across the city of Madrid, and electroencephalograms where sensor positions reflect the distribution of electrodes on the human scalp. We provide a general formalization of the key properties of these datasets with the aim of facilitating their analysis in similar contexts. We evaluate properties such as temporal and spatio-temporal autocorrelation, and propose variants of compact data structures

adapted to each domain type. Compared to state-of-the-art approaches, our proposals show competitive results, particularly highlighting spatial efficiency for dense datasets with high spatio-temporal locality, as well as query times for some cases of window-based access.

4.1. Introduction

Rapid technological advances have enabled the development of diverse types of sensors that generate large volumes of data through continuous sampling stored as time series. In many cases, these sensors form part of a network of devices in which all elements collect samples at regular, synchronized time intervals. In such scenarios, the data often exhibit not only a temporal dimension but also a spatial one, determined by the location of each sampling point (i.e. each sensor). Moreover, time series from sensors that are spatially close or functionally related may contain similar value sequences. This scenario, which we refer to as Spatially-Embedded Time Series, presents an opportunity to design data structures capable of storing such data efficiently. Two key aspects motivate this work: (1) the continuous measurement and storage of data by these devices, and (2) the potential redundancy in sensor readings arising from similarities in their spatial arrangement or other shared characteristics.

Those two characteristics provide a favorable context for a particular class of data structures known as Compact Data Structures (CDS), which are designed to handle large volumes of information by exploiting regularities, duplicated values, or similarities present in the data. CDS allow storing the data along with auxiliary structures using even less space than the original dataset, while still supporting queries without requiring decompression [95]. These structures are designed to exploit specific properties of the data, enabling the implementation of various operations – often requiring more computational steps than a classical representation, but offering a much more efficient use of memory. In some cases, it is possible to store the entire dataset in main memory using a CDS, whereas the original representation would not fit and would instead require access to secondary storage. This scenario enables query results to be retrieved with response times that can differ by several orders of magnitude. Compact data structures have been successfully applied in diverse domains, such as web graphs [12], social network

graphs [7], temporal rasters [31], images [83], DNA sequences [11], decision trees [32, 86, 107], and many others.

As previously mentioned, CDSs take advantage of certain inherent data properties, such as different forms of entropy and repetitiveness. Although less frequently used in this domain, different types of auto-correlation present in the data can also help in designing CDSs. Auto-correlation is an indicator of the sequential dependence within a set of elements, and it can occur across different dimensions: temporal, spatial, and spatio-temporal. Its usefulness lies in indicating whether *nearby* (where the definition of *nearby* depends on the domain and dimension) elements within the dataset are similar to each other, exhibit opposite behavior, or if no clear pattern exists.

The hypothesis motivating this proposal is as follows: Given a set of time series with a certain degree of similarity among them, it is possible to divide the set of time series into sub-sets, allowing the representation of all series within the same sub-set with respect to the same reference series. In such cases, if a set of time series exhibits a high degree of spatio-temporal auto-correlation, it is possible to form subsets that allow reference-based representations among their closest elements, yielding good compression results.

The main results of this work are: (1) the classification of data based on the analysis of temporal and spatio-temporal auto-correlation of three time series datasets, which include a meteorological dataset arranged in a grid, a set of electroencephalograms (EEG), and a group of traffic sensors from the city of Madrid. The primary difference among the datasets lies in how the time series they contain can be related. The meteorological data exhibit an evident proximity-based relationship among their elements, unlike the other two datasets where neighborhoods are defined by different criteria. For example, EEG sensors can be grouped based on the codes assigned to them or based on their location on the scalp. On the other hand, traffic sensors may be located close to each other but on roads with opposite directions, so their time series may not present similarities over time. Thus, for the meteorological data, location is the only available criterion for defining neighborhoods, whereas for EEG and traffic sensors, associated codes enable neighborhood definitions beyond mere spatial proximity. (2) A data structure is presented for representing spatially-embedded time series in two variants: one designed for grid-based data, where autocorrelation is defined

by the physical proximity of measurement points, and another that groups elements based on similarities established through domain-specific relationships. (3) Experimental results are presented using the datasets described above, comparing the proposed structure with state-of-the-art representations appropriate for each domain, including a compact grid-based data structure and integer value encoders.

The structure of this article is as follows: Section 4.2 presents the related work, including a description of auto-correlation indices and a data structure for spatio-temporal data. Next, the methodology used is described, followed by Section 4.3, which provides a description of the datasets and their preprocessing. Then, in Section 4.4, the proposed data structure is presented in two variants. For each variant, the strategy underlying its design is detailed, and the corresponding experimental results are shown. Finally, Section 4.6 summarizes the results of the work, discusses possible explanations, and outlines potential future research directions.

4.2. Background and Related Work

Large volumes of data are currently generated continuously from various sources, including general-purpose sensors, mobile devices, and even computational models.

An important characteristic in the description of a dataset is its auto-correlation. Auto-correlation describes the extent to which observations of a variable are correlated with themselves over spatial [79], temporal [73], or spatio-temporal lags, implying that observations close to each other in space and/or time tend to exhibit similar values. Although spatial auto-correlation in data matrices has been widely studied, its application toward achieving efficient space-saving representations presents a valuable research opportunity.

The following subsections describe key concepts related to auto-correlation across different dimensions, as well as a data structure for the representation of spatio-temporal data.

4.2.1. Auto-correlation Indices

The phrase that best captures the essence of auto-correlation indices is Tobler's first law of geography [127]: *everything is related to everything else, but near things*

are more related than distant things. Following this idea, auto-correlation measures make it possible to assess the degree of similarity among data elements based on a given criterion of interest. This criterion is not restricted solely to spatial proximity. It is also highly likely that two samples taken close in time will be more similar than two samples collected at distant time instants.

Below, several auto-correlation indices used for characterizing time series are described. First, the Pearson product-moment correlation coefficient and its modification for determining temporal auto-correlation are introduced; then, Moran's Index, which is used to determine spatial auto-correlation; and finally, a proposed index for evaluating spatio-temporal auto-correlation (STA) in raster time series. These three indices allow identifying auto-correlation by establishing three possible situations: positive auto-correlation, where elements with high values are close to other elements with similarly high values, and vice versa; negative auto-correlation, where elements with high values are close to elements with low values, and vice versa; and finally, the absence of auto-correlation, where an element may be surrounded by both high and low values without following a specific pattern.

4.2.1.1. Temporal Auto-correlation: Pearson Product-Moment Correlation Coefficient

For the evaluation of temporal auto-correlation, the Pearson product-moment correlation coefficient is used [33]. This statistical index assesses the linear relationship between two variables [28]. The Pearson coefficient provides values in the numeric range $[-1, 1]$, where 1 corresponds to positive auto-correlation, 0 indicates absence of auto-correlation, and -1 denotes negative auto-correlation. It is available in the Python *numpy* library as *corrcoef*[1].

$$r = \frac{\sum_i^N (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_i^N (X_i - \bar{X})^2}\sqrt{\sum_i^N (Y_i - \bar{Y})^2}} \tag{4.2.1}$$

Equation 4.2.1 describes r as the centered and standardized sum of the cross-product of two variables [78]. This coefficient can be applied to a single time series with a temporal lag between the compared elements [8]. Equation 4.2.2 shows the

[1] https://numpy.org/doc/2.2/reference/generated/numpy.corrcoef.html

formula for determining temporal auto-correlation on a time series with a lag of k elements.

$$r_k = \frac{\sum_i^N (X_i - \bar{X})(X_{i+k} - \bar{X})}{\sqrt{\sum_i^N (X_i - \bar{X})^2}\sqrt{\sum_i^N (X_{i+k} - \bar{X})^2}} \tag{4.2.2}$$

4.2.1.2. Spatial Auto-correlation: Moran's I

Moran's Index [89] is one of the most popular spatial auto-correlation indices. It allows identifying the degree of similarity present in a dataset based on the spatial distribution of the values representing the data. The index values range from $[-1, 1]$, where values close to 1 correspond to positive auto-correlation, values near -1 indicate negative auto-correlation, and values close to 0 suggest that the dataset does not exhibit auto-correlation.

The interpretation of auto-correlation indicates the tendency of neighboring cells within the dataset. In cases of positive auto-correlation, cells with high values are surrounded by cells that also have high values, and vice versa. Negative auto-correlation implies that a low-value cell is surrounded by high-value cells, and vice versa. Finally, when no auto-correlation exists, cells may be surrounded by both high and low values simultaneously, without any predominant relationship between the values of neighboring cells.

Equation 4.2.3 corresponds to the formula for obtaining Moran's spatial auto-correlation index I. In this equation, X denotes the dataset of size N, where each element is a value. Each cell X_i is either a neighbor or not of another cell X_j, as indicated by the neighborhood matrix W of size $N \times N$. If $W_{ij} = 1$, element X_i is a neighbor of element X_j. If $W_{ij} = 0$, then elements X_i and X_j are not neighbors. There are two restrictions: (1) $W_{ii} = 0$ for any $i = [1, N]$, meaning that an element is not considered a neighbor of itself, and (2) for every element in X there is at least one neighbor ($\forall i, \sum_{j=1}^N W_{ij} \geq 1$). The average of the dataset elements is represented by $\bar{X}$.

$$I = \frac{N}{\sum_i^N \sum_j^N W_{ij}} \frac{\sum_i^N \sum_j^N W_{ij}(X_i - \bar{X})(X_j - \bar{X})}{\sum_i^N (X_i - \bar{X})^2} \tag{4.2.3}$$

Spatial neighborhood can be determined using different criteria, depending on

the context of the spatial analysis [100]. Some of these contiguity-based criteria are shown in Figure 4.2.1. The examples in this figure correspond to criteria that define neighborhood strictly by contiguity.

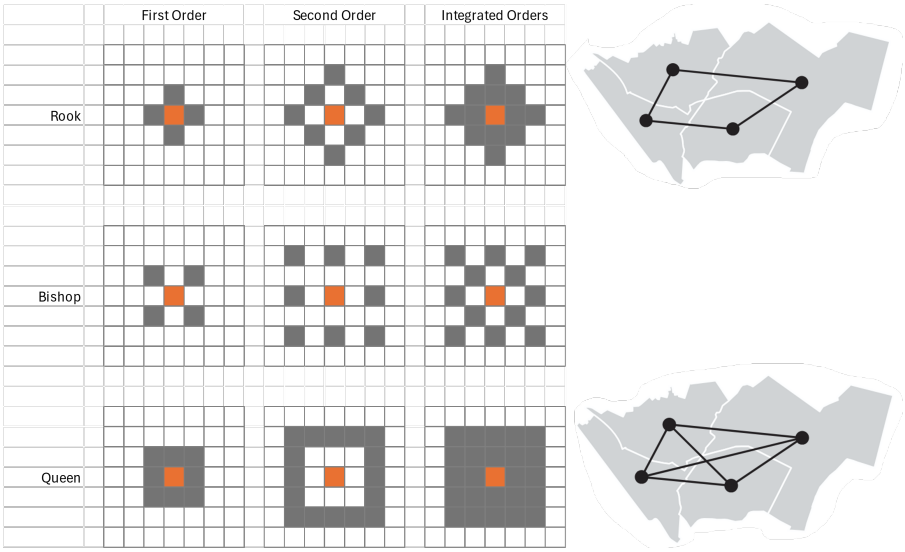


Figura 4.2.1: Contiguity criteria for auto-correlation taken from [100].

4.2.1.3. Spatio-Temporal Auto-correlation on Raster Time Series

Applying a spatial auto-correlation index to raster time series would imply obtaining as many auto-correlation values as there are time instants in the evaluated dataset. In this case, instead of having a single index as an evaluation criterion, one obtains a temporal evolution of spatial auto-correlation. There are indices that consider the temporal aspect based on the spatial auto-correlation formula [36, 77]. In particular, this work uses a spatio-temporal auto-correlation (STA) measure, also adapted from Moran's I index, which accounts for deviations in magnitude in the spatial dimension and the proximity of temporal trends in the time dimension [50]. The proposal was originally designed to be applied on human mobility time series based on taxi data from the city of Beijing, with the objective of discovering patterns in collective human mobility.

$$I^T = \frac{N}{\sum_i^N \sum_j^N W_{ij}} \frac{\sum_i^N \sum_j^N W_{ij} Z_i^T Z_j^T}{\sum_i^N (Z_i^T)^2} \tag{4.2.4}$$

This STA index is defined by Equation 4.2.4. The element Z_i^T corresponds to the deviation between the time series X_i^T, which contains T time instants, and the

average of the time series in the dataset. The Z_i^T is defined by the Equation 4.2.5:

$$Z_i^T = \phi(CORT(X_i^T, \bar{X}^T))(V_i - \bar{V}) \tag{4.2.5}$$

In this equation the ϕ function is an exponential adaptive tuning function, V_i is the accumulative volume of X_i^T, and $\bar{V}$ is the accumulative volume of $\bar{X}^T$. The $CORT$ function is the adaptive temporal dissimilarity measure [25], and is defined by the Equation 4.2.6:

$$CORT(X_T, Y_T) = \frac{\sum_t^{T-1}(X_{t+1} - X_t)(Y_{t+1} - Y)}{\sqrt{\sum_t^{T-1}(X_{t+1} - X_t)^2}\sqrt{\sum_t^{T-1}(Y_{t+1} - Y_t)^2}} \tag{4.2.6}$$

Finally, the STA index I^T gives a value in the range $[-1, 1]$, where a high value means that nearby time series (defined by W_{ij}) have similar quantitative deviation and similar temporal variance.

4.2.2. Time Series Representation

Efficient representation of time series has been widely addressed from different approaches, aiming to reduce space and enable competitive query response. For example, Time Lattice [87] leverages the implicit temporal hierarchy to execute multiresolution OLAP queries over large volumes of data, maintaining linear memory overhead and allowing constant-time updates, even in data streaming scenarios.

Another area of interest is symbolic representations, where methods such as Symbolic Aggregate approXimation (SAX) [81], Bag-of-Patterns (BOP) [82], or Multiresolution Vector Quantized (MVQ) [85] transform the series into discrete sequences or pattern histograms. This preserves the possibility of applying data mining techniques, improving efficiency in tasks such as similarity search, anomaly detection, and clustering, especially in long sequences or with structural patterns. Other proposals have explored the use of multiresolution approximations and vector transformations [113] to preserve both global and local information of the series, improving accuracy compared to classical distances such as Euclidean or Dynamic Time Warping.

In the spatial context, the raster model is widely used to represent attributes that vary continuously in space, such as temperature, precipitation, or elevation. Compact representations based on structures like k^2-trees and their variants [31] have allowed reducing the required space while supporting advanced queries over the stored values. Extensions to temporal raster data exploit spatial and temporal locality through strategies such as snapshots and logs [124], or the use of space-filling curves [106], enabling direct access to compressed data without the need for full decompression. Among these proposals, structures like *temporal k^2-raster* [21] and its variants with adaptive sampling [124] have shown high efficiency in datasets with reduced spatial changes between time instants, while approaches such as 2D1D-map and 3D2D-map [106] optimize access depending on the type and size of the queries.

These ideas are relevant to the present work, where the data structure is designed to exploit temporal and spatio-temporal autocorrelation of the data. The proposal of the *temporal k^2-raster* is described in more detail below, as it is used as a baseline for comparison in the experimentation (see Section 4.5).

4.2.2.1. Temporal k^2-raster

The *temporal k^2-raster* is a CDS for the efficient representation of raster time series. It employs a strategy based on snapshots and logs, capable of answering various queries over the data. The *temporal k^2-raster* is based on the k^2-raster [74, 75], a structure for compressing and indexing integer matrices. It supports queries for specific values, cells containing values within a range, and queries over ranges of rows and columns.

The *temporal k^2-raster* considers a grid of size $n \times n$, which is constructed by dividing this matrix into k rows and k columns, generating k^2 quadrants, dividing the space following the strategy used by the k^2-tree [18]. This division is performed iteratively, working with smaller sub-quadrants at each iteration. The sub-quadrants of each division can be represented as a tree where each node represents an area of the matrix, with the root node corresponding to the entire grid of integer values. Nodes of the tree store the minimum and maximum values representing the range of the cells within the area. The division of a sub-quadrant stops in two cases: (1) when the minimum and maximum values are equal, indicating that all cells contain the same value, and (2) when the sub-quadrant

corresponds to a single cell, meaning there are no further divisions of the space.

To store this structure, three bitmaps are used: one for the tree structure, one for the local maximum values of the nodes, and one for the minimum values. Figure 4.2.2 shows an example of the matrix and its iterative subdivision, followed by the conceptual representations, and finally the bitmaps used to represent the structure.

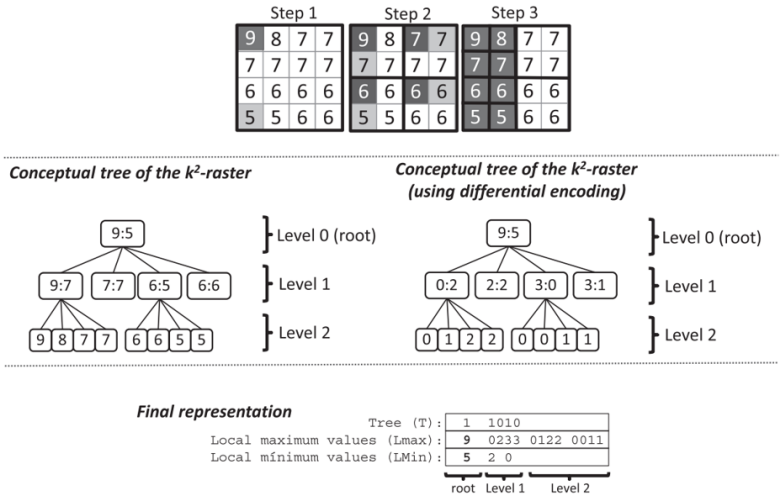


Fig. 3. Example (using $k = 2$) of integer raster matrix (top), conceptual tree of the *k²raster*, conceptual tree with differential encoding, and final representation of the raster matrix.

Figura 4.2.2: Example of k^2-raster [124].

The *temporal k^2-raster* uses the k^2-raster to represent the matrices corresponding to data at different time instants. Figure 4.2.3 shows an example of the building blocks that compose a *temporal k^2-raster*. In this snapshots-and-logs approach, the snapshots correspond to time instants that the structure uses as samples and are represented in full. The remaining time instants correspond to logs, which are represented using a snapshot as a reference. All value ranges stored in the nodes of the k^2-raster are represented as differences with respect to the snapshot node. Since these differences can result in negative values, zig-zag encoding [53] is applied.

Through snapshots and logs, temporal regularities are exploited by replacing repeated portions of the data with pointers to previous occurrences. The *temporal k^2-raster* proposes two methods for selecting snapshots and logs: one based on sampling, and the other based on heuristics.

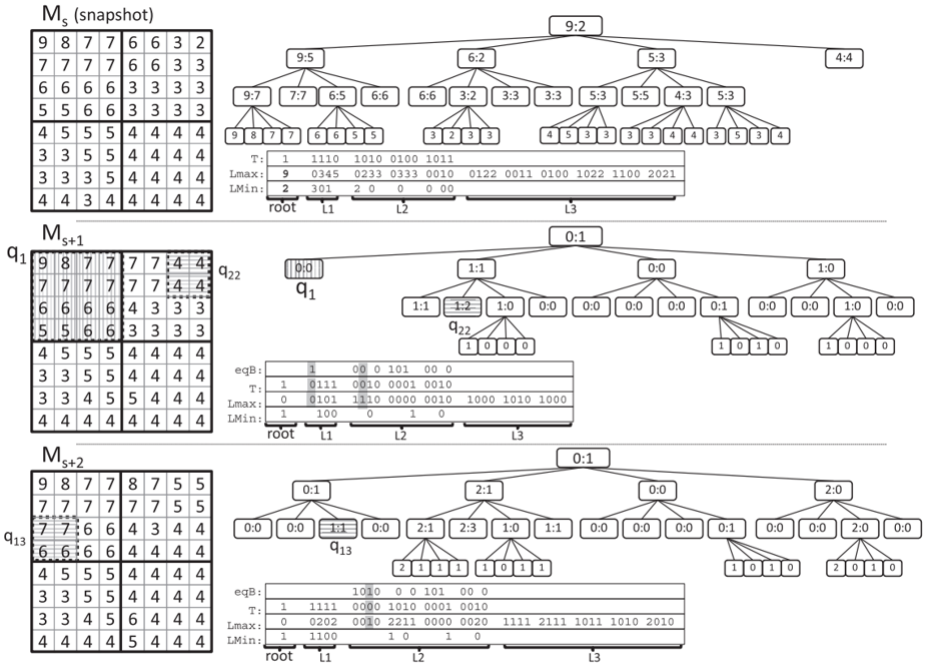


Fig. 8. Structures involved in the creation of a $T - k^2$*raster* considering $\tau = 3$.

Figura 4.2.3: Example of *temporal* k^2*-raster* [124]

4.3. Formalization and Analysis of the Datasets

Before describing and analyzing the datasets, we propose a formalization of the underlying characteristics of spatially-embedded time series (see Definition 9) that can be generalized to many other domains.

Definition 9 (Spatially-Embedded Time Series)**.** *A* spatially-embedded time series *is a collection of time series that share a common temporal resolution and length, such that each time series can be associated with a location in a spatial domain. This spatial assignment enables the definition of proximity relationships among the series according to a specified criterion.*

We characterize the Spatially-Embedded Time Series according to two main properties: spatial density and spatial neighborhood. Regarding density, we define a dataset as *dense* when, given a region of interest, the dataset contains defined values for most of the points comprising that space. Drawing an analogy with the GIS domain, this would correspond, at a logical level, to a raster model. Conversely, we define a dataset as *sparse* when, for a given region of interest, the dataset only provides data for certain specific locations (analogous to a vector

model in GIS terminology).

With respect to spatial neighborhood, we refer to *Euclidean* domains when proximity between time series is defined based on Euclidean distance; and to *semantic* domains when proximity depends on other characteristics inherent to the semantics of the domain, such as the type and direction of a road. It is important to note that, unlike density, neighborhood is not an exclusive characterization, and there may be domains in which both types of neighborhoods can coexist.

In the following, we describe the datasets employed in our study. For each dataset, we first describe the original data, followed by an explanation of the preprocessing applied to each dataset. These three datasets differ not only in domain but also in topology, spatial scale, temporal scale, and in how neighborhood relationships are defined among their elements.

4.3.1. Description of the Data Sources

This subsection provides a detailed description of the three datasets used in the experimental evaluation. Each dataset originates from a different domain and presents distinct characteristics in terms of spatial and temporal resolution, data density, and structure. The three datasets used are `nasa_data`, `eeg_data`, and `madrid_data`. Each of them is described below.

4.3.1.1. NASA Dataset (`nasa_data`)

This dataset corresponds to data obtained from the North American Land Data Assimilation System (NLDAS), created to provide reliable land surface statistics in order to improve weather forecasts [137]. The data includes records of temperature, pressure, humidity, among others (see the list of attributes in Table 4.3.1). The spatial resolution corresponds to a 1/8th degree regular latitude–longitude grid.

The data used in the experiments consists of hourly samples taken from January 1, 2018, to April 21, 2018 (2664 time instants). The spatial grid is 224×464 cells and represents the territory of the USA. Each attribute in the dataset is composed of a text file listing the corresponding binary files, where each binary file contains the information for a single time instant.

The range of values, mean, and standard deviation of the dataset are specified in Table 4.3.2. It can be observed that the attribute with the lowest variation in

Shorthand	Parameter	Level, Layer	Unit
APCP	Rainfall	Surface	[mm]
CAPE	Convective Available Potential Energy	180m above ground	[J kg-1]
DLWRF	Downward long-wave radiation flux	Surface	[W m-2]
DSWRF	Downward short-wave radiation flux	Surface	[W m-2]
FRAIN*	Rain fraction of total cloud water	Surface	proportion
PEVAP	Potential Evaporation	Surface	[kg m-2]
PRES	Atmospheric pressure	Surface	[Pa]
SPFH	Specific humidity	2m above ground	[kg kg-1]
TMP	Air temperature	2m above ground	[C]
UGRD	Zonal wind speed	10m above ground	[m s-1]
VGRD	Meridional wind speed	10m above ground	[m s-1]

Tabla 4.3.1: Description of the parameters comprising the `nasa_data` dataset.

its values is `SPFH`, with a range between -1 and 2. At the other end, the attribute `PRES` exhibits a much larger value scale compared to the others, ranging from millions to tens of millions.

	Min	Max	Mean	Median	Std. Dev.
APCP	-2004.0	2182.0	-431.77	-102.0	906.37
CAPE	-1.0	410867.0	1671.03	0.0	11708.37
CONV	-1.0	100.0	1.11	0.0	9.87
DLWRF	9919.0	47911.0	22147.16	23230.0	8254.09
DSWRF	-1.0	128095.0	11581.99	0.0	20572.08
PEVAP	-79.0	185.0	-10.05	1.0	40.21
PRES	6301609.0	10502674.0	8679280.44	9320085.0	1438057.70
SPFH	-1.0	2.0	-0.18	0.0	0.47
TMP	23421.0	31223.0	26531.28	27016.0	1965.16
UGRD	-2163.0	2147.0	-409.97	-25.0	992.03
VGRD	-2004.0	2182.0	-431.77	-102.0	906.37

Tabla 4.3.2: Description of values for the attributes of the `nasa_data` dataset.

Figure 4.3.1 shows a representation of each attribute in the dataset. The images in this figure correspond to heatmaps of the average value of each cell in the grid. The color blue represents the lowest value in the grid for each attribute, while the color red represents the highest value. For example, in the `PRES` attribute, several zones can be identified where the average value in multiple cells is very close to the maximum value. Similarly, there are other attributes where the opposite occurs, such as `CAPE`, where there are cells with fixed values in their time series. The marine surface is a zone without data for all attributes, which can be identified because in all cases those zones have values below the range of values of the territory with sensors. In some of the images, these zones are easily identifiable,

such as in `DSRF`, `SPFH`, and `VGRD`.

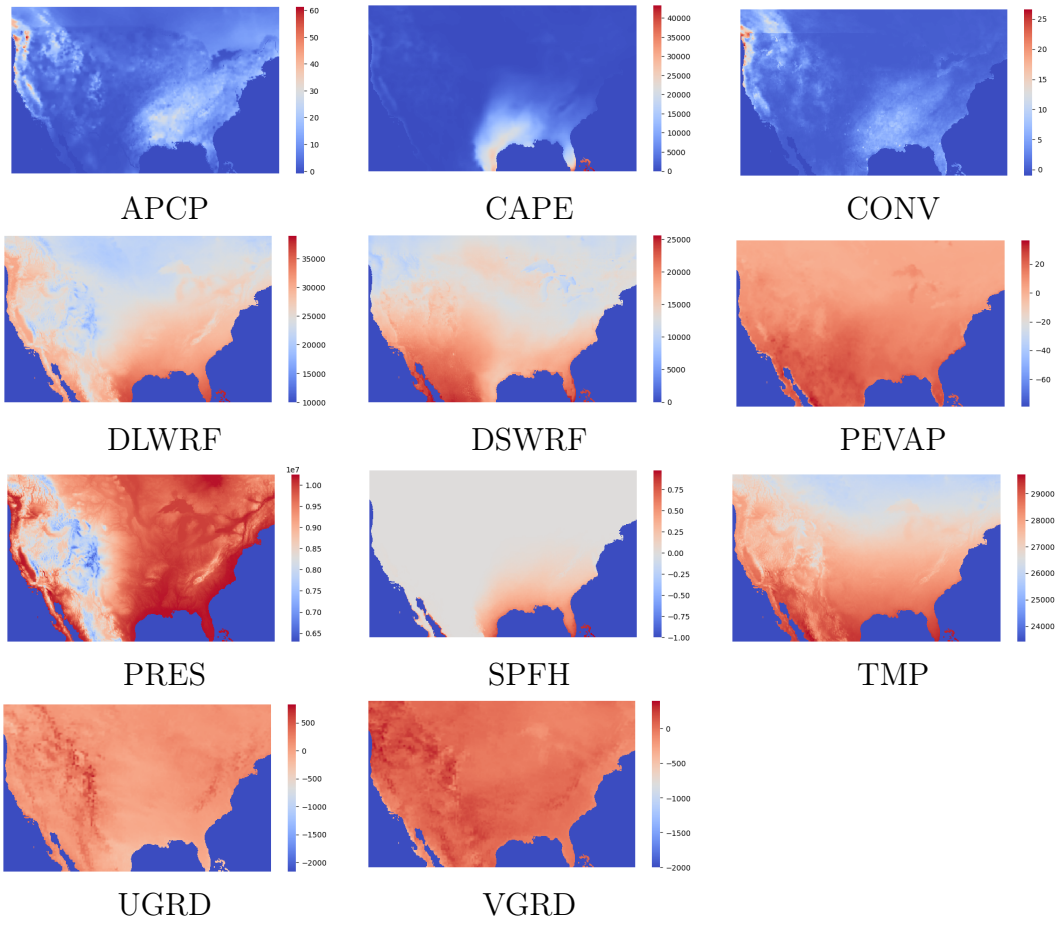


Figura 4.3.1: Heatmap based on the average of the time series of each cell in the NASA temporal raster.

According to the formalization presented in the previous section, this dataset is considered *dense*, since there is a time series associated with each cell of the grid, and *Euclidean*, as the neighborhood between time series is defined by their proximity within the grid, which corresponds to a Euclidean distance.

4.3.1.2. Electroencephalogram Dataset (`eeg_data`)

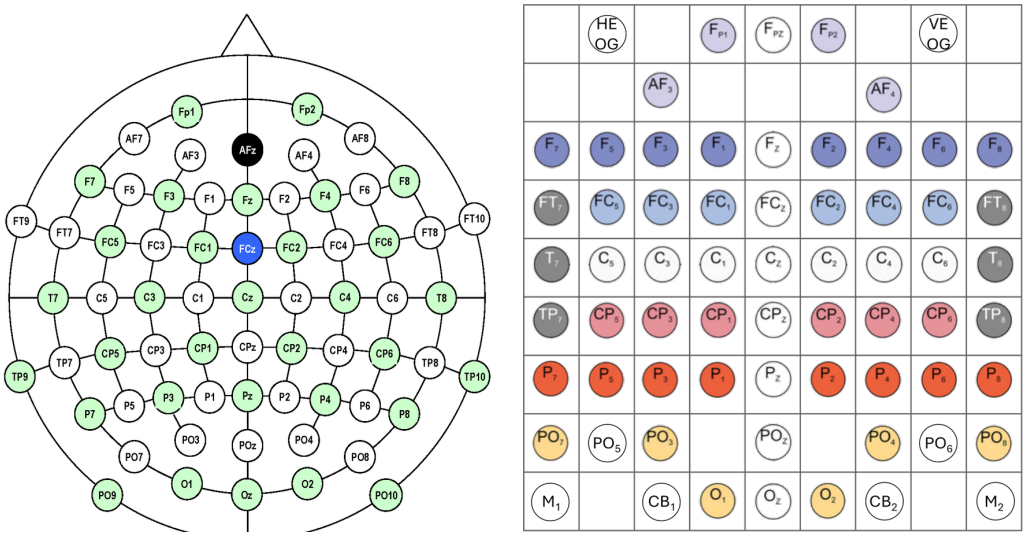
An EEG is a test that measures the electrical activity in a person's brain. Electrodes distributed over the subject's scalp are used for the measurement. The sensor placement follows a protocol, and each sensor is labeled with a code to identify which brain region it represents. The number of sensors may vary depending on the type of test performed.

The measurements from each of these electrodes correspond to time series, where

each sensor generates the same number of measurements per second. Thus, for a single EEG, the number of samples per time series is the same. However, between different EEG recordings, the length of the samples may vary.

The files in this dataset correspond to a database from the "Depression REST" project, downloaded from the "Patient Repository for EEG Data + Computational Tools" website[2]. This repository provides these files in MatLab format for post-processing and classification of EEG data. The dataset used in this work consists of 119 EEG files, with a total size of 6.68 [GB], where each file averages 57.5 [MB] in size.

There are different modalities of the EEG exam, differing in the number of electrodes used. Figure 4.3.2 shows on the left side (a) a diagram with the sensor names and their locations [115]. On the right side (b), the figure shows an approximation of the sensor locations available in the `eeg_data` dataset.



(a) Sensor names from [115] (b) Sensor electrode location

Figura 4.3.2: EEG sensor distribution diagram.

Figure 4.3.3 shows a preview of the data from one of the files in the dataset. The sensor plots follow the same pattern as the diagram in Figure 4.3.2(b). The first rows of the matrix (top) correspond to sensors located at the front, and the last rows (bottom) represent data from sensors located at the back of the subject. In this example, it is possible to see that the center of the matrix shows plots without significant visible alterations, while the sensors in the first rows exhibit

[2] http://predict.cs.unm.edu/

more variation in their behavior.

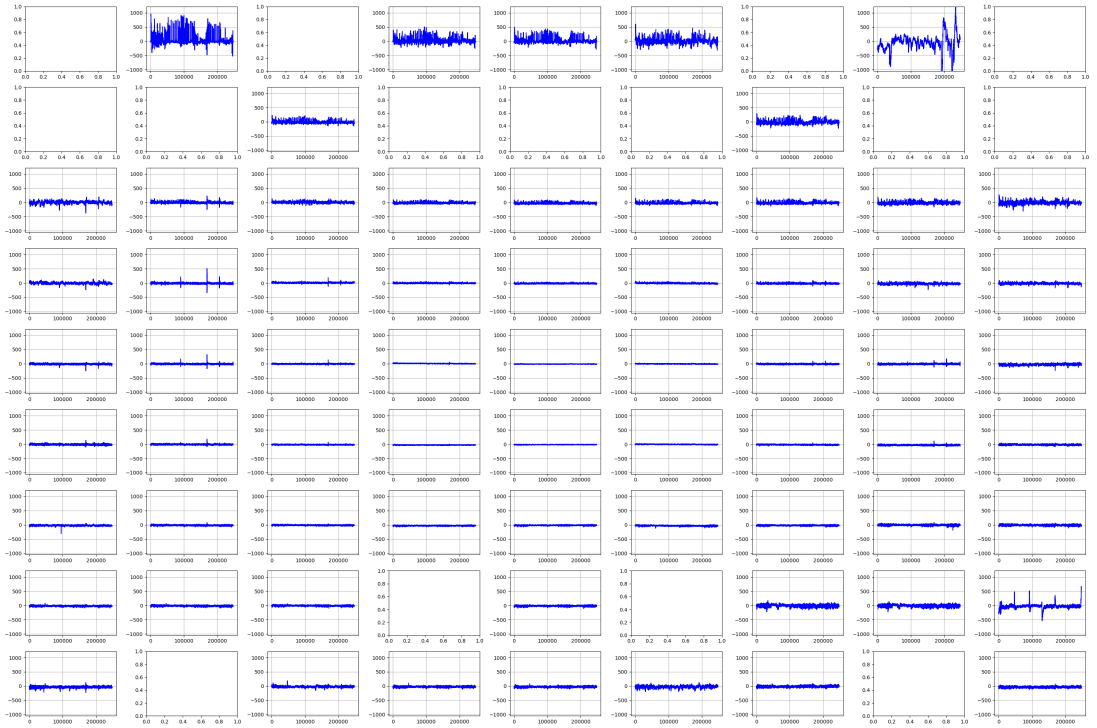
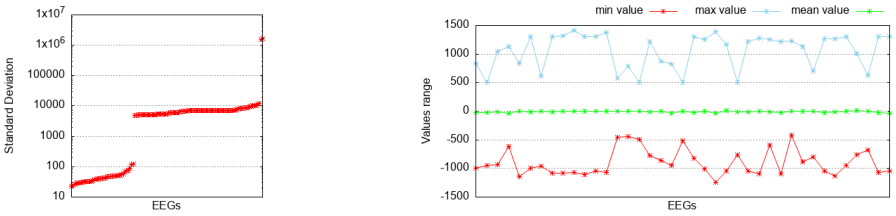


Figura 4.3.3: EEG data preview.

The EEGs in the dataset have an average of 248, 474,1 samples per sensor, with the longest recording containing 290, 020 samples and the shortest 116, 692. The average value range of the `eeg_data` dataset lies between $-1,284,629{,}2$ and $1,283,659{,}1$. The minimum recorded value in the entire dataset is $-64,000,000$, while the maximum is $63,955,080$. The average mean value across the dataset is 249,4, with the smallest average being $-68{,}5$ and the largest average $15,123{,}5$. The average standard deviation of the dataset is $31,067$, with the lowest deviation at 22,5 and the highest at $1,605,167{,}3$.

Figure 4.3.4 presents statistics on the values of the EEGs in `eeg_data`. Figure 4.3.4(a) details the standard deviation of all EEGs in the dataset. The standard deviation values are sorted in ascending order to facilitate the identification of three distinct behaviors. The first group has deviations below 150, while a large number of EEGs have deviations close to 10,000. Two EEGs exhibit deviations around 1.6 million. Figure 4.3.4(b) shows the value range of the 39 EEGs with the lowest deviations. This group is depicted in the descriptive figure mainly because the magnitudes of the other EEGs cause these data to lose visibility when all are plotted together. Additionally, the other ranges do not show as much variation

relative to their magnitude. The range of values for the first group is between -1500 and 1500, with an average close to 0 in all cases. Among the remaining elements, two EEGs provide the extreme values of the dataset, with an average of 14,383,75. The remaining 78 EEGs have an average close to 16,67, and their overall behavior is regular, with all having a minimum value of $-318,407$ and maximum values between $317,500$ and $318,500$.



(a) Standard deviation of the EEGs (b) Value range of an EEG subset

Figura 4.3.4: Statistics on `eeg_data`.

The characterization of this dataset within our formalization is less straightforward; however, based on the results presented in the following section, we consider it to be *sparse* and *semantic*.

4.3.1.3. Madrid Sensors Dataset (`madrid_data`)

This dataset corresponds to traffic measurement values provided by sensors deployed in the city of Madrid, Spain, published on the city council's open data portal[3]. The data is organized in two files: one with the recordings from each sensor and another describing the location of those sensors. In this work, information from the entire year 2024 is used, restricted to the sensors on the `M30` road, which are shown in Figure 4.3.5. This road corresponds to a ring road with highway characteristics in the city of Madrid, Spain. The road is designed for a high flow of vehicles, allowing long stretches of the city to be traversed in short periods of time.

The first file is titled "Traffic. Historical traffic data since 2013" and contains data that is automatically collected and integrated in 15-minute intervals. Some of the fields in the CSV file are: `id, fecha, tipo_elem, intensidad, ocupacion, carga,` and `velocidad`. The `id` corresponds to a unique identifier for the measurement point (sensor), which is a sequential numeric value that does not

[3] https://datos.madrid.es/portal/site/egob

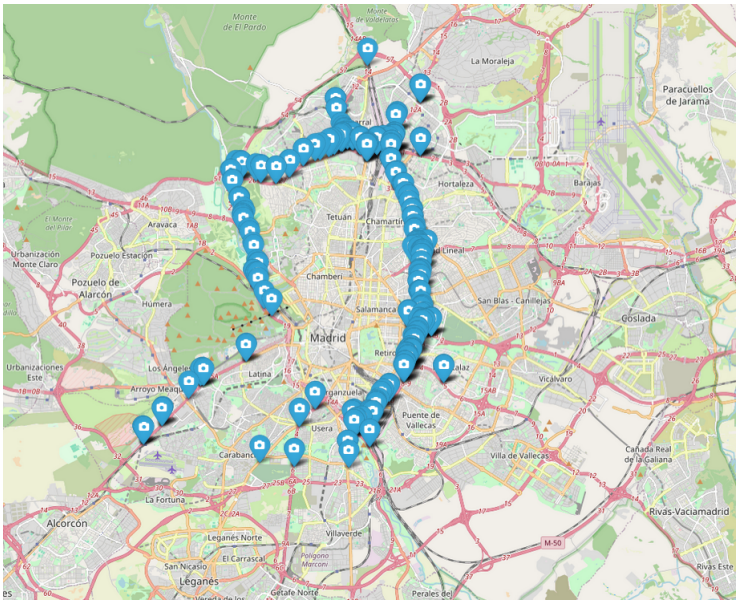


Figura 4.3.5: Location of traffic sensors on the M30 road in Madrid.

change over time. The `fecha` field corresponds to the official date and time of the sample. The `tipo_elem` label identifies two types of sensor locations: some sensors are located on the ring highway known as `M30` in the city of Madrid, while the rest are located elsewhere in the urban area. The fields `intensidad, carga, ocupacion`, and `velocidad` correspond to the average of traffic sensor measurements over a 15-minute period.

The value of `carga` is a synthetic indicator that attempts to measure the degree of saturation of a road. Its formula considers intensity, saturation, occupancy time, and the load coefficient of the road. The attribute `ocupacion` is the percentage of time the sensor detects a vehicle, `intensidad` indicates the number of vehicles per hour at the measurement point, and `velocidad` corresponds to the average speed recorded by vehicles passing through the measurement point during the measurement interval.

Figure 4.3.6 shows a histogram with the number of samples for the 4 attributes available in the dataset within the `M30` road. The first histogram, which displays `ocupacion, carga`, and `velocidad`, represents $9,579,712$ samples of the attributes, while `intensidad` has $9,468,920$ samples. This difference in the number of samples occurs because, in some cases, sensors do not have values for all the attributes. Table 4.3.3 shows information about the values of these attributes. It can be seen that the value range for all attributes starts at 0, reaching up

to 100 for `carga` and `ocupacion`, up to 235 for `velocidad`, and up to 8452 for `intensidad`.

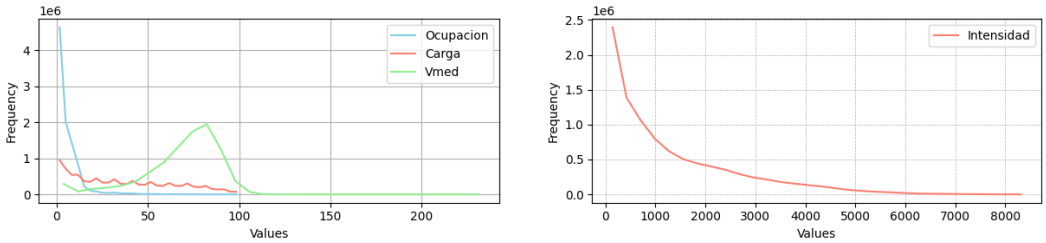


Figura 4.3.6: Histogram of values for `madrid_data`.

	min	max	mean	std
`intensidad`	0.00	8452.00	1294.70	1313.85
`ocupacion`	0.00	100.00	5.49	7.09
`carga`	0.00	100.00	36.28	27.52
`velocidad`	0.00	235.00	68.44	21.62

Tabla 4.3.3: Description of values for the attributes of the `madrid_data` dataset.

The second file is titled "Traffic. Location of traffic measurement points" and contains the description of the points where the sensors are located. The data includes the measurement point code, its name, district, element type, location in geographic coordinates (latitude and longitude), as well as in Universal Transverse Mercator (UTM), along with a unique numeric identifier that allows relating both data files. In this work, only the file describing the sensors from January 2024 is used. This is because in January there are 296 sensors registered on the `M30`, and no new devices are registered until November and December, which add 15 and 10 sensors, respectively. These latter devices have less than 17 % of the number of samples possessed by the January sensors, so their use is discarded in this work.

The sensors on the `M30` road can be differentiated by the code that describes them. This code consists of 2 letters, "PM", followed by 5 digits that describe the following three aspects: the lane (1 digit), the kilometer point (3 digits), and the lane situation (1 digit). The first criterion indicates whether it is the inner lane, outer lane, exit from Madrid, or entrance to Madrid. The second gives the location measured in hectometers. The third indicates whether the lane is main, lateral, access, exit, or one of four other types. In this way, the code allows identification of each sensor existing on the `M30` road.

Finally, with respect to the characterization of this dataset, in terms of density it is clearly *sparse*. Regarding neighborhood, although it could initially be considered *Euclidean*, it is important to note that the closest sensors in terms of distance may be located on opposite sides of the roadway (or in other configurations, depending on the types of lanes described above). Therefore, as will be shown by the results presented in the next section, a *semantic* characterization is more appropriate.

4.3.2. Data Preparation and Auto-correlation Measures

This section describes the steps carried out to evaluate autocorrelation in the datasets used. Each dataset undergoes a pre-processing stage, which prepares and cleans the data before applying any procedure, either to determine its autocorrelation or to store it using the proposed data structures.

In a first phase, the datasets are characterized with respect to their auto-correlation using two measures: one for temporal auto-correlation and the other for STA. The first auto-correlation index corresponds to the Pearson product-moment correlation coefficient, which is applied over the temporal dimension of the data using different intervals[4]. The second index corresponds to the STA index described in Subsection 4.2.1.3, using Equation 4.2.4.

It is important to note that auto-correlation measures indicate the similarity among the data. For example, a high temporal auto-correlation index indicates that values at consecutive time instants are similar, and therefore temporal similarity can be exploited. On the other hand, a high STA indicates that two time series close in space exhibit similar behavior.

4.3.2.1. NASA Dataset (`nasa_data`)

No pre-processing is necessary for this dataset. This is because there are no missing values, and its format corresponds to a grid representation where each cell stores a time series, all with the same number of samples.

In order to compute the STA, the neighborhood of the cells is established by contiguity, using the "queen" criterion; that is, each element is considered a

[4]To determine the temporal auto-correlation using the Pearson product-moment correlation coefficient, the *corrcoef* function from the *NumPy* library (*Python*) is used. Lags of 1, 2, 5, and 10 time steps are applied.

neighbor to the 8 elements surrounding it, as one of the examples shown in Figure 4.2.1.

It is relevant to note that some grid cells are located over areas representing the ocean. These cells have the lowest value in the dataset with no variation over time and do not represent useful information. To discard these cells, the following restriction is considered: cells without valid information are not considered neighbors of others.

Figure 4.3.7 shows a graph of the temporal evolution of Moran's I index for all attributes in `nasa_data`. Most graphs show a line close to the upper limit (value 1), indicating a high spatial auto-correlation in almost all time instants. This means that cells with high values are surrounded by cells that also have high values, and vice versa. There are some others whose index oscillates between 0.5 and 1, such as `APCP`, `CAPE`, and `CONV`. Finally, among all attributes, `DSWRF` shows the largest range in the auto-correlation index, containing some time instants close to 0. This indicates that some time instants do not exhibit spatial auto-correlation at all.

Figure 4.3.8 presents other auto-correlation values for the dataset, where some instances along the temporal dimension are considered, as well as the STA index (explained in Section 4.2.4).

Regarding temporal auto-correlation, there is an expected decrease as the lag value increases. That is, when the compared time instants are farther apart, the data exhibits less auto-correlation. The temporal indices lag_5 and lag_10 show auto-correlation values between -0.5 and 0.5 for the attributes `APCP, CAPE, CONV, DSWRF, PEVAP`, and `SPFH`, whereas for lag_1 and lag_2, the dataset exhibits high positive auto-correlation.

Finally, regarding the STA, a positive auto-correlation is observed, with high values always above 0.9. Only the attributes `CAPE` and `TMP` present STA values that are exceeded by temporal auto-correlation values. This generally indicates that even though consecutive values in time have high similarity, the behavior of time series close in space presents a higher degree of similarity among themselves.

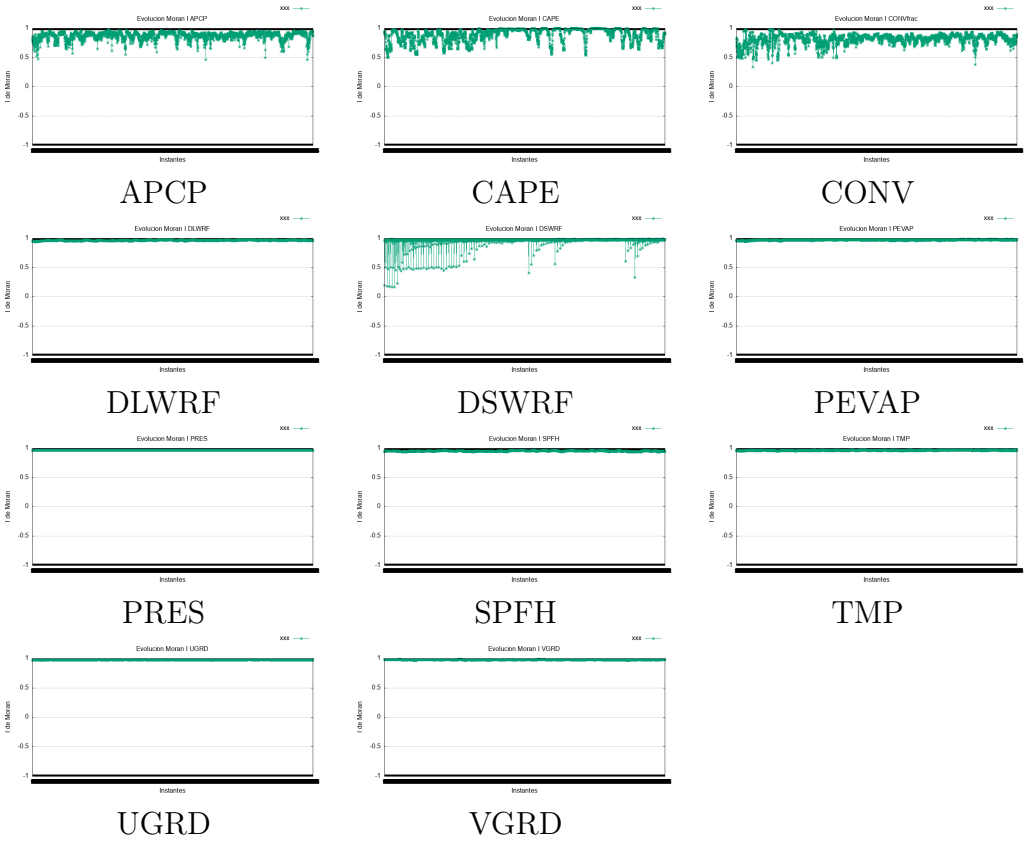


Figura 4.3.7: Evolution of Moran's index for each attribute of the NASA dataset across all time instants.

4.3.2.2. Electroencephalogram Dataset (`eeg_data`)

The pre-processing of `eeg_data` focuses on two aspects of the dataset: the type of data it contains and the number of sensors in each EEG. Then, it is possible to define how the elements relate to each other in order to obtain the auto-correlation indices of the EEGs.

The data of this dataset consists of decimal numbers, then, a modification is made to work only with integers. This is because the interpretation of EEGs does not require high precision, as the behavior of the signals is evaluated beyond the specific values. Still, two decimals are retained for all recorded numbers by multiplying the values by 100 and then truncating the decimal part.

All EEGs in `eeg_data` record at least 66 sensors labeled by their identification code. Some files include an additional sensor coded as "EKG", corresponding to recordings of cardiac activity during the EEG acquisition. Therefore, this time series is omitted in those files that contain it to ensure the same number of sensors

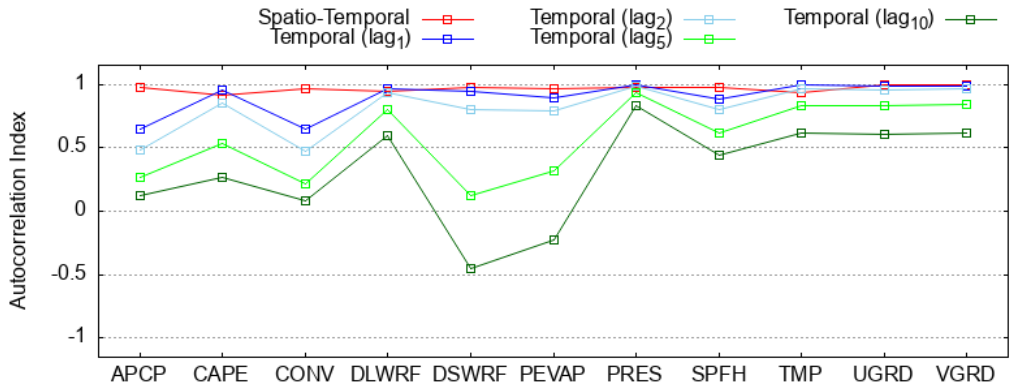


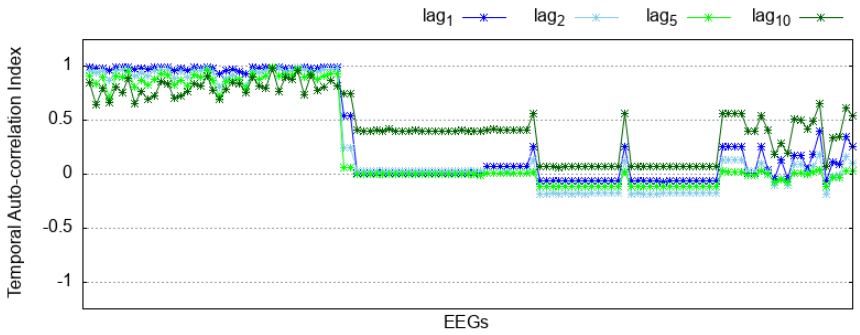
Figura 4.3.8: Temporal auto-correlation and STA indices of `nasa_data`

in all EEGs.

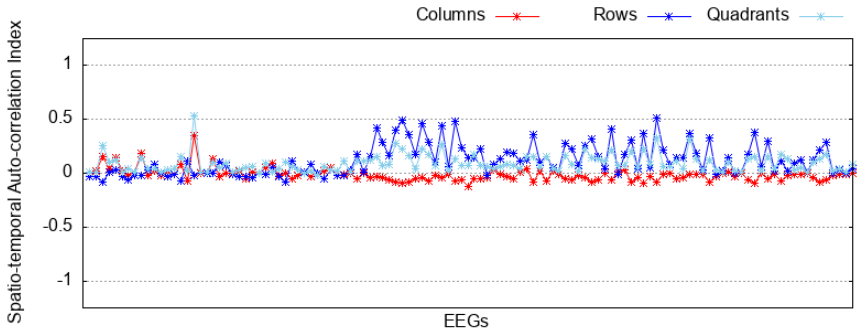
In this dataset, neighborhood is explored using three different configurations of relationships between sensors. The configurations are based on the matrix shown in Figure 4.3.2(b):

- By rows, where the criterion corresponds to the first letter of the sensor name, except for the first and last rows with sensors $HEOG, VEOG, M_1, M_2, CB_1$, and CB_2. These sensors are considered but do not follow the described rule.
- By columns, where grouping is done by the numeric component of the sensor name (except for the sensors $HEOG, VEOG, M_1, M_2, CB_1$, and CB_2, which do not follow the rule but are considered according to the scheme).
- By proximity, where each sensor is related to those surrounding it, using the "queen" contiguity criterion.

The autocorrelation values of `eeg_data` with different clustering criteria can be seen in the graph in Figure 4.3.9. These results show that the temporal dimension is the only one presenting some cases with auto-correlation values above 0.5 and very close to 1. Interestingly, this group of high temporal auto-correlation corresponds to those EEGs with low standard deviation in the dataset (see Figure 4.3.4). For the other EEGs, the temporal auto-correlation is close to 0, except for some cases of temporal auto-correlation lag_{10} showing values close to 0.5. These cases are higher than other variants of the index and correspond to a subset of EEGs with standard deviation close to 10,000. Figure 4.3.4(b) shows the spatio-temporal autocorrelation for the three different neighborhood criteria described above, where the *Quadrants* correspond to a "queençontiguity criterion. The STA values reaches a maximum of some cases close to 0.5, but in general the trend



(a) Temporal Auto-correlation



(b) Spatio-temporal auto-correlation

Figura 4.3.9: Autocorrelation indices in `eeg_data`. For temporal evaluation, lags of 1, 2, 5, and 10 are used. For spatio-temporal auto-correlation, the three neighborhood criteria are presented: by columns, by rows, and by quadrants (queen).

approaches 0. The column-based neighborhood variant shows the lowest trend among the three cases, remaining close to 0, except for some points in the graph. Conversely, the row-based neighborhood has the highest number of values close to 0.5, where the EEGs with this behavior belong to the subset with standard deviations near 10,000.

Figure 4.3.10 complements the analysis of the possible relationship between auto-correlation values and standard deviations in `eeg_data` , using the proximity-based neighborhood definition. A trend can be observed where STA values range between 0 and 0.5 regardless of the standard deviation value. In contrast, temporal auto-correlation presents values close to 1 for standard deviations below 150, and temporal auto-correlation values below 0.5 and close to 0 when the standard deviation approaches 10,000.

The `eeg_data` dataset exhibits STA values below 0.5, indicating that the time

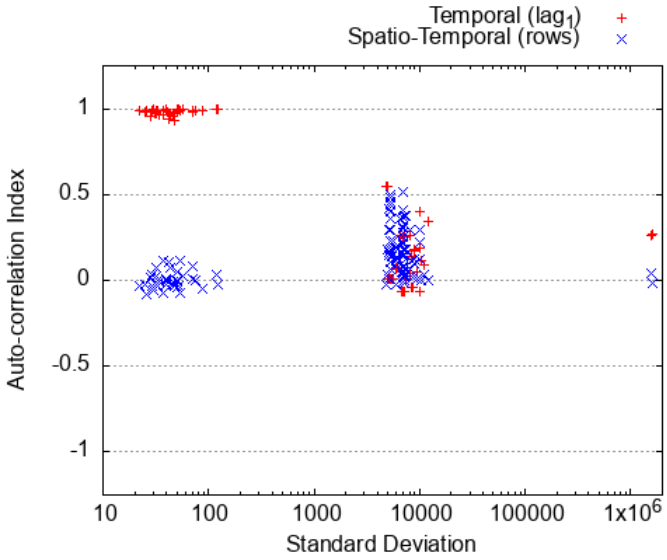


Figura 4.3.10: Scatter plot of auto-correlation vs. standard deviation in `eeg_data` dataset.

series do not show a marked pattern of similarity based on proximity. On the other hand, temporal auto-correlation shows values very close to one for a subset of EEGs, which share the characteristic of having a standard deviation near or below 100. That is, in this EEG subset, consecutive values of a time series exhibit similar behavior.

4.3.2.3. Madrid Sensors Dataset (`madrid_data`)

For this dataset, sensors to be included in the analysis and experimentation are selected based on their location. Then those lacking sufficient data are filtered out either for discarding or for applying imputation of missing data. As mentioned in Section 4.3.1, this work only considers data from the `M30` highway. It is expected that sensors located along this highway exhibit readings with a high degree of similarity, which translates into high auto-correlation. The initial dataset consists of 296 sensors according to the measurement points description.

The cleaning process aims to obtain time series that are formed with at least 95 % original samples [118]. For the remaining data, linear imputation is applied to have complete time series based on the possible samples for the year 2024. Cross-referencing the data with the samples, there are 291 sensors with registered samples for the four attributes in the dataset. Then, sensors with less than 5 % missing samples for each attribute are identified, obtaining 240, 241, 241, and

232 sensors for the attributes `intensidad`, `ocupacion`, `carga`, and `velocidad`, respectively. Time series with fewer than 35, 135 samples (based on the days of the year and samples every 15 minutes) are subjected to linear imputation.

For the auto-correlation of `madrid_data`, neighborhood is defined based on a grouping of sensors according to their codes. This grouping aims to consider neighbors those sensors that have continuity along the highway such that a vehicle passing through a measurement point (sensor) on a lane will most likely pass through the next measurement point on the same lane. Four groups are created based on lane and direction, plus a fifth group for sensors not belonging to any of the others. The four groups are: Inner-Main (IM), Outer-Main (OM), Inner-Side (IS), and Outer-Side (OS).

Besides this neighborhood criterion, others based on Euclidean distance with incremental ranges (50, 100, 150, and 250 meters) and one with the two closest neighbors are tested. Note that the first neighborhood criterion also uses two neighbors per sensor, but those neighbors are not necessarily the two closest sensors, since they are the previous and next sensors on the same lane.

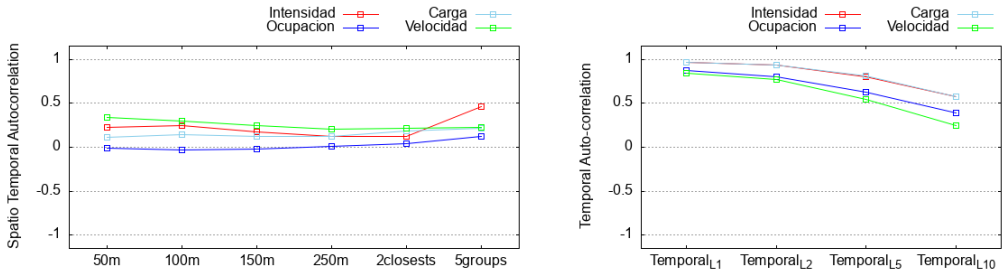


Figura 4.3.11: Auto-correlation in `madrid_data`.

Figure 4.3.11 shows the behavior of the temporal and STA indices for the attributes of the `madrid_data` dataset. In the case of STA, it can be seen that the six defined contiguity types have values below 0.5, with the highest being the attribute `intensidad` in the `5groups` configuration explained above. This is followed by the attribute `velocidad` in the neighborhood defined by a *50m* distance. In the same Figure 4.3.11, the temporal auto-correlation is shown, which presents higher values for all attributes except for $Temporal_{L10}$. In contrast, the case of $Temporal_{L1}$ shows a positive auto-correlation with values close to 1 for all attributes of `madrid_data`.

Thus, it can be established that the `madrid_data` dataset has low spatial auto-

correlation for all attributes regardless of the neighborhood criterion, indicating that neighboring time series are not necessarily similar to each other. On the other hand, the same attributes present very high temporal auto-correlation at lags 1 and 2. Therefore, consecutive time instants are very similar for all attributes in this dataset.

4.4. Proposed Data Structures

The analysis presented in previous section provides insight into the internal structure of the data and help in the selection of appropriate representation strategies. Given the differences among the datasets, and their respective levels of temporal auto-correlation and STA, the proposed data structures are applied to each case. The obtained results are then compared against the baselines defined for each dataset. Ultimately, the goal is to identify whether a relationship exists between the auto-correlation values and the performance of different space-efficient representation strategies.

In this section we first present a proposal designed for *dense* dataset. These are time series data with spatial locations where their elements are arranged in a grid of contiguous cells. Then, an adaptation for *sparse* dataset is described. Both proposed data structures share great similarity, so the first structure is described in detail regarding how the data are created, and the second structure highlights the differences with the first for sparse datasets.

4.4.1. QuadComp: Quadrant Compress

A *dense* collection of spatially-embedded time series can be regarded as a temporal raster, for which the *temporal k^2-raster* described in Section 4.2.2.1 represents the most efficient state-of-the-art solution. In this section, we propose the *QuadComp* strategy, which compresses data using a different approach: instead of considering temporal rasters as a sequence of matrices at different time instants, it considers each cell as storing a time series. The hypothesis of this work is that a time series can be efficiently represented using a reference series that is relatively similar to it.

This approach also takes advantage of the similarity that may exist in some datasets regarding their locality or spatial proximity, which can be computed

using some of the metrics explained in Section 4.2.1. Based on the fact that high STA indicates that spatially close time series are similar, it is possible to divide the grid of time series into blocks of a certain size, allowing the representation of all series within the same block with respect to the same reference time series.

4.4.1.1. Structure Description

Given a temporal raster M with r rows and c columns, M can be divided into quadrants of size $d \times d$, where the grid of quadrants is composed of $f_q = \lceil r/d \rceil$ rows and $c_q = \lceil c/d \rceil$ columns. Thus, the set of quadrants Q contains $|Q| = f_q \times c_q$ quadrants, and in this matrix, the quadrant identified as $q_{i,j}$ (with $0 \leq i < f_q$ and $0 \leq j < c_q$) is located in the i-th row and j-th column of quadrants. Therefore, a cell in M, identified as $m_{x,y}$ (with $0 \leq x < r$ and $0 \leq y < c$), located in row x and column y of the grid M, belongs to the quadrant $q_{i,j}$, with $i = \lfloor x/d \rfloor$ and $j = \lfloor y/d \rfloor$.

The number of reference time series would be equivalent to the number of quadrants in Q, as all cells within a quadrant could be represented by the same reference series in datasets with high correlation among their data. However, there are cases where the time series show no variation in their values, either due to consistently constant readings, technical failures, or absence of data during the time span of the time series. As such, quadrants may exist where all series exhibit this invariance over time. These series are referred to as *fixed time series*. Our structure takes these particular cases into account and it consists of the following components:

- RS: the set of reference time series, with a maximum size equal to the number of quadrants in the grid ($|Q|$).
- CS: the set of time series encoded with respect to the reference time series of their corresponding quadrant. In this case, the maximum number of encoded series is approximately $|M|$, but it depends on how the reference time series are defined. In this initial proposal, the reference time series corresponds to a time series from the dataset itself, so the maximum number of time series to be encoded with respect to Q is $|M| - |Q|$. The assumption here is that using a small quadrant in a dataset with high STA allows selecting one of the time series within the quadrant directly as reference. In other contexts, it might be possible to use a reference time series generated from those within the quadrant, potentially improving compressibility.

- FB: a set of binary values (bitmap) used to identify the *fixed time series* in the dataset.
- FS: a set of values, each representing one of the *fixed time series* in the dataset M. This assumes that the *fixed time series* have distinct values. In cases where all *fixed time series* in the dataset share the same value, this set may contain only one element.
- QB: a bitmap used to identify those quadrants that have a reference time series. There may be quadrants that contain only *fixed time series*, and thus would not have an associated reference time series.
- RB: a bitmap used to identify the time series that are used as reference for a quadrant.

Figure 4.4.1 shows, in its upper part, a grid of time series, and in the lower part, a diagram of the *QuadComp* structure representing it. This grid contains two types of cells: those containing time series (in cyan), and others containing *fixed time series* (in salmon). Above the grid, four distinct quadrants can be identified, each corresponding to a group of 3×3 cells. The first quadrant Q_1 contains all cells A_i, the second quadrant Q_2 contains the B_i cells, the third Q_3 the C_i cells, and the last quadrant Q_4 the D_i cells. Among these quadrants, three types can be distinguished: quadrants in which all cells contain complete time series (e.g. Q_1), quadrants containing only *fixed time series* (e.g. Q_4), and quadrants containing both time series and *fixed time series* (e.g. Q_2 and Q_3).

The *QuadComp* structure is shown in the bottom part of Figure 4.4.1, illustrating the six components previously described for representing a grid of time series. The RS set holds the encoded reference series for each quadrant. Since quadrants that contain only *fixed time series* do not require a reference series, only 3 reference series are stored instead of 4. In this proposal, the reference series used corresponds to the first valid series found in each quadrant. Tests were conducted using other series as reference, including a generated average series from all others in the quadrant, but these alternatives did not achieve significant improvements. Reference series are encoded by computing the difference between consecutive time instants, followed by a zig-zag encoding to deal with negative differences. In Figure 4.4.1, time series B_1 is highlighted as it passes through an encoder before being stored in RS. For example, if $B_1 = \{4, 2, 6, 8, ...\}$, then its delta encoding would be $B'_1 = \{4, -2, 4, 2, ...\}$, and finally, $zigzagencoding(B'_1) = \{8, 3, 8, 4, ...\}$. This

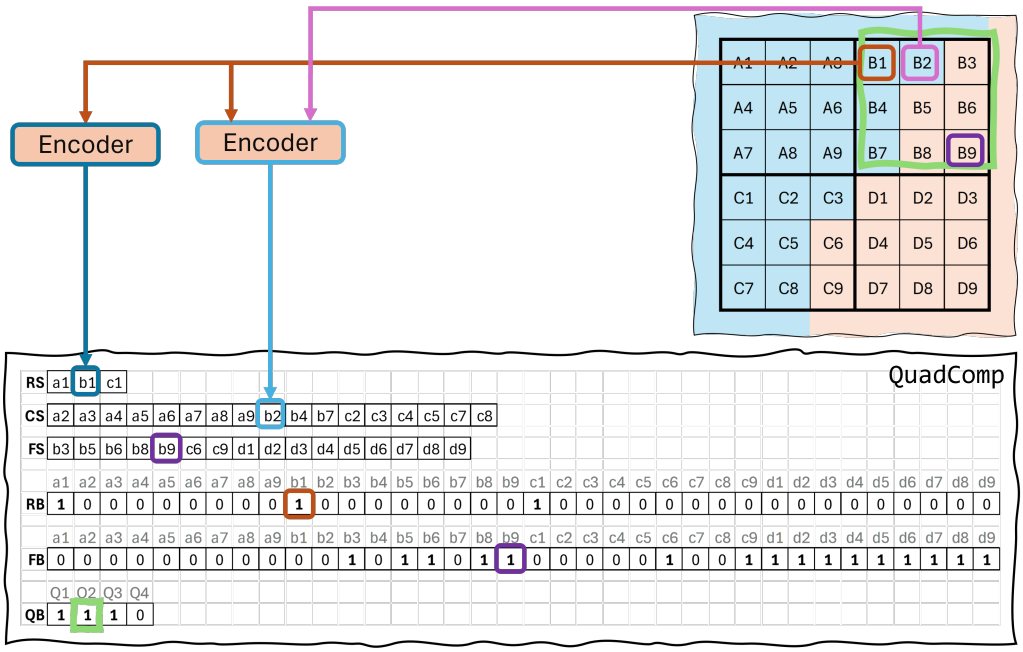


Figura 4.4.1: Diagram of the *QuadComp* structure.

encoding method leverages the temporal locality of the data – i.e. the similarity between consecutive time instants within a time series – while the use of references simultaneously exploits the spatial locality among the time series as we describe below.

The encoded series set CS stores all remaining series that are not *fixed time series*. These series are encoded using the reference series of their quadrant. As shown in Figure 4.4.1, time series B_2 from quadrant Q_2 is encoded using B_1 as reference, and the resulting series b_2 is stored in CS. The encoding is defined as $b_2 = zigzagencoding(B_2 - B_1)$. The difference between two time series J and K results in a new time series L, computed as $L_i = J_i - K_i$. For example, if $B_2 = \{5, 4, 5, 5, ...\}$ and $B_1 = \{4, 2, 6, 8, ...\}$, then $B_2 - B_1 = \{1, 2, -1, -3, ...\}$, and $b_2 = zigzagencoding(B_2 - B_1) = \{2, 4, 1, 5, ...\}$. This reference-based difference encoding exploits spatio-temporal locality, i.e. the similarity over time between nearby time series. Note that the structure stores encoded series in row-major order by quadrant: first those from Q_1, then Q_2, and so on.

The final set FS is used to store the values of the *fixed time series*. Since these series show no variation, only the constant value is stored and later repeated as many times as the series length. In the example, the *fixed time series* B_9 is highlighted in purple in both FS and the bitmap FB, which is explained below.

The remaining elements in Figure 4.4.1 are the bitmaps that support navigation through the structure. The first bitmap, RB, marks all series selected as references for their quadrant. As shown in brown, the position corresponding to series b_1 is marked, indicating that it is the reference for quadrant Q_2. Similarly, the positions for series a_1 and c_1, which are references for Q_1 and Q_3, respectively, are also marked. The second bitmap, FB, identifies those cells containing *fixed time series*, such as the highlighted b_9 cell, showing that series B_9 from quadrant Q_2 is a *fixed time series*. Finally, the QB bitmap indicates whether a quadrant contains time series other than *fixed time series*. For example, the position corresponding to Q_2 is highlighted in green. Note that a quadrant with only *fixed time series* will have a value of 0 in this bitmap, as is the case for quadrant Q_4 in the illustrated example. In general, these bitmaps may contain very few bits set to one, compared to the number of bits set to zero, so the structure uses an efficient implementation for sparse binary sets[5].

4.4.1.2. Queries

Three queries have been implemented on the *QuadComp* structure:

- `getSerie`: described in Algorithm 4, this query retrieves a time series from the set M given a row and column.
- `accesQuery`: retrieves the value of a time series (located at a specific row and column) at a given time instant.
- `windowQuery`: retrieves a subset of time series over a range of rows, a range of columns, and a range of time instants (see Algorithm 5).

The procedure for the `getSerie` (Algorithm 4) and `accesQuery` queries is similar for both. The strategy to answer the query is as follows: First, it is evaluated whether the time series belongs to the set of *fixed time series*. In that case, the value that is repeated for the entire series is retrieved and the corresponding result is returned (a one-dimensional array with the value repeated as many times as the number of time instants in the dataset, or just the integer value in the case of `accesQuery`). Then, if it is not a *fixed time series*, the series may either be the reference series of its quadrant or a series represented based on that reference series. Therefore, it is mandatory to retrieve the reference series of the quadrant. If the series turns out to be the reference, it is returned directly – either the value

[5] https://github.com/simongog/sdsl-lite/blob/master/include/sdsl/sd_vector.hpp

at the queried time instant for `accesQuery`, or the entire series for `getSerie`. Finally, if the time series is not a reference one, the stored series is decoded based on the quadrant's reference series (previously retrieved), and the result is returned as appropriate.

Algorithm 4 Algorithm getSerie(QC, i, j)

```
result ← ∅
posQLP ← QC.getLinealPositionOfQuadrant(i,j)
if QC.isFixedTimeSeries(posQLP) then
  result ← QC.decodeFixedSeries(posQLP)
  return result
end if
reference ← QC.decodeReferenceSeriesOfQuadrant(QC.getQuadrant(i,j))
if QC.isReferenceTimeSeries(posQLP) then
  return reference
end if
result ← QC.decodeTimeSeries(posQLP, reference)
return result
```

Algorithm 5 describes the `windowQuery` query, which is resolved by a row-wise traversal that reuses the recovery of the quadrant reference series while it remains useful. The cells within the query area are traversed in row-major order, verifying the type of series in each cell in the same order as in the two previous queries. The difference lies in how the reference series is retrieved for cells that are not *fixed time series*. A map structure is used to store the reference series for quadrants encountered during the row-wise traversal. Each time a new row is entered, it is verified whether a new quadrant has also been entered, in which case the references stored in the map can be discarded. Thus, the modification of the previous procedure consists of checking whether the quadrant's reference series is already stored in the structure, and retrieving it from there. If the series is not in the map, it is retrieved from its encoded format, used to decode the queried cell values, and then stored in the map.

4.4.2. GroupComp: Group Compress

It is evident that the structure described in Section 4.4.1, due to the type of neighborhood it uses, only applies to *dense* datasets. "GroupComp" corresponds to an adaptation of the structure for *sparse* data, such as the `eeg_data` and `madrid_data` datasets. In general terms, *GroupComp* uses the same elements

Algorithm 5 Algorithm windowQuery($QC, r_s, r_e, c_s, c_e, t_s, t_e$)

```
result ← ∅
referencesMap ← ∅
dq ← QC.quadrantdimension
for i ← r_s to r_e do
  if i MOD dq = 0 {It's a new quadrant row} then
    referencesMap ← ∅
  end if
  for j ← c_s to c_e do
    posQLP ← QC.getLinealPositionOfQuadrant(i,j)
    if QC.isFixedTimeSeries(posQLP) then
      result.add(QC.decodeFixedSeries(posQLP))
    else
      quadID ← QC.getQuadrant(i,j)
      if quadID ∉ referencesMap then
        referenceMap[quadID] ← QC.decodeReferenceSeriesOfQuadrant(quadID)
      end if
      if QC.isReferenceTimeSeries(posQLP) then
        result.add(referenceMap[quadID])
      else
        result.add(QC.decodeTimeSeries(posQLP, referenceMap[quadID]))
      end if
    end if
  end for
end for
return result
```

described in Section 4.4.1.1 for *QuadComp*, but with important differences that allow the neighborhood between the data to be specified directly. In the context of sensors that can be represented as a graph, there is not always a strict Euclidean distance relationship between the elements. This can be seen in the traffic sensors, where vehicles recorded by one sensor will not necessarily be detected by one of the closest sensors, because they are probably located in a different lane. On the other hand, it is very likely that the vehicle will pass by the next sensor on the same lane even if the distance is greater.

It is called *GroupComp* because instead of considering quadrants of a grid with time series, the data are grouped and encoded with respect to one of the time series in their group. The grouping strategy may vary from one dataset to another but, in general, it is related to the *semantics* used to define proximity. Using the `madrid_data` dataset as an example, a group corresponds to the set of sensors that share some common characteristic by which they can be grouped, such as the description made in Subsection 4.3.2.3 where 5 subsets are formed based on the lane type where the sensors are located.

Regarding neighborhood in the structure, this information is specified during its construction, indicating the number of groups and the list of identifiers of the time series in the group. In the implementation of this variant of the structure, there are slight differences in the way the data is provided for construction, but they align with this general description.

4.5. Experimental Evaluation

This section describes the considerations for conducting the experimental tests and analyzes the results obtained. The goal is to evaluate the performance of the proposed data structures using the real datasets previously described in Section 4.3.

Subsequently, different baselines are evaluated for the datasets according to their characteristics. In the case of *dense* datasets, the baseline is the temporal k^2-raster data structure, which is widely used due to its good performance on raster data with temporal aspects. For *sparse* datasets, the comparison baselines include the following numerical sequence compressors: `elias_delta`, `elias_gamma`, and `fibonacci`. The comparison with these compressors involves different combinations

of differences, aiming to reduce the numerical range of the values.

Given the different datasets, their respective auto-correlation measures, and the various baselines, a data structure is proposed to store the time series. The obtained results are contrasted with the results from the respective baselines defined for each dataset. The goal is to identify a relationship between the auto-correlation values of the data and the performance of different space-efficient representation strategies. In this way, it becomes possible to determine, based on the dataset indices, the most efficient way to represent the data.

The implementation used is available on GitHub[6].

4.5.1. Experimental Results

The experimental evaluation considers different baselines depending on the nature of the dataset. The following subsections present the results obtained for each dataset individually.

4.5.1.1. Experimentation on `nasa_data`

As noted in Section 4.3.1.1, we consider this dataset to be *dense* and *euclidean*. Therefore, we use *temporal* k^2*-raster* (see details in Section 4.2.2.1) as the baseline for comparison. Its implementation in C++ is available on GitLab[7], developed by Fernando Silva [124]. The following presents a comparison of the results obtained through experimentation. Both the execution times for the queries described in Section 4.4.1, as well as the space required by the structures, are included.

Figure 4.5.1 shows the results regarding the storage size required to represent all attributes of `nasa_data` using the tested structures. The evaluated structures are *temporal* k^2*-raster*, and two variants of *QuadComp*: one that encodes its arrays using `bit compress` (QC_{bc}), and another that uses `fibonacci` (QC_{fibo}). It can be observed that QC_{fibo} achieves a smaller size than *temporal* k^2*-raster* in all cases except for a single instance, namely `CONV`. Some of the situations that arise when comparing the *QuadComp* variants with *temporal* k^2*-raster* are as follows:

- The largest size reduction in favor of QC_{fibo} occurs in `PRES`, with almost 300 [MB], while QC_{bc} requires 200 [MB] less than *temporal* k^2*-raster* for `TMP`.

[6] https://github.com/cquijadafuentes/TempSeriesComp_paper_pub
[7] https://gitlab.lbd.org.es/fsilva/k2-raster

- The worst case for QC_{bc} is in `CAPE`, with nearly 200 [MB] more, while for QC_{fibo} the worst case is `CONV`, requiring 22 [MB] more than *temporal* k^2*-raster*.
- The best proportional reduction for QC_{bc} is seen in `TMP`, where it uses 53,5 % of *temporal* k^2*-raster*'s size, while QC_{fibo} requires only 34 % in `PEVAP`.
- The worst ratio for QC_{bc} is in `CONV`, requiring 5 times more space than the *temporal* k^2*-raster*, while the worst ratio for QC_{fibo} is also in `CONV`, where it uses 160 % of the space required by *temporal* k^2*-raster*.

To represent the entire dataset, *temporal* k^2*-raster* requires 3.5 [GB], whereas QC_{bc} uses 2.9 [GB] (i.e. 82,3 % of the space required by *temporal* k^2*-raster*), and QC_{fibo} requires 2.3 [GB] (i.e. 66,1 % of the space required by *temporal* k^2*-raster*).

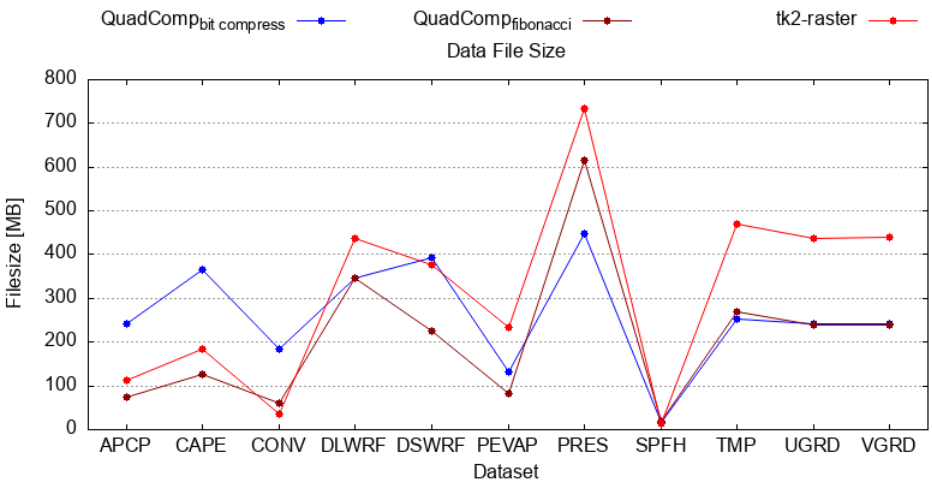
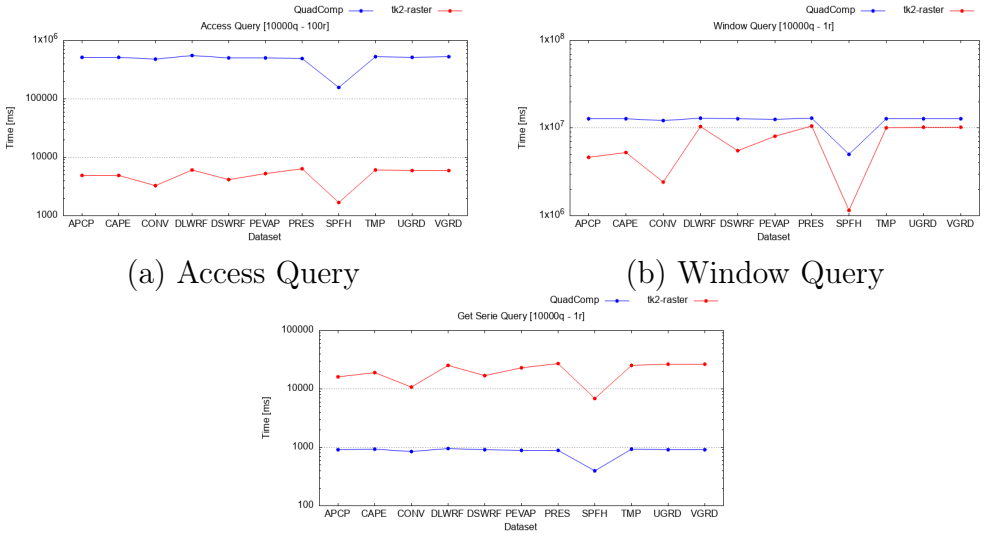


Figura 4.5.1: File sizes in `nasa_data` dataset.

We now present the analysis of the query times. The queries used in this experiment are generated using the scripts available in the *temporal* k^2*-raster* implementation, which generate random queries based on the characteristics of a structure file, ensuring that the dimensions along any axis, or the time interval if applicable, fall within the range of the data.

Figure 4.5.2 shows the comparison results against the *temporal* k^2*-raster* baseline. For `accesQuery`, it can be observed that *temporal* k^2*-raster* has an advantage of nearly two orders of magnitude compared to the execution times of our proposed structure, while maintaining consistent proportions across the evaluated attributes. With respect to the `windowQuery` operation, the *temporal* k^2*-raster* also achieves better query times, although in this case the differences are much smaller. In

some instances, the query times are virtually identical. Finally, the last query, `getSerie`, shows an advantage for our proposal of approximately one order of magnitude. Overall, the query times of the proposed structure show stable behavior across different attributes, except for `SPFH`, which – similarly to *temporal k^2-raster*– requires significantly less time than the queries over the other attributes. This can be explained by the characteristics of the dataset, as the value range includes only three possible values (see Table 4.3.2).



(a) Access Query

(b) Window Query

(c) getSerie Query

Figura 4.5.2: Experimentation on the `NASA` dataset.

In summary, the proposed structure proves to be highly competitive in this dataset, providing a considerable space savings. Regarding query times, although it is clearly outperformed in the single time instant access operation, it offers a significant advantage when retrieving a complete time series. Therefore, depending on the application domain, it can be an interesting alternative. Moreover, the autocorrelation analysis presented in Section 4.3.1.1 helps to better explain the space results of the structure. Recall that this domain exhibits strong autocorrelation across the different dimensions, including spatio-temporal autocorrelation (STA). This explains why it is possible to improve upon the results of a structure like *temporal k^2-raster*, which already exploits STA. However, it can also be observed that for those variables where the difference between STA and purely temporal autocorrelation is greater (APCP and CONV[8]), the differences between the proposed approach and *temporal k^2-raster* decrease.

[8]Disregarding SPFH due to the anomalous characteristics discussed earlier.

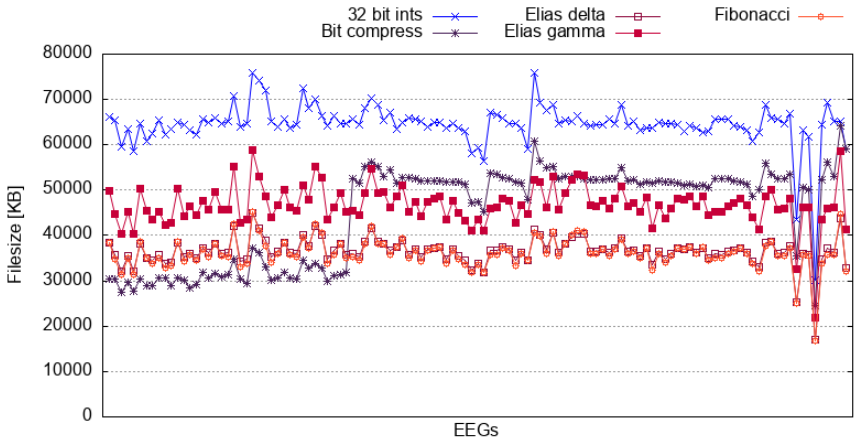
4.5.1.2. Experimentation on `eeg_data`

For `eeg_data`, comparisons are made between the data representation using the *GroupComp* structure and the representation using the encoders available in the SDSL library (`bit_compress, elias_delta, elias_gamma`, and `fibonacci`). For the structure representation, three different group configurations are defined, which are the same as those used to determine the auto-correlation of the dataset in Figure 4.3.9 (b). This is consistent with our characterization of the dataset as *sparse* and *semantic*.

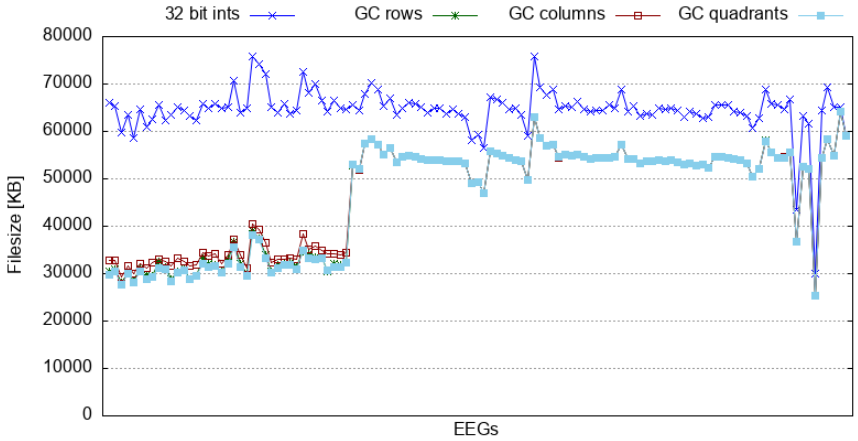
Figure 4.5.3 plots the space required for data representation. First, in Figure 4.5.3 (a), the sizes correspond to the representation using the SDSL library encoders, and then in 4.5.3 (b), the sizes correspond to the *GroupComp* proposal in its three variants. Both graphs include the size used by a 32-bit integer value representation, which also serves as a reference for the length of the samples in each EEG (as mentioned before, not all EEGs have the same number of samples).

In Figure 4.5.3(a), the three variable-length encoders show similar behavior across all EEGs, with `elias delta` and `fibonacci` requiring the least space among them, exhibiting almost identical behavior. Finally, `bit compress` exhibits irregular behavior compared to the other elements in the same graph: for the first EEGs, it requires less space than the variable-length encoders, but for the remaining ones, it uses more space. This behavior makes sense considering that the EEGs represented along the X-axis of the graph maintain the same order as the graphs in Section 4.3, sorted by their standard deviation. In turn, EEGs with lower standard deviation have a smaller range of values in their recordings. Therefore, `bit compress` uses fewer bits to represent the values contained in each sensor's array.

Then, Figure 4.5.3(b) shows the size required to represent the data using *GroupComp* for three neighborhood variants: grouped by rows, by columns, or by 3×3 quadrants (equivalent to a queen's criterion). The three cases exhibit similar behavior among them, where the `GC columns` representation is slightly less efficient for EEGs with lower standard deviation. On the other hand, the space required by the proposal is very similar to that required by `bit compress`, where EEGs with higher standard deviation require more space to be represented by GC (in any of its neighborhood variants) than the `elias_gamma` encoder.



(a) Encoders Filesizes



(b) GroupComp Filesizes

Figura 4.5.3: File sizes of `eeg_data` dataset.

Complementarily, Figure 4.5.4 has been generated to relate the behavior of the required data representation size versus their standard deviation. The `elias delta` encoder, the `bit compress` representation, and the proposed method in its `GC rows` variant are included. In the graph, it can be seen that the `elias delta` encoder does not show a variation in file size related to the EEG standard deviation. In contrast, `GC rows` and `bit compress` show larger representation sizes as the standard deviation increases, exhibiting a similar behavior. This behavior could be due to the increase in the value range presented by elements with a high standard deviation, as mentioned in Subsection 4.3.1.2. Meanwhile, the encoding by `elias_delta` is less affected by large integer values due to its variable-length encoding method.

Finally, it is possible to establish that the *GroupComp* proposal shows better

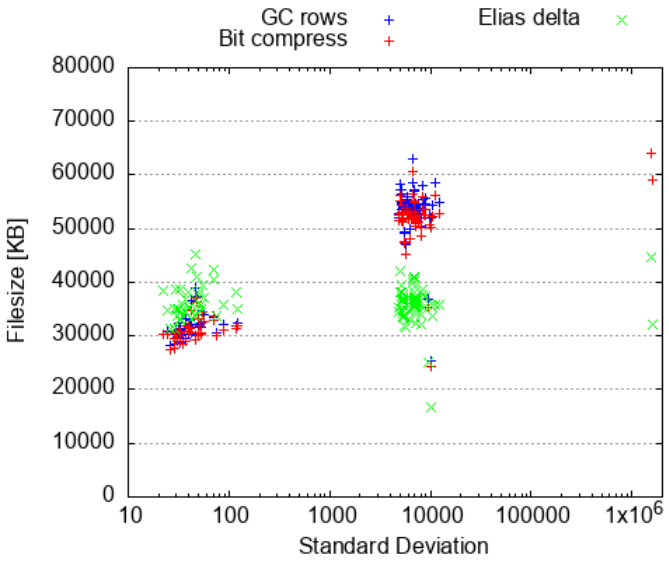


Figura 4.5.4: Scatter plot of file size versus standard deviation in `eeg_data` dataset.

performance in terms of the space required for representation when the temporal auto-correlation of the data is high, which coincides with low standard deviation values. On the other hand, there seems to be no relationship regarding the values of STA.

4.5.1.3. Experimentation on `madrid_data`

For the `nasa_data` dataset, *GroupComp* is also compared against the SDSL encoders, following the same approach as in the previous subsection for *sparse* and *semantic* datasets. In this case, the structure uses the `5groups` configuration from Figure 4.3.8, which is based on the lane and direction associated with the sensor locations.

Figure 4.5.5 shows the space usage results for the structure and the compressors on the `madrid_data` data. The X-axis displays the 4 attributes of the dataset, while the Y-axis represents the size in kilobytes. First, the 32-bit integer representation maintains a high space usage with a slight increase towards the end, for the `velocidad` attribute. This matches the number of sensors after preprocessing, where `intensidad, ocupacion, carga`, and `velocidad` have 240, 241, 241, and 232 sensors with valid samples, respectively. The variable-length encoders exhibit very similar behavior across all attributes, with `fibonacci` showing the best performance among the three. The fixed-length encoder `bit compress` achieves

the best results among all methods in the chart for three out of the four attributes, but shows the worst performance for the fourth attribute. The `GroupComp` structure shows better performance than the variable-length compressors, being the best option for the `ocupacion` attribute and very close to `bit compress` in the others.

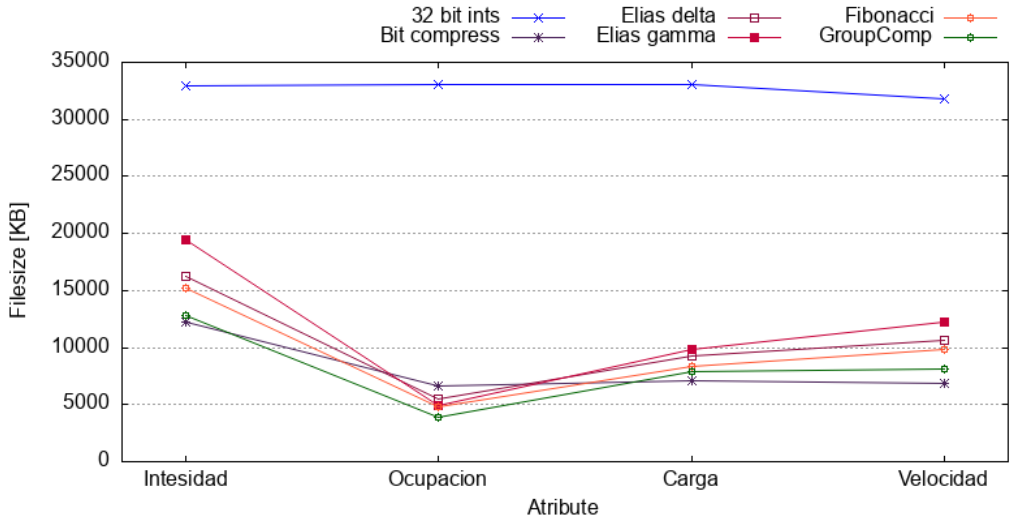


Figura 4.5.5: File sizes of `madrid_data` dataset.

In `madrid_data` it is difficult to indicate any relationship with either spatio-temporal or temporal auto-correlation. The best performance of the *GroupComp* structure relative to the comparison baselines corresponds to the attribute `ocupacion`, which also has the lowest STA value for the neighborhood configuration used in the experiments. The attribute with the highest auto-correlation, both temporal and spatio-temporal, is `intensidad`, and its best representation is achieved using `bit_compress`, followed by *GroupComp*.

4.6. Conclusions and Future Work

This work presents a study on spatially-embedded time series, i.e. time series associated with spatial locations across different domains. First, three datasets are characterized with an emphasis on evaluating their autocorrelation to examine its relationship with space-efficient representation. Additionally, a data structure is proposed in two variants, each tailored to the two types of datasets considered.

Specifically, the contributions presented here are:

- Three datasets are described using temporal auto-correlation and STA on:

time series in raster meteorological data of a territory, time series from EEG exams, and time series measuring traffic variables. The analysis results show that the first dataset presents a high STA for all attributes, even exceeding the values of its temporal auto-correlation. Among the three datasets, it is the only one presenting this situation. Its counterpart, the time series in graphs, shows lower STA values than temporal, not exceeding the index of 0,5. This indicates that, in general, the time series located in a given position does not behave similarly to the surrounding elements. Based on this, two types of datasets are identified: *dense data*, where positions are contiguous to each other, and *sparse data*, where data have a more dispersed distribution among elements and can be related by criteria other than Euclidean distance between their positions.

- A data structure is proposed for representing time series in two variants; to support both *dense data* with *QuadComp*, and *sparse data* with its variant *GroupComp*. The *QuadComp* structure for *dense data* is directly comparable with the compact data structure *temporal k^2-raster*. While the variant for *sparse data*, *GroupComp*, is compared with the representation using integer value encoders available in the SDSL library.
- An experimental evaluation of both structures is performed in terms of space required to represent the data and response time for three queries. The results are positive regarding space, while for query response the performance drops by up to 2 orders of magnitude difference. On the other hand, the structure for graph data shows inferior storage performance when compared to integer value encoders.

QuadComp shows good results in representation size for *dense data*, achieving up to 53 % compression for one of the attributes in the `nasa_data` dataset, and 82.3 % for the whole dataset. This dataset is characterized by having higher STA than temporal auto-correlation. In contrast, the *sparse data* exhibit higher temporal auto-correlation, and their space usage does not outperform the variable-length encoders used in the experiments. Thus, a relationship is established between the possibility of leveraging a data structure for efficient representation and the STA of the data.

Currently, the reference series used to represent a quadrant (or group according to the variant) is directly taken from one of the elements to be represented. The

study of alternative ways to represent elements based on references remains open, whether by using a different encoding strategy or by generating a reference element distinct from the elements being represented. Another possible line of exploration is the way groups are established for the second variant, aiming to achieve better experimental results. Currently, the structure follows the specified pattern, but a data-driven suggestion could be introduced. Thus, a heuristic based on metrics such as local auto-correlation could be used to determine dataset groups and their elements.

Capítulo 5

Conclusiones y Trabajo Futuro

En este capítulo se entrega el análisis final del trabajo de tesis titulado “Representación eficiente de secuencias espaciales y espacio-temporales". Se realiza una reflexión de las propuestas expuestas a través de las contribuciones realizadas, y una discusión sobre posibles líneas de trabajo futuro.

5.1. Discusión

En la primera etapa de este trabajo se desarrollan los conceptos previos para facilitar la comprensión del contexto en el que se trabaja. Se inicia con algunos conceptos básicos de estructuras de datos compactas, para luego dar una revisión de la literatura respecto de las representación y procesamiento de datos espacio-temporales, indicando los modelos de representación de objetos espaciales, luego la introducción del concepto de trayectoria que corresponde a datos espaciales con dimensión temporal, algunas formas de representación y relaciones que existen entre ellas donde se incluye también las relaciones entre segmentos de línea, para finalizar con propuestas eficientes para la representación de trayectorias.

Posteriormente, entendiendo las relaciones topológicas en la dimensión espacial de las trayectorias fue posible definir una aproximación para la representación de un conjunto de trayectorias restringidas en un modelo en red, la cual se detalla en la sección de Anexos 6.2. Esta estructura permite responder consultas sobre las existencia de ciertas relaciones topológicas entre pares de elementos del conjunto. Esta propuesta inicial, a pesar de contener un conjunto completo de caminos

representados, no explota el potencial de la estructura base utilizada, ya que se aplica para la verificación de una relación topológica determinada entre un par de caminos, entregando como respuesta si dicha relación existe o no.

Para aprovechar el potencial presente en la estructura base utilizada es que surge la propuesta *TRGST*, descrita en el Capítulo 3. Esta estructura, ademas de almacenar todo el conjunto de caminos, permite recuperar todos aquellos caminos del conjunto que cumplen con una determinada relación topológica respecto de un camino del conjunto. En el capítulo que se presenta dicha estructura, se da a conocer en detalle los elementos que componen la estructura, los algoritmos para responder consultas de relaciones topológicas, y los resultados experimentales obtenidos.

Su experimentación usa tres conjuntos de datos, el más grande fue generado en base a la red de transporte público de Madrid, el segundo fue derivado a partir del primero para restringir el largo de los caminos representados, y finalmente un tercer conjunto con datos reales basados en recorridos de taxi en Shangai. Las consultas que aquí se implementaron fueron: `Contains, Within, Equals`, e `Intersects`. De manera general, la estructura propuesta mostró buenos resultados en todos los datasets tanto para las consultas como para la representación de la información. Respecto del tiempo al incrementar el número de elementos del conjunto, la operación `Equals` muestra una ventaja de hasta 3 órdenes de magnitud para *TRGST*, y un comportamiento casi constante para la consulta `Contains`. Al evaluar el largo de los caminos representados la propuesta es mejor en tres de las 4 operaciones. La propuesta presentó resultados con hasta 3 órdenes de magnitud de ventaja respecto del baseline utilizado (el mismo de la exploración inicial).

Finalmente, el Capítulo 4 desarrolla el trabajo sobre secuencias espacio-temporales, que se compone del análisis de conjuntos de datos reales, su caracterización, dos estructuras de datos para la representación de secuencias espacio-temporales, algoritmos para las consultas, y finalmente resultados experimentales con los datos reales.

En la exploración de los datos se hace énfasis en el análisis de la auto-correlación que presentan, tanto en la dimensión temporal como en la espacio-temporal. Basados en estos resultados y en las características propias del contexto de los mismos datos fue posible identificar 2 tipos de conjuntos, denominados *esparsos*

y *densos*. Esta caracterización, junto con el tipo de vecindad que se define para cada conjunto, guía el desarrollo de 2 estructuras, una para cada tipo de datos.

La primera estructura de datos para secuencias espacio-temporales fue *QuadComp*, la cual fue diseñada para datos *densos*, como puede ser el conjunto de datos meteorológicos denominado `nasa_data`. *QuadComp* aprovecha la similitud espacial presente en los datos, donde celdas próximas entre sí tienden a contener secuencias de datos similares. Así, esta estructura divide el espacio en cuadrículas y selecciona una secuencia de referencia para cada cuadrícula, a partir de la cual serán representadas las demás series de la misma cuadrícula.

En la evaluación experimental de *QuadComp* se realiza la evaluación por medio de la comparación contra la estructura de datos compacta para datos espacio-temporales *temporal k^2-raster*. La estructura propuesta muestra buenos resultados respecto del espacio requerido, usan un 82,3 % del espacio necesario por *temporal k^2-raster* para representar todos los datos de `nasa_data`. En cuanto a las consultas, en la recuperación de una secuencia del conjunto presenta resultados con un orden de magnitud de ventaja. Por otro lado, surge la oportunidad de mejorar el rendimiento de las otras consultas, sobre todo para *Access Query*, que presenta una desventaja de casi dos órdenes de magnitud respecto de *temporal k^2-raster*.

En el caso de los datos *esparsos*, sus características muestran comportamientos diferentes respecto de los datos *densos*. En primer lugar se puede apreciar que en algunos casos presentan correlaciones temporales mucho más altas que la correlación espacio-temporal de su conjunto. Así también se puede ver una relación entre la desviación estándar de las secuencias y la auto-correlación temporal. Para este tipo de datos se desarrolla la estructura *GroupComp*, que tiene como principal diferencia la forma en que se agrupan los datos. Mientras que *QuadComp* realiza una división automática de la grilla en cuadrantes de igual dimensión, *GroupComp* permite la definición de grupos de secuencias, entregando la flexibilidad de determinar bajo qué criterio se definen los grupos.

En la categoría de datos *esparsos* se utilizaron dos conjuntos: `eeg_data` y `madrid_data`, que corresponden a grupos de datos de exámenes de electroencéfalogramas y datos de sensores de tránsito de la carretera `M30` de la ciudad de Madrid, España. En el conjunto `eeg_data` la estructura *GroupComp* muestra un buen rendimiento en cuanto al tamaño necesario para la

representación en un subconjunto de los EEG, particularmente en aquellos que presentan una desviación estándar baja en sus secuencias. Mientras tanto, para el conjunto `madrid_data` los resultados muestran un buen rendimiento para uno de los atributos del conjunto, correspondiente a `Ocupacion`. En general, la estructura *GroupComp* requiere de una mejor estrategia que permita mejorar el rendimiento en el espacio para la representación de los datos.

Finalmente, este trabajo aborda con éxito el diseño y análisis de representaciones eficientes para la consulta de secuencias espaciales y espacio-temporales en dos dominios: datos de trayectorias y series de tiempo asociadas a ubicación espacial. Se detallan las estructuras propuestas, sus algoritmos para consultas relevantes, y la evaluación experimental utilizando datos reales, comparando los resultados con estructuras representativas del estado del arte.

5.2. Trabajo futuro

A continuación se describen líneas de trabajo futuro que quedan abiertas para dar continuidad a la investigación realizada en esta tesis.

Para la propuesta *TRGST* que permite la representación de un conjunto de secuencias espaciales restrigidas a una red y la recuperación de elementos que cumplen cierta relación topológica, se hace necesario el buscar una estrategia que permita mejorar el rendimiento en las consultas donde no existe contención entre las secuencias espaciales representadas. Una línea interesante que se abre directamente es el abordar secuencias espaciales de objetos en movimiento en el espacio libre. Esta línea incrementa la complejidad del problema al considerar que se incrementa la dimensión espacial, por lo que la estrategia de representación debe ser capaz de tratar con al menos una dimensión adicional.

Así mismo, tanto para trayectorias dentro de un modelo en red o espacio libre, existe la posibilidad de considerar, en un paso futuro, la integración de la dimensión temporal en los datos registrados. Actualmente la dimensión temporal está parcialmente considerada al representar la trayectoria en base a secuencias de la ubicación espacial, pero no toma completamente dicha dimensión al no considerar el instante de tiempo específico en el que un objeto está en una ubicación. Esta posible línea requiere extender el trabajo al campo de relaciones temporales, para luego realizar la combinación con las relaciones topológicas que se han descrito en

la Sección 2.2.3.

Por otro lado, para la línea de representación de secuencias espacio-temporales existen varios aspectos mejorables en el trabajo. Respecto de los conjuntos de datos, sería interesante evaluar una mayor cantidad de datasets que permitan corroborar la caracterización aquí entregada, o extenderla en base a los resultados obtenidos por diversos análisis.

Respecto de las estructuras en sí se vuelven interesantes dos aspectos: la definición de la secuencia de referencia, y la forma en que se representa una secuencia en base a su referencia, para las estructuras *QuadComp* y *GroupComp*, respectivamente. Sería interesante establecer una estrategia de selección de la secuencia de referencia de un cuadrante (para *QuadComp*) que permita tomar la mejor opción dentro del conjunto. Durante las pruebas desarrolladas se utilizaron exploratoriamente distintas estrategias pero no mostraron diferencias notables en los resultados. En el caso de *GroupComp* la representación no muestra grandes ventajas en el tamaño de la representación, y tal vez es posible hallar la manera de explotar la similitud temporal antes que la espacio-temporal. Esto basado en los resultados de autocorrelación temporal obtenidos en el análisis de los datos.

Finalmente, considerando que las propuestas presentadas en esta tesis hacen uso de estructuras de datos compactas para optimizar el espacio de representación, surge como un punto de interés futuro el estudio del dinamismo en estas estructuras. Dentro del ámbito de las estructuras compactas, existen variantes que permiten modificar la información contenida, ya sea mediante actualizaciones, eliminaciones o inserciones de datos. Este dinamismo podría abordarse, en términos generales, por dos vías: empleando estructuras base que ya ofrecen soporte dinámico, o bien adaptando las estructuras actuales para que permitan modificaciones eficientes. Explorar este aspecto representaría una línea de trabajo interesante para las tres estructuras desarrolladas en esta tesis.

Todas las líneas futuras recién descritas pueden presentar resultados interesantes para la comunidad, dado el grado de interés que existe en los datos espacio-temporales utilizados para la experimentación de este trabajo.

Capítulo 6

Anexos

En este Capítulo se incorpora material de apoyo adicional que ha sido movido desde las secciones anteriores del documento para facilitar la lectura. Primero, se muestran todos los esquemas que componen los modelos de representación de relaciones topológicas entre segmentos de línea. Y luego, se entrega una descripción detallada de la versión preliminar a la estructura contenida en el Capítulo 3.

6.1. Ejemplos de Relaciones Topológicas entre Segmentos de Línea

A continuación se detallan las matrices de intersección utilizadas por los modelos de representación de relaciones topológicas que han sido mencionadas en la Sección 2.2.3.

6.1.1. QTC

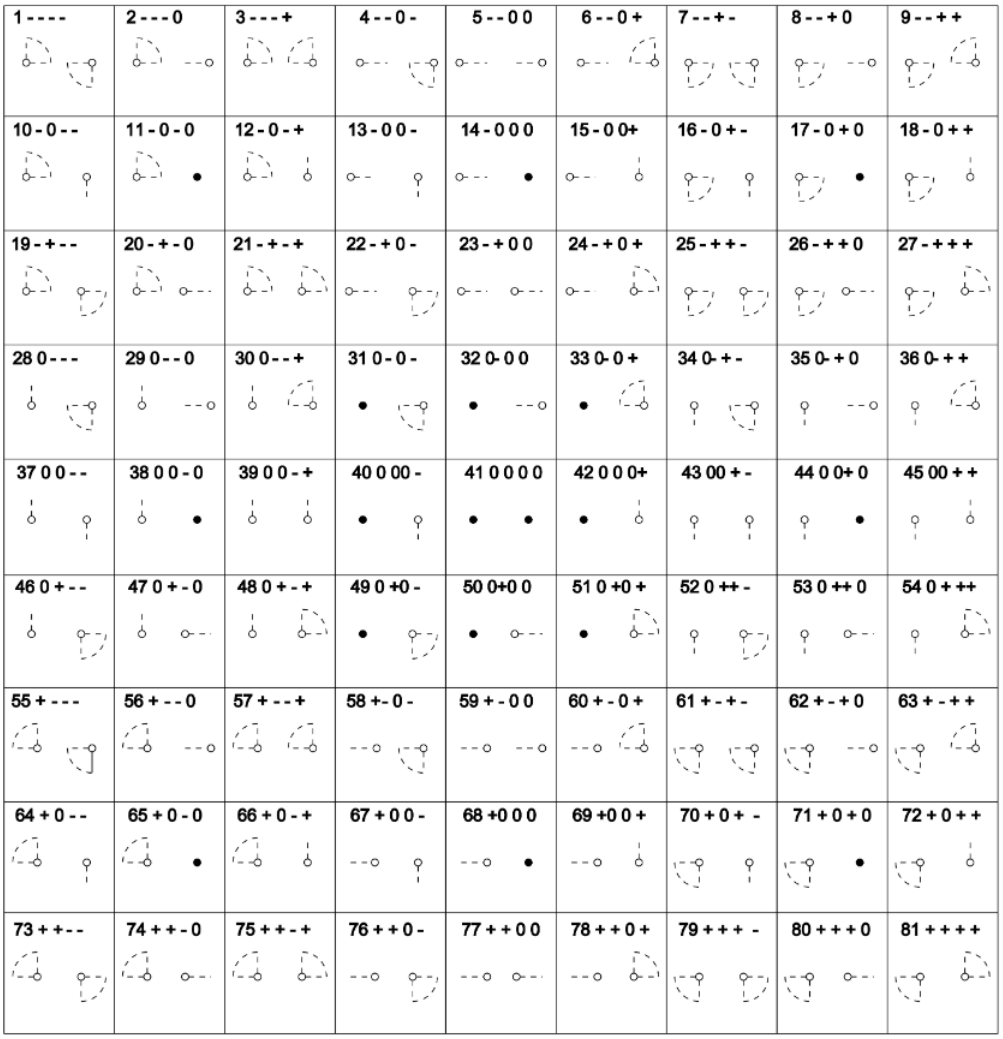


Figura 6.1.1: Íconos de relaciones en QTC$_c$ [9]

6.1.2. 9-IM

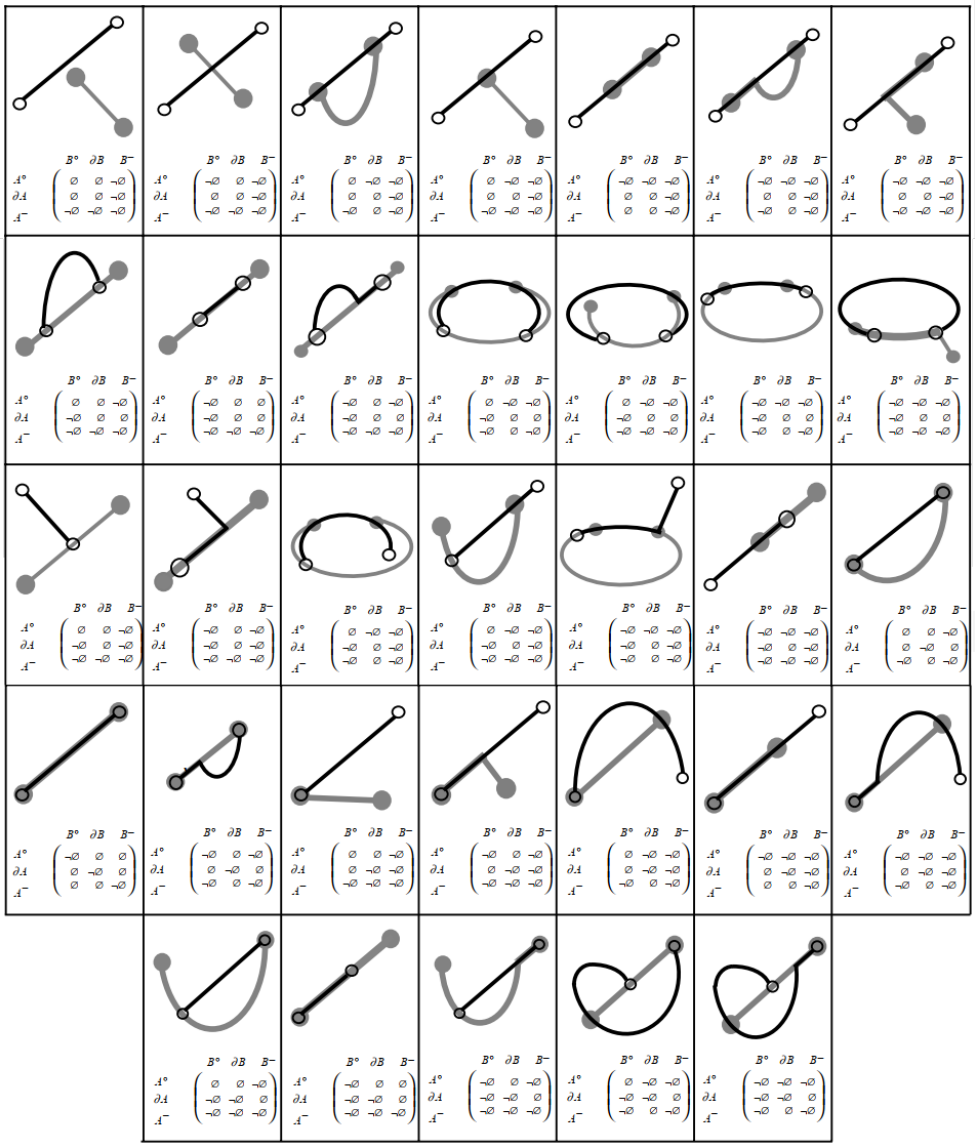


Figura 6.1.2: Relaciones topológicas posibles entre dos líneas simples [45], cuyos elementos se representan por A° (interior), ∂ (bordes) y A^- (exterior).

6.1.3. head body tail model

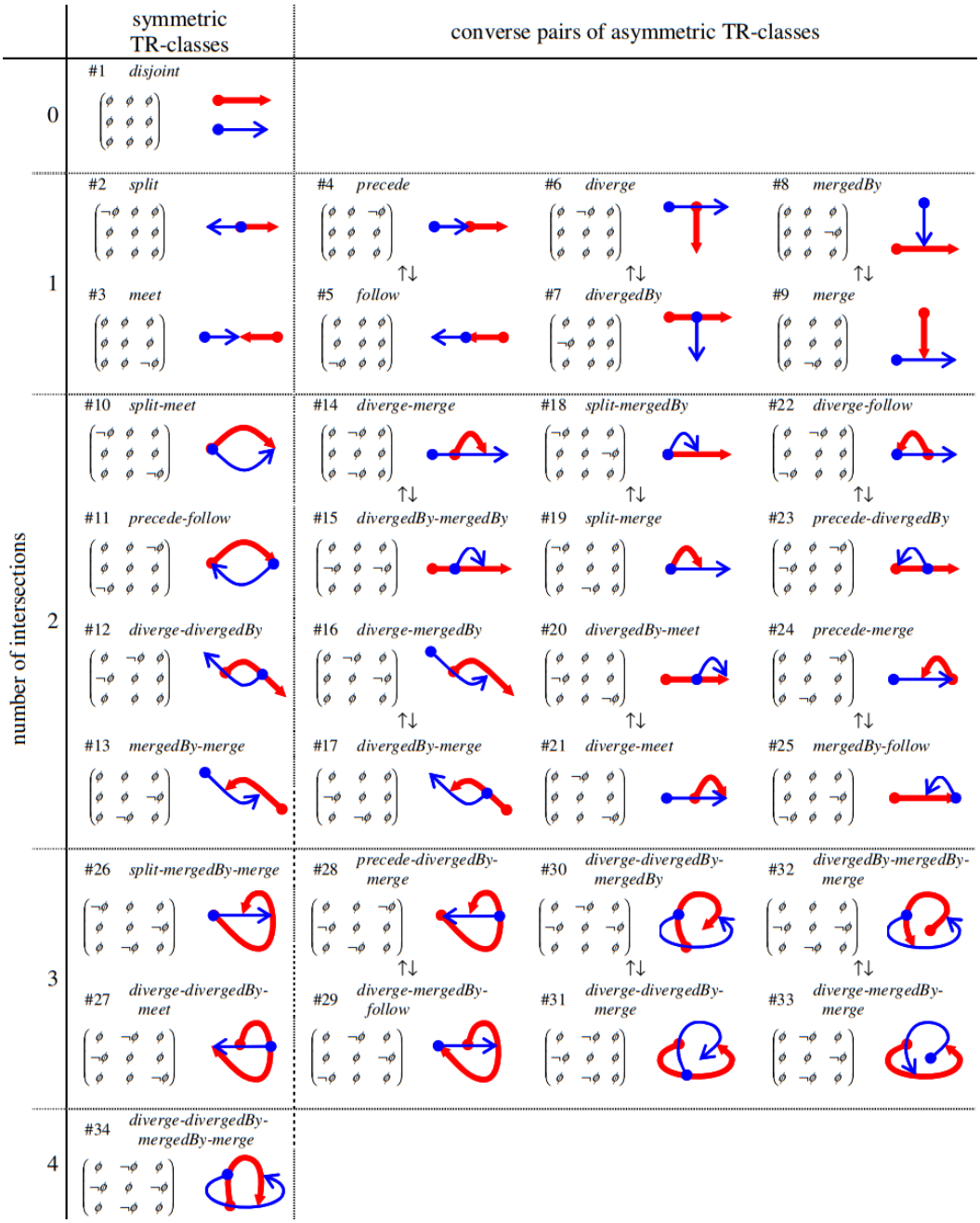


Figura 6.1.3: Clases de relaciones topológicas sin intersección interior-interior [72] donde los elementos que se consideran son punto inicial, interior y punto final.

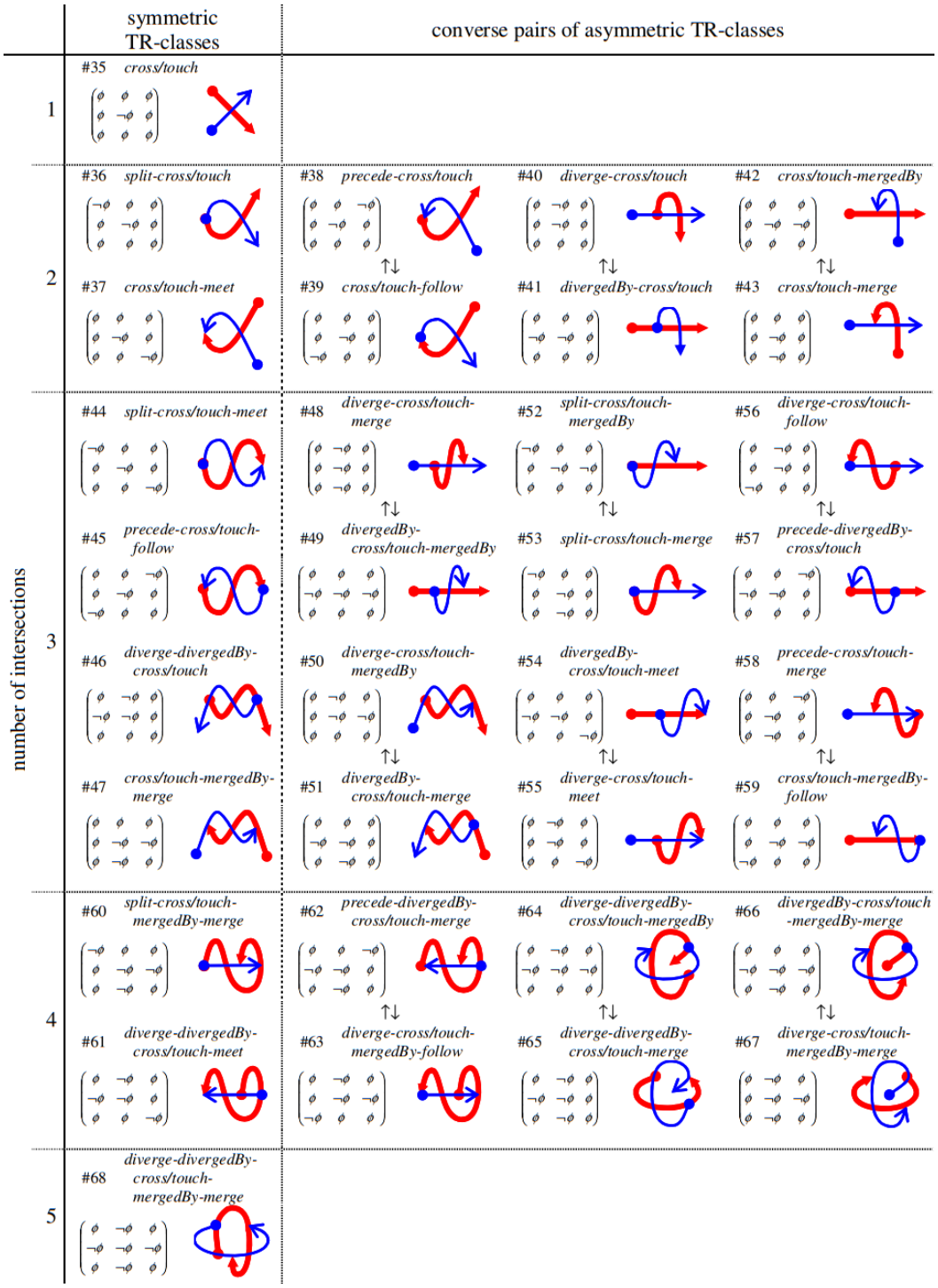


Figura 6.1.4: Clases de relaciones topológicas con intersección interior-interior [72] donde los elementos que se consideran son punto inicial, interior y punto final.

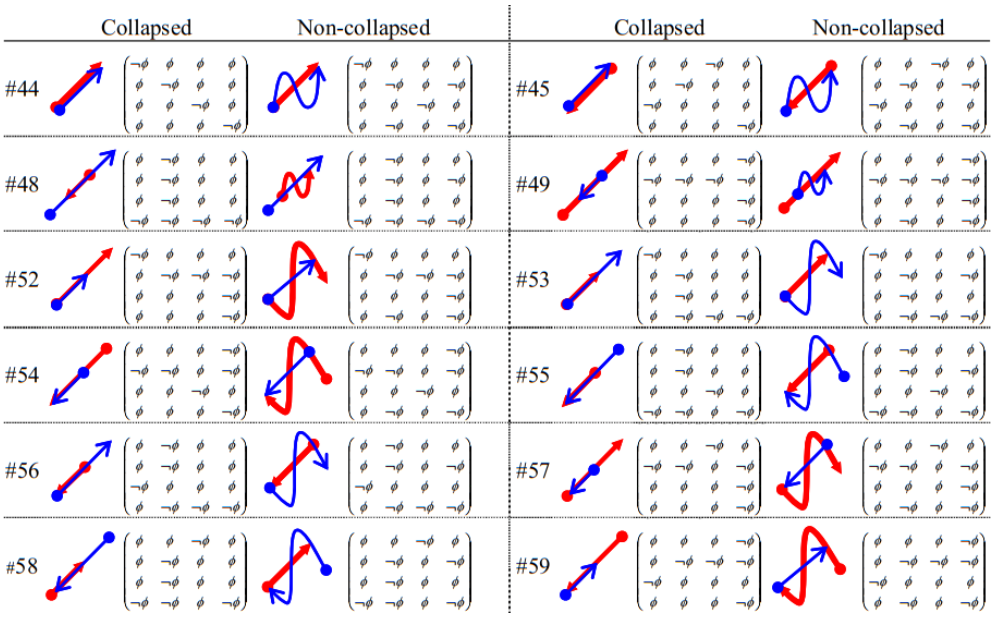


Figura 6.1.5: Clases de relaciones topológicas hbt^+ [72] donde los elementos que se consideran son punto inicial, interior, punto final y exterior.

6.1.4. Espacio cíclico

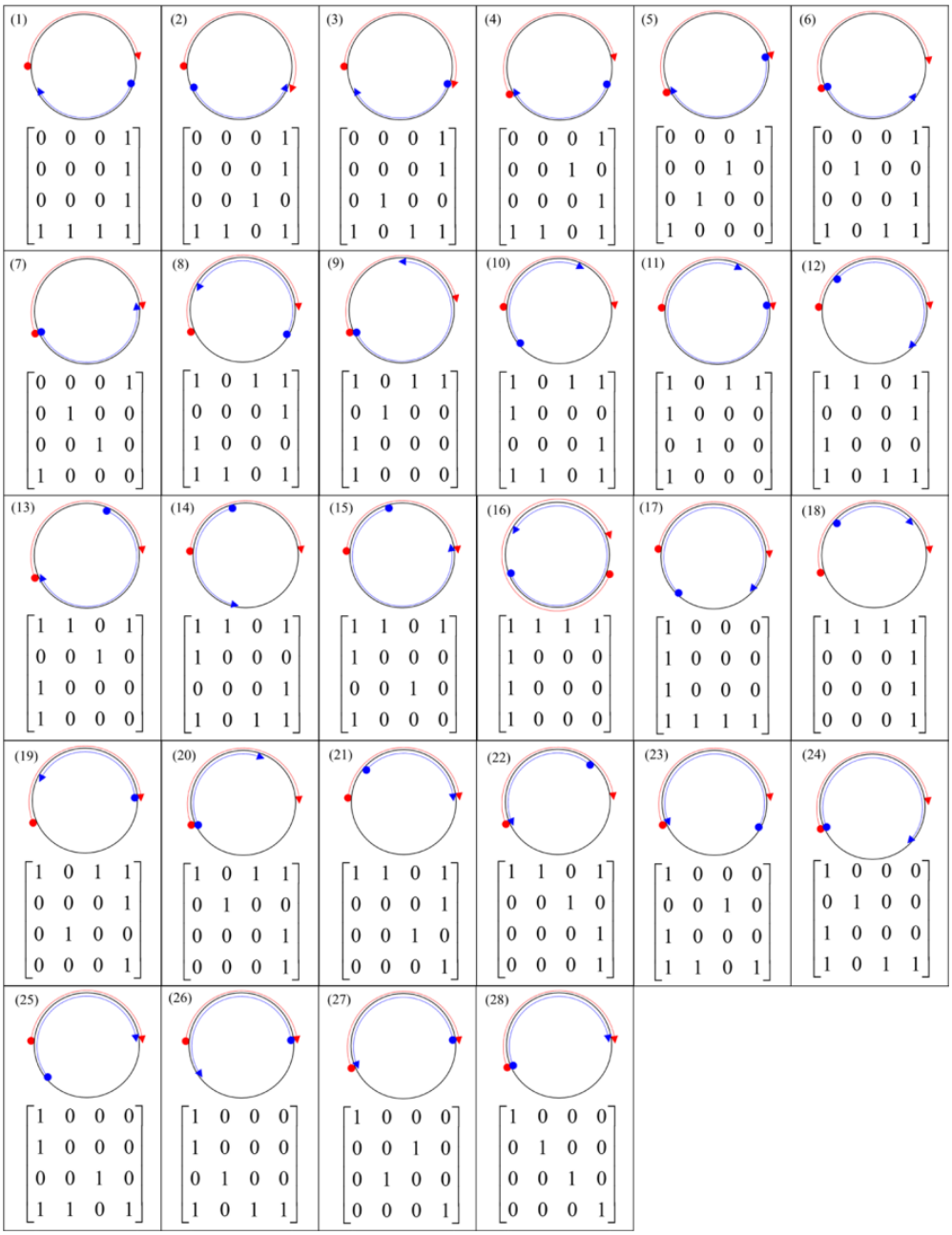


Figura 6.1.6: Relaciones topológicas en espacio cíclico sin coincidencia en extremos de segmento de línea [122].

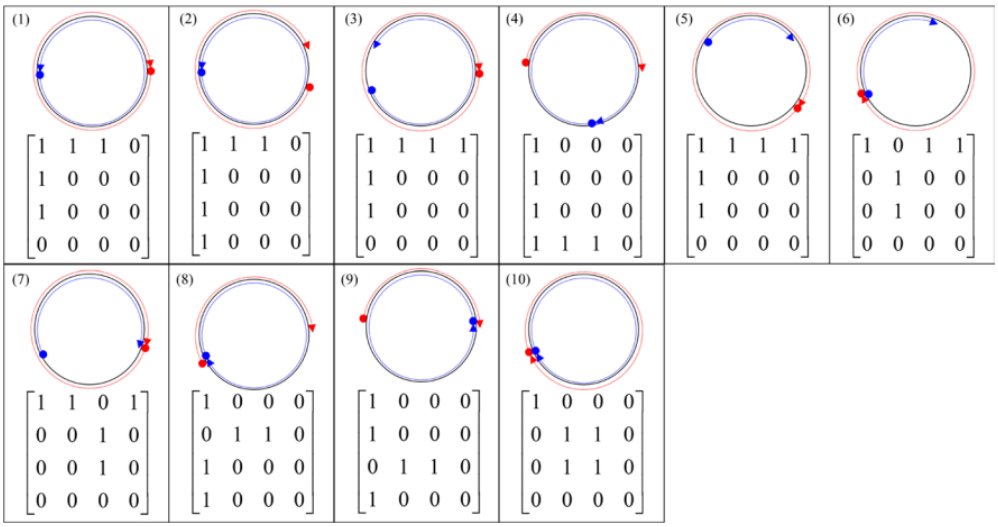


Figura 6.1.7: Relaciones topológicas en espacio cíclico con coincidencia en extremos de segmento de línea [122].

6.1.5. Relaciones Topológicas con Detalles Métricos

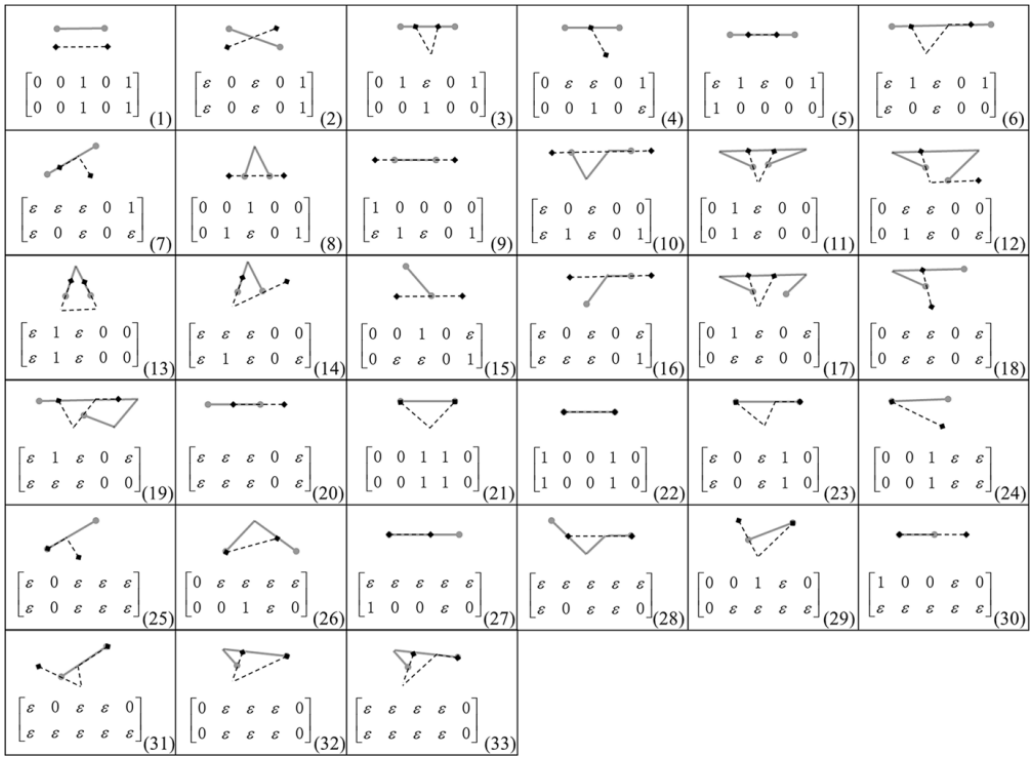


Figura 6.1.8: Relaciones Topológicas entre dos líneas simples representadas por TRM-MD [123].

6.2. GST para Consultas Binarias de Relaciones Topológicas

A continuación se describe la propuesta para identificar relaciones espaciales en trayectorias representadas en red. En primer lugar se describen las relaciones topológicas a identificar, luego se proponen algoritmos sobre una implementación ingenua y una que hace uso de estructuras de datos compactas. Finalmente se muestran los resultados experimentales de la implementación de los algoritmos.

6.2.1. Introducción y Trabajo relacionado

En la Sección 2.2.3 se describen trabajos que permiten la identificación de relaciones topológicas entre segmentos de línea dentro de distintos escenarios y con diferentes métricas. A continuación se menciona una propuesta en la que se agrupan las relaciones topológicas y luego una sección que permite identificar algunas de las relaciones en base a la similitud del texto que podría representar una trayectoria representada como una secuencia.

6.2.1.1. Relaciones topológicas agrupadas

Se han descrito una serie de propuestas en las que se definen y ejemplifican relaciones topológicas entre segmentos de línea. Para reconocer y distinguir de manera acotada las relaciones topológicas es posible agruparlas en 8 categorías distintas: *disjoint*, *meet*, *overlap*, *inside*, *contains*, *covers*, *coveredby* y *equals* [120].

La definición estándar del OGC (Open Geospatial Consortium), considera las siguientes relaciones entre objetos espaciales: *ST_ Contains*, *ST_ Crosses*, *ST_ Disjoint*, *ST_ Equals*, *ST_ Intersects*, *ST_ Overlaps*, *ST_ Touches*, *ST_ Within*, *ST_ Covers*, *ST_ CoveredBy* y ST_ContainsProperly. La Figura 3.2.1 muestra un esquema que propone una jerarquía entre entre todos los elementos del conjunto relaciones topológicas binarias τ antes mencionadas [10]. En la parte inferior del esquema se encuentran las 8 relaciones básicas: *Overlaps (O), CoveredBy (CB), Inside (IS), Equals (EQ), Covers (CV), Includes (IC), Touches (TO)*, y *Disjoint (D)*. Además de las relaciones básicas hay otras relaciones que consideran los lenguajes de consulta como relaciones derivadas que corresponde a una agregación de las básicas. Es el caso de *Intersects (IT), Within (W)*, y *Contains*

(C). Adicionalmente se considera en la gráfica la relación *IDisjoint (IDC)* que corresponde a la disyunción de interiores entre dos elementos. Los arcos en la gráfica representan que la relación topológica representada en el inferior está contenida en la superior, así se tiene que, por ejemplo, la relación *TO* está contenida en la relación *IDC*, al igual que *D* también lo está.

En la Tabla 6.2.1 se realiza una clasificación de las relaciones topológicas que se consideran relevantes de las propuestas que utilizan matrices de intersección. Dentro de las propuestas se han seleccionado aquellas secciones en las que se consideran líneas simples, sin ciclos. Esto debido a que la propuesta que se expone en la Sección 6.2.2 está pensada como una primera aproximación que trabaja con trayectorias simples.

- 9-IM [45]: Las relaciones se enumeran por orden de filas según la imagen de la Figura 6.1.2 que corresponde a las 33 relaciones topológicas posibles entre dos líneas simples.
- hbt$^+$ [72]: Las relaciones desde la número 1 a la 34 se presentan en la Figura 6.1.3, luego las relaciones desde la 35 a la 68 se presentan en la Figura 6.1.4, y finalmente las relaciones con el símbolo $^+$ corresponden a la forma "Collapsed" que se muestra en la Figura 6.1.5 donde se considera la intersección de dos líneas simples considerando su exterior.
- TRM-MD [123]: Muestra las relaciones del 9-IM para dos líneas simples en la Figura 6.1.8 numeradas desde 1 a 33.

6.2.1.2. Determinar similitud entre secuencias de texto

Los algoritmos de similitud de cadenas de texto se pueden dividir en 3 tipos, basados en distancia de edición, basados en tokens y basados en secuencias. Esta última opción resulta ser la más conveniente debido a que la comparación se basa en la similitud entre sub-secuencias de los textos.

Un algoritmo de similitud es el *Gestalt Pattern Matching* [61] también conocido como *Ratcliff/Obershelp Pattern Recognition*, el que entrega un valor que indica la similitud de las cadenas por la fórmula:

$$D_{ro} = \frac{2K_m}{|S_1| + |S_2|} \tag{6.2.1}$$

	9-IM[45]	hbt$^+$[72]	TRM-MD[123]
Overlaps	2,6,7,10,13,14, 16,19,20,23,25, 28,31,33	35,36,37,38,39, 40,41,42,43,44, 45,46,47,48,49, 50,51,52,53,54, 55,56,57,58,59, 60,61,62,63,64, 65,66,67,68,	2,6,7,10,13,14, 16,19,20,23,25, 28,31,33
CoveredBy	30	53^+,55^+,56^+,59^+	30
Inside	9	48^+	9
Equal	22	44^+,45^+	22
Covers	27	52^+,54^+,57^+,58^+	27
Includes	5	49^+	5
Touches	3,4,8,11,12, 15,17,18,21, 24,26,29,32	2,3,4,5,6,7,8,9, 10,11,12,13,14, 15,16,17,18,19, 20,21,22,23,24, 25,26,27,28,29, 30,31,32,33,34	3,4,8,11,12, 15,17,18,21, 24,26,29,32
Disjoint	1	1	1

Tabla 6.2.1: Relaciones topológicas por modelo agrupadas en las 8 relaciones topológicas básicas.

Donde K_m es el número de caracteres que coinciden. Esto se calcula basado en el substring común más largo de manera recursiva con las cadenas que no coinciden. Esta operación puede ser $\Theta(n^3)$ en el peor caso, y $\Theta(n^2)$ en el caso promedio.

Reflexionando sobre el problema, para reconocer la intersección basta con identificar la subsecuencia común más larga que tiene complejidad temporal $\Theta(nm)$ con programación dinámica y $\Theta(n+m)$ con un suffix tree genérico. Estos algoritmos funcionan reconociendo en un sentido, por lo que sería necesario ejecutarlos dos veces invirtiendo una de las secuencias que definen la trayectoria.

Approximate String Matching using a Bidirectional Index [70] propone un método de string matching aproximado bidireccional. El hecho que sea aproximado puede producir errores por lo que se descarta su revisión.

6.2.2. Propuesta

En las secciones anteriores se describen una serie de modelos para determinar relaciones topológicas entre objetos espaciales mediante el uso de matrices de

intersección. A su vez se clasificaron las relaciones identificadas en los modelos de relaciones topológicas para líneas simples, en las 8 relaciones topológicas generales que resultan equivalentes al compararlas con la representación que se ejemplifica junto a su matriz de intersección.

En la Sección 2.2.1, se describen las dos formas generales de representación para trayectorias, las cuales corresponden a representaciones sobre espacio libre y a representaciones sobre una red. Para trayectorias que se representan sobre una red, es posible describir las trayectorias como la secuencia de nodos de la red por los que la trayectoria transita. Esta propuesta se enfoca en dicha representación para las trayectorias donde la secuencia de nodos está ordenada según el momento en el que la trayectoria coincide con el nodo.

Dado un conjunto de trayectorias simples representadas como una secuencia de nodos sobre una red es posible determinar cuál es la relación topológica que hay entre dos de las trayectorias del conjunto. Para esta propuesta se considera que una trayectoria $T = (p_0, t_0), (p_1, t_1), ..., (p_n, t_n)$ cumple que $t_i < t_{i+1}$ para $1 \leq i \leq n$ respecto a la dimensión temporal. Las ubicaciones espaciales se restringen sobre una red representada como un grafo por lo que se debe cumplir que dados p_i y p_{i+1} ambos corresponden a dos nodos y existe un arco en el grafo que conecta ambas ubicaciones. Por el momento esta propuesta contempla que dados dos pares de la misma trayectoria (p_i, t_i) y (p_j, t_j) se cumple que $p_i \neq p_j$ y $t_i \neq t_j$ para $i \neq j$.

Para la representación de la dimensión espacial como una secuencia de texto se tiene que el alfabeto Σ posee tantos elementos como nodos en el grafo, de manera que cada elemento del alfabeto representa un nodo del grafo. De esta forma, la secuencia S que representa la dimensión espacial de una trayectoria T corresponde a la concatenación de los elementos del alfabeto Σ que representan las ubicaciones espaciales de T. Es decir, que si $T = p_1, p_2, ..., p_n$ se tiene $S = s_1, s_2, ..., s_n$ donde s_i corresponde al elemento de Σ que representa el nodo p_i.

Esta propuesta contempla el desarrollo de un algoritmo que haga uso de estructuras de datos compactas para identificar las relaciones topológicas entre dos trayectorias representadas como una secuencia de nodos en una red, y la propuesta se evaluará comparando contra un algoritmo ingenuo que trabaje directamente sobre las secuencias para determinar su relación topológica.

En ambos casos, resulta de mucha utilidad para la implementación el establecer

cuál es el costo de determinar relaciones topológicas entre dos secuencias, para priorizar la verificación de aquellas menos costosas por sobre las otras. En esta primera aproximación, las coincidencias entre las secuencias se consideran sin hacer diferencia en el orden de los elementos. Es decir, que una secuencia formada por XYZ se considera igual a una secuencia ZYX ya que se asume que sigue el mismo camino sobre la red pero en sentido contrario. Sin embargo, una secuencia XZY es distinta de XYZ porque, a pesar de transitar por los mismos nodos, no son los mismos arcos que se recorren en la trayectoria representada. Esta consideración vale para la igualdad y la contención.

6.2.2.1. Procedimiento Ingenuo

El algoritmo que se propone para la versión ingenua determina la relación topológica por medio de las siguientes consideraciones:

- Equals: Las secuencias deben tener la misma longitud y coincidir ambas cadenas o con una de ellas en orden inverso. La igualdad se puede verificar en $\Theta(n)$ donde n corresponde a la longitud de la secuencia más corta de las dos.
- Contención (inside, includes, covers y coveredby): La contención de una secuencia en otra se puede verificar de manera sencilla mediante el algoritmo de Knuth-Morris-Patt [66] que determina si existe un patrón en un texto donde el costo en el peor caso es $\Theta(n+m)$, con n la longitud de la secuencia más corta y m la longitud de la secuencia más larga. Esta verificación se hace con la secuencia más corta como el patrón y se debe buscar también en orden inverso. La verificación omite la relación *Equals*.
- Overlaps: En este caso es necesario verificar la sub-secuencia común más larga que existe entre las dos secuencias que representan las trayectorias. La implementación con programación dinámica permite obtener el resultado en $\Theta(nm)$ tanto para la complejidad espacial como para la temporal, donde n corresponde a la longitud de la secuencia más corta y m es la longitud de la secuencia más larga. La verificación omite las relaciones del grupo *Contención*.
- Touches: Se puede verificar en tiempo lineal si se asume que no existen intersecciones del tipo interior-interior entre las trayectorias, vale decir que antes se ha verificado que la relación topológica entre las trayectorias no

es *Overlaps*, *Equals* ni ninguna de contención. Por tanto, sólo se necesita verificar que hay intersecciones de alguno de los bordes de cada secuencia contra uno de los caracteres de la otra secuencia.
- Disjoint: Se pueden descartar todas las demás relaciones y determinar que no existe ninguna relación topológica distinta.

Dado lo anterior, el costo de determinar la relación topológica que existe entre dos trayectorias representadas como una secuencia de nodos sobre una red tiene costo $\Theta(nm)$.

6.2.2.2. Procedimiento con Estructuras de Datos Compactas

El procedimiento utilizando estructuras de datos compactas sigue consideraciones equivalentes a las que se proponen en el procedimiento ingenuo haciendo uso de un *Árbol de Sufijos Generalizado* (`GST`).

Un `GST` es un *Árbol de Sufijos* (`ST`, ver en Sección 2.1.2) que se construye sobre la concatenación de más de una secuencia. El `ST` permite la búsqueda de patrones, sub-secuencias, sufijos y palíndromos, generalmente sobre texto. Su extensión a múltiples secuencias (`GST`) permite obtener elementos comunes en un conjunto de secuencias.

En el `GST` cada secuencia tiene un fin de línea propio, por lo que cada uno de sus sufijos terminan hacia un nodo hoja por medio de un arco asociado a dicho carácter. El procedimiento que se propone en esta sección hace uso de dos elementos adicionales al `GST`:

- Por cada secuencia de texto con el que se construyó el `GST` (que representan segmentos de línea) se guarda una referencia hacia el nodo que representa la secuencia completa desde el nodo raíz hasta la hoja con su fin de línea. Esto permite facilitar algunas consultas respecto del Lowest Common Ancestor entre dos secuencias dadas. Lo mismo se hace con el nodo hoja que representa la secuencia en orden inverso.
- Por cada nodo del `GST` se tienen marcas indicando qué secuencias tienen algún sufijo que visita dicho nodo. Con esto se pueden identificar coincidencias entre las secuencias en tiempo constante.

La Figura 6.2.2 muestra un ejemplo de un `GST` con los elementos adicionales que se han descrito. Para ejemplificar, se considera un grupo de 5 trayectorias

($T1, T2, T3, T4$ y $T5$) representadas en la Figura 6.2.1, en una red de 6 nodos (A, B, C, D, E y F). En el GST de la Figura 6.2.2 corresponde a la estructura propuesta en la que se insertan tanto las trayectorias como sus trayectorias reversas, etiquetadas con el nombre invertido de la secuencia que la origina. Las secuencias utilizadas para construir el árbol de la Figura 6.2.2 se especifican en la Tabla 6.2.2.

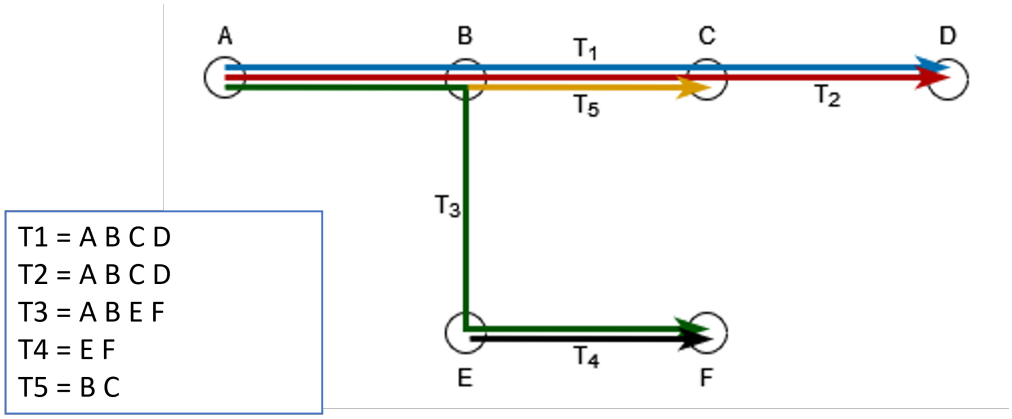


Figura 6.2.1: Ejemplo de trayectorias en una red.

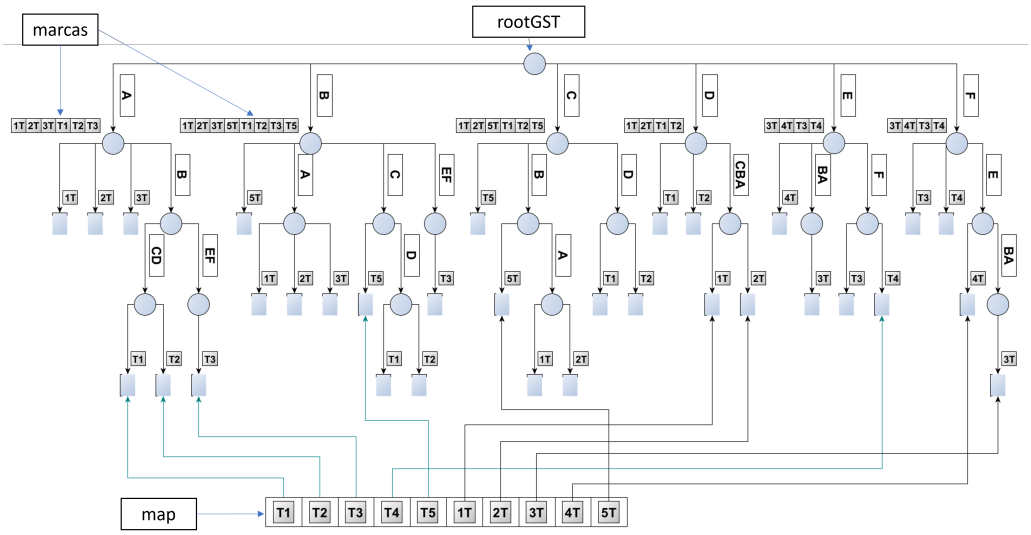


Figura 6.2.2: Ejemplo de GST propuesto.

Para determinar las relaciones topológicas entre dos trayectorias en la propuesta que utiliza el GST se utilizan criterios similares a los del algoritmo ingenuo, donde el descarte de una de las relaciones se pueden explorar otras. A continuación se describen las consideraciones utilizadas para determinar las relaciones topológicas y, en algunos casos, también se utilizan en las operaciones en las que se evalúa una relación topológica.

	Secuencia	Sec. Invertida	
$T1$	$ABCD$	$DCBA$	$1T$
$T2$	$ABCD$	$DCBA$	$2T$
$T3$	$ABEF$	$FEBA$	$3T$
$T4$	EF	FE	$4T$
$T5$	BC	CB	$5T$

Tabla 6.2.2: Secuencias usadas para construir `GST`.

- Equals: Si el padre del nodo hoja de cada secuencia que está referenciada por el *map* de la Figura 6.2.1, es el mismo nodo, entonces las secuencias son exactamente iguales. Esto se comprueba tanto para las dos secuencias originales, y también para una de ellas invertidas (en caso que las secuencias fueran iguales pero en sentido contrario).

  Ejemplo: para las secuencias $T1$ y $T2$, que poseen secuencias igual de largas, se verifican los nodos hojas de ambas secuencias en el *map* y se identifica que comparten padre, por tanto son iguales.
- Contención (inside, includes, covers y coveredby): Se explora el padre del nodo hoja de la secuencia más corta, donde se busca en las marcas una referencia a la otra secuencia (o a su inversa). Determinar cuál de las 4 relaciones básicas corresponde se puede verificar en tiempo constante identificando si hay intersección en los bordes y de cuál es la secuencia más larga/corta.

  Ejemplo: para las secuencias $T3$ y $T4$ tienen distinto secuencias de distinto largo, se mira en *map* la hoja de $T4$ (por ser la más corta). Luego se verifica si el padre posee una marca con $T3$ que en este caso existe, por tanto $T3$ contiene a $T4$. Dado que $T4$ es la secuencia más corta y que comparten el nodo F en sus bordes, se tiene que $T3$ *covers* $T4$, o lo que es equivalente $T4$ *coveredBy* $T3$.
- Overlaps: Por cada elemento de la secuencia más corta, se sigue el arco que corresponde desde el nodo raíz y se busca en el nodo hijo si existen marcas de la otra secuencia. Se debe comprobar que las coincidencias encontradas no corresponden al borde de alguna de las secuencias, caso que se considera *Touches*.

  Ejemplo: para las secuencias $T1$ y $T3$ no hay contención determinada en las hojas del `GST`, por lo que se escoge $T1$ (en este caso no hay secuencia más corta). Desde la raíz se va al nodo por el arco A que si tiene una *marca* de

$T3$ pero en ambos casos es un borde. Luego se va desde la raíz al arco B donde el nodo también tiene una *marca* de $T3$. Para el tercer elemento se va por el arco C donde ya no existe marca de $T3$, y lo mismo sucede en el arco F. Por tanto, dada que no todos los elementos de $T1$ intersecan con $T3$, y no todas las intersecciones implican bordes de alguna de las secuencias, se determina que existe *overlap* entre $T1$ y $T3$.

- Touches: Si todas las coincidencias encontradas al revisar *Overlaps* relacionan al menos un borde de las secuencias, y no existen otras coincidencias, se considera *Touches*.

  Ejemplo: para las secuencias $T3$ y $T5$ no hay contención determinada desde las hojas, por lo que se toma$T5$ por ser más corta y se recorre como sigue: El primero elemento de $T5$ sigue el arco B desde la raíz, donde el nodo si posee una marca de $T3$ y se tiene que dicha intersección involucra un elemento del borde de la secuencia $T5$. Luego, el siguiente elemento de $T5$ sigue el arco C donde el nodo correspondiente no posee marca de la secuencia $T3$. Dado que la única coincidencia que presentan las secuencias $T3$ y $T5$ corresponden a un borde, se determina que si hay *touches* entre ellas.
- Disjoint: Cuando ninguna de las otras relaciones existe. Es decir, se revisan todos los elementos de la secuencia más corta, desde el nodo raíz siguiendo el arco que corresponda, y no existen marcas asociadas a la otra secuencia en dicho nodo.

  Ejemplo: para las secuencias $T1$ y $T4$ no hay contención entre ellas, por lo que se verifica elemento a elemento desde la más corta que corresponde a $T4$. En los nodos que van desde la raíz a E y desde la raíz a F no hay coincidencia entre las secuencias, por lo tanto se determina que entre ellas la relación *disjoint* existe.

6.2.3. Experimentación, Resultados y Discusión

A continuación se describen aspectos de la implementación de los algoritmos propuestos y del ambiente de experimentación, junto con los resultados obtenidos.

6.2.3.1. Implementación

La implementación de ambos algoritmos se hizo en lenguaje de programación C++ y los códigos se encuentran disponibles en la plataforma GitHub[1]. El algoritmo ingenuo hace uso de algoritmos publicados en el sitio *GeeksForGeeks* con la implementación de Knuth-Norris-Pratt[2] y de LCS[3] (sub-secuencia común más larga). La propuesta con estructuras de datos compactas hace uso de la implementación del `ST` disponible en la librecía SDSL[4]. Esta librería tiene diferentes estructuras de datos compactas disponibles, entre las que se encuentra el Compressed Suffix Tree [117] y una mejora a dicha versión [98]. La implementación de esta última mejora está disponible en la librería en la clase cst_sct3[5].

6.2.3.2. Datos de prueba

Los datos de prueba corresponden a una porción de un dataset real con los viajes realizados por los servicios de transporte público de Santiago de Chile (TRANSANTIAGO) que están disponibles en el sitio de datos abiertos del gobierno bajo el título *Feed GTFS de Transantiago* [6] en formato GTFS. El dataset original se compone de un conjunto de archivos que tienen diferentes datos como: identificación de servicios, identificación de paradas y trips de los servicios entre otros.

El objetivo de esta propuesta es obtener relaciones topológicas entre trayectorias definidas en una red, entonces, el utilizar todos los servicios genera una repetición de muchas rutas. Por esto se decidió considerar el primer trip de cada servicio registrado en el conjunto mediante el procesamiento de los archivos *trips.txt* y *stops_ times.txt*. De esto se obtuvieron 8206 paradas y 393 trayectorias, las cuales se redujeron a 391 debido a que 2 de los servicios no cumplían la condición de ser representadas como segmentos de líneas simples (segmentos de línea continuos sin intersección entre su propio interior o bordes). La secuencia de mayor tamaño es de 138 elementos, la de menor tamaño de 2, y en promedio las secuencias tienen a 47 elementos.

[1] https://github.com/cquijadafuentes/codigos_propuesta_tesis
[2] https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/
[3] https://www.geeksforgeeks.org/longest-common-substring-dp-29
[4] https://github.com/simongog/sdsl-lite
[5] http://algo2.iti.kit.edu/gog/docs/html/group__cst.html
[6] https://datos.gob.cl/dataset/33245

6.2.3.3. Resultados Experimentales

Los resultados que a continuación se muestran corresponden a los tiempos de ejecución de los algoritmos medidos en una máquina con procesador Intel® Core™ i5-2430M de 2.40 GHz x 4 núcleos y 6 GB de memoria RAM. Los códigos fueron compilados en gcc versión 9.4 sobre el sistema operativo Ubuntu 20.04.4 de 64 bits. Los tiempos fueron tomados en el código con la librería *ctime* de C++ y sólo se considera el tiempo de ejecución de las consultas, no se incluyen tiempos de lectura de datos ni de salida de la información.

La Figura 6.2.3 muestra los tiempos promedio de la consulta booleana para cada implementación, es decir, que se responde verdadero o falso para cada una de las posibles relaciones topológicas dadas dos trayectorias. En este caso no se refleja el tiempo de construcción de la estructura en los resultados de la gráfica. Las consultas booleanas para cada relación topológica se probaron de dos formas distintas: consultar por cada relación posible todas las combinaciones posibles (*Naive All* y *GST All* en el gráfico) y consultar por cada relación posible sólo en aquellas combinaciones que se conoce su resultado como verdadero (*Naive True* y *GST True* en el gráfico).

CoveredBy	15	Includes	6
Covers	15	Inside	6
Disjoint	143.280	Overlaps	8.176
Equals	391	Touches	992

Tabla 6.2.3: Cantidad de relaciones topológicas entre las trayectorias.

Para determinar el tiempo promedio reportado en las consultas booleanas se consideró repetir 15 veces cada conjunto de las combinaciones que dan como resultado verdadero, para cada operación consultada. Esto debido a que existen algunas relaciones que poseen muy pocas combinaciones con valor verdadero. Los números de combinaciones para cada operación se detallan en la Tabla 6.2.3.

6.2.3.4. Discusión

A continuación se analizan los datos obtenidos como resultados de la experimentación entre ambas implementaciones.

Los resultados de la Figura 6.2.3 muestran los tiempos de las operaciones individualmente, donde por cada implementación se registran los tiempos

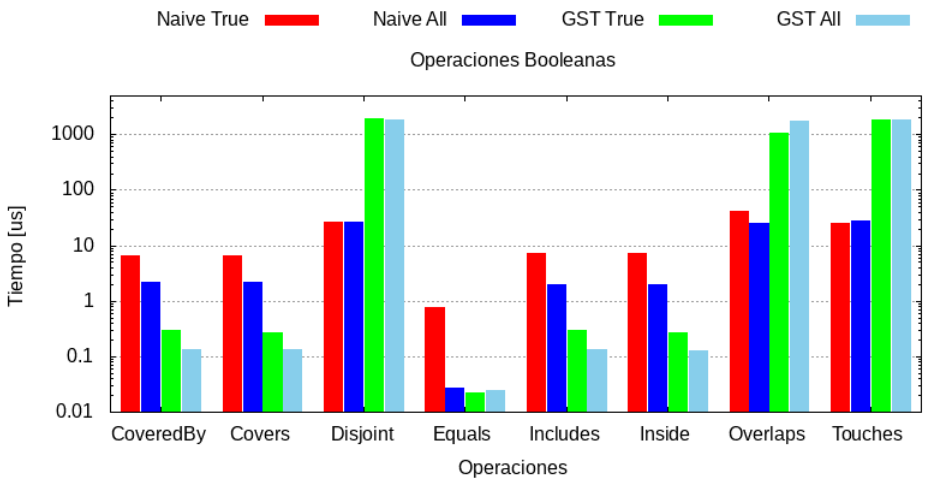


Figura 6.2.3: Tiempos para consulta booleana para cada relación.

promedios de revisar si existe o no la relación entre dos trayectorias para todas las combinaciones posibles (etiquetadas como *All* en la gráfica) o el promedio de aquellas combinaciones que resultan verdaderas (etiquetadas como *true* en la gráfica).

Las consultas de CoveredBy, Covers, Includes e Inside tienen un comportamiento similar en sus resultados, donde la implementación sobre el `GST` resulta más eficiente que la implementación ingenua, con alrededor de un orden de magnitud. En ambos casos el promedio es más bajo en las consultas generales comparadas con las consultas de resultado verdadero.

Las consultas Disjoint, Overlaps y Touches también presentan un comportamiento similar entre si, donde la implementación *Naive* es casi 2 órdenes de magnitud más rápida que la implementación sobre el `GST`. Entre consultas que siempre son verdaderas y todas las consultas hay leves diferencias que no destacan.

En la operacion Equals existe un comportamiento que llama la atención, ya que las consultas de la implementación `GST` y las consultas de todas las combinaciones presentan un promedio similar, mientras que las consultas que sólo son verdaderas en la implementación ingenua son por más de un orden de magnitud más lentas. Esto se puede explicar directamente por la implementación, ya que la versión `GST` es una consulta de orden constante, en cambio en la versión ingenua primero verifica si las longitudes de las secuencias son del mismo largo y luego hace una revisión lineal de los caracteres de la secuencia. Esta primera verificación sobre la longitud de las secuencias hace que el descarte por longitud reduzca el tiempo

promedio de realizar todas las consultas, en cambio el verificar igualdad siempre resulta más costoso.

En términos generales, las consultas se pueden agrupar por: igualdad, inclusión y otras. Igualdad es la operación menos costosa para todas las categorías. Luego la Inclusión que corresponde a CoveredBy, Covers, Includes e Inside tienen un tiempo similar entre sus correspondientes, y el comportamiento de las consultas es también parecido en términos generales. Finalmente, se tiene que las operaciones más costosas, tanto para la implementación ingenua como para la implementación con la estructura de datos compactas resultan ser Disjoint, Overlaps y Touches.

Bibliografía

[1] Mohammad Akbari, Ali Mousavi, Seyed Ahmad Eslaminezhad, Mobin Eftekhari, and Mohsen Ghorani. A new semantic trajectory ontology model for modelling and reasoning on movement data. In *29th Annual GIS Research UK Conference (GISRUK 2021)*, 04 2021.

[2] Luis Otavio Alvares, Vania Bogorny, Bart Kuijpers, Jose Antonio Fernandes de Macedo, Bart Moelans, and Alejandro Vaisman. A model for enriching trajectories with semantic geographical information. In *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*, pages 1–8, 2007.

[3] Maria Astefanoaei, Paul Cesaretti, Panagiota Katsikouli, Mayank Goswami, and Rik Sarkar. Multi-resolution sketches and locality sensitive hashing for fast trajectory processing. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 279–288, 2018.

[4] John Carter Bays. *The compleat PATRICIA*. PhD thesis, The University of Oklahoma., 1974.

[5] Bieganski, Riedl, Cartis, and Retzel. Generalized suffix trees for biological sequence data: applications and implementation. In *1994 Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*, volume 5, pages 35–44. IEEE, 1994.

[6] Vania Bogorny, Bart Kuijpers, and Luis Otavio Alvares. St-dmql: a semantic trajectory data mining query language. *International Journal of Geographical Information Science*, 23(10):1245–1276, 2009.

[7] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th international conference on World Wide Web*, pages 587–596, 2011.

[8] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[9] Sotiris Brakatsoulas, Dieter Pfoser, and Nectaria Tryfona. Modeling, storing and mining moving object databases. In *Proceedings. International Database*

Engineering and Applications Symposium, 2004. IDEAS'04., pages 68–77. IEEE, 2004.

[10] Loreto Bravo and M Andrea Rodriguez. Formalization and reasoning about spatial semantic integrity constraints. *Data & Knowledge Engineering*, 72:63–82, 2012.

[11] Nieves R Brisaboa, Rodrigo Cánovas, Francisco Claude, Miguel A Martínez-Prieto, and Gonzalo Navarro. Compressed string dictionaries. In *International Symposium on Experimental Algorithms*, pages 136–147. Springer, 2011.

[12] Nieves R Brisaboa, Guillermo De Bernardo, Gilberto Gutiérrez, Susana Ladra, Miguel R Penabad, and Brunny A Troncoso. Efficient set operations over k2-trees. In *2015 Data Compression Conference*, pages 373–382. IEEE, 2015.

[13] Nieves R Brisaboa, Antonio Fariña, Daniil Galaktionov, Tirso V Rodeiro, and M Andrea Rodriguez. Improved structures to solve aggregated queries for trips over public transportation networks. *Information Sciences*, 584:752–783, 2022.

[14] Nieves R Brisaboa, Antonio Fariña, Daniil Galaktionov, and M Andrea Rodriguez. A compact representation for trips over networks built on self-indexes. *Information Systems*, 78:1–22, 2018.

[15] Nieves R Brisaboa, Travis Gagie, Adrián Gómez-Brandón, Gonzalo Navarro, and José R Paramá. An index for moving objects with constant-time access to their compressed trajectories. *International Journal of Geographical Information Science*, 35(7):1392–1424, 2021.

[16] Nieves R Brisaboa, Adrián Gómez-Brandón, Gonzalo Navarro, and José R Paramá. Gract: a grammar-based compressed index for trajectory data. *Information Sciences*, 483:106–135, 2019.

[17] Nieves R Brisaboa, Susana Ladra, and Gonzalo Navarro. k2-trees for compact web graph representation. In *International symposium on string processing and information retrieval*, pages 18–30. Springer, 2009.

[18] Nieves R Brisaboa, Susana Ladra, and Gonzalo Navarro. Compact representation of web graphs with extended functionality. *Information Systems*, 39:152–174, 2014.

[19] Nieves Rodríguez Brisaboa, Antonio Fariña, Gonzalo Navarro, and Tirso Varela Rodeiro. Semantrix: A compressed semantic matrix. In *2020 Data Compression Conference (DCC)*, pages 113–122. IEEE, 2020.

[20] Daniela Campos, Adrián Gómez-Brandón, and Gonzalo Navarro. A disk-based index for trajectories with an in-memory compressed cache. In *2021 Data Compression Conference (DCC)*, pages 340–340. IEEE, 2021.

[21] Ana Cerdeira-Pena, Guillermo de Bernardo, Antonio Fariña, José Ramón Paramá, and Fernando Silva-Coira. Towards a compact representation of temporal rasters. In *International Symposium on String Processing and Information Retrieval*, pages 117–130. Springer, 2018.

[22] V Prasad Chakka, Adam Everspaugh, Jignesh M Patel, et al. Indexing large trajectory data sets with seti. In *CIDR*, volume 75, page 76. Citeseer, 2003.

[23] Ji-Dong Chen and Xiao-Feng Meng. Indexing future trajectories of moving objects in a constrained network. *Journal of Computer Science and Technology*, 22(2):245–251, 2007.

[24] Jidong Chen, Xiaofeng Meng, Yanyan Guo, Stéphane Grumbach, and Hui Sun. Modeling and predicting future trajectories of moving objects in a constrained network. In *7th International Conference on Mobile Data Management (MDM'06)*, pages 156–156. IEEE, 2006.

[25] Ahlame Douzal Chouakria and Panduranga Naidu Nagabhushan. Adaptive dissimilarity index for measuring time series proximity. *Advances in Data Analysis and Classification*, 1:5–21, 2007.

[26] David Clark. *Compact pat trees*. PhD thesis, University of Waterloo Canada, 1996.

[27] Eliseo Clementini, Paolino Di Felice, and Peter Van Oosterom. A small set of formal topological relationships suitable for end-user interaction. In *International symposium on spatial databases*, pages 277–295. Springer, 1993.

[28] Israel Cohen, Yiteng Huang, Jingdong Chen, Jacob Benesty, Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. *Noise reduction in speech processing*, pages 1–4, 2009.

[29] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[30] Philippe Cudre-Mauroux, Eugene Wu, and Samuel Madden. Trajstore: An adaptive storage system for very large trajectory data sets. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 109–120. IEEE, 2010.

[31] Guillermo De Bernardo, Sandra Álvarez-García, Nieves R Brisaboa, Gonzalo Navarro, and Oscar Pedreira. Compact querieable representations of raster data. In *String Processing and Information Retrieval: 20th International Symposium, SPIRE 2013, Jerusalem, Israel, October 7-9, 2013, Proceedings 20*, pages 96–108. Springer, 2013.

[32] Shuhei Denzumi, Jun Kawahara, Koji Tsuda, Hiroki Arimura, Shin-ichi Minato, and Kunihiko Sadakane. Densezdd: a compact and fast index for families of sets. In *Experimental Algorithms: 13th International Symposium, SEA 2014, Copenhagen, Denmark, June 29–July 1, 2014. Proceedings 13*, pages 187–198. Springer, 2014.

[33] Timothy R Derrick, Barry T Bates, and Janet S Dufek. Evaluation of time-series data sets using the pearson product-moment correlation coefficient. *Medicine and science in sports and exercise*, 26(7):919–928, 1994.

[34] Zhiming Ding. Utr-tree: An index structure for the full uncertain trajectories of network-constrained moving objects. In *The Ninth International Conference on Mobile Data Management (mdm 2008)*, pages 33–40. IEEE, 2008.

[35] Cedric Du Mouza and Philippe Rigaux. Multi-scale classification of moving objects trajectories. In *SSDBM*, pages 307–316. Citeseer, 2004.

[36] Jean Dubé and Dièg0 Legros. A spatio-temporal measure of spatial dependence: An example using real estate data. *Papers in Regional Science*, 92(1):19–31, 2013.

[37] Patrick Eades, Ivor van der Hoog, Maarten Löffler, and Frank Staals. Trajectory visibility. In *17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[38] M Egenhofer and Jayant Sharma. Assessing the consistency of complete and incomplete topological information. *Geographical Systems*, 1(1):47–68, 1993.

[39] Max J Egenhofer. A formal definition of binary topological relationships. In *Foundations of Data Organization and Algorithms: 3rd International Conference, FODO 1989 Paris, France, June 21–23, 1989 Proceedings 3*, pages 457–472. Springer, 1989.

[40] Max J Egenhofer. Reasoning about binary topological relations. In *Symposium on Spatial Databases*, pages 141–160. Springer, 1991.

[41] Max J. Egenhofer. Spatial sql: A query and presentation language. *IEEE Transactions on knowledge and data engineering*, 6(1):86–95, 1994.

[42] Max J Egenhofer and Andrew U Frank. Towards a spatial query language: User interface considerations. In *VLDB*, volume 88, pages 124–133, 1988.

[43] Max J Egenhofer and Robert D Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information System*, 5(2):161–174, 1991.

[44] Max J Egenhofer and Robert D Franzosa. On the equivalence of topological relations. *International journal of geographical information systems*, 9(2):133–152, 1995.

[45] Max J Egenhofer and John Herring. Categorizing binary topological relations between regions, lines, and points in geographic databases. *The*, 9(94-1):76, 1990.

[46] Max J Egenhofer and Jayant Sharma. Topological relations between regions in r^ 2 and z^ 2. *Lecture Notes in Computer Science*, pages 316–316, 1993.

[47] Peter Elias. Efficient storage and retrieval by content and address of static files. *Journal of the ACM (JACM)*, 21(2):246–260, 1974.

[48] Antonio Fariña, Nieves R. Brisaboa, Gonzalo Navarro, Francisco Claude, Ángeles Saavedra Places, and Eduardo Rodríguez. Word-based self-indexes for natural language text. *ACM Trans. Inf. Syst.*, 30(1):1:1–1:34, 2012.

[49] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings 41st annual symposium on foundations of computer science*, pages 390–398. IEEE, 2000.

[50] Yong Gao, Jing Cheng, Haohan Meng, and Yu Liu. Measuring spatio-temporal autocorrelation in time series data of collective human mobility. *Geo-spatial Information Science*, 22(3):166–173, 2019.

[51] 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.

[52] Leticia Gómez, Bart Kuijpers, and Alejandro Vaisman. Querying and mining trajectory databases using places of interest. In *New trends in data warehousing and data analysis*, pages 1–26. Springer, 2009.

[53] Inc Google. Protocol buffers version 3 language specification. https://developers.google.com/protocol-buffers/docs/reference/proto3-spec, 2020.

[54] Peter D Grünwald, In Jae Myung, and Mark A Pitt. *Advances in minimum description length: Theory and applications*. MIT press, 2005.

[55] Dan Gusfield. Algorithms on stings, trees, and sequences: Computer science and computational biology. *Acm Sigact News*, 28(4):41–60, 1997.

[56] Ralf Hartmut Güting, Michael H Böhlen, Martin Erwig, Christian S Jensen, Nikos A Lorentzos, Markus Schneider, and Michalis Vazirgiannis. A foundation for representing and querying moving objects. *ACM Transactions on Database Systems (TODS)*, 25(1):1–42, 2000.

[57] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, 1984.

[58] Yunheng Han, Weiwei Sun, and Baihua Zheng. Compress: A comprehensive framework of trajectory compression in road networks. *ACM Transactions on Database Systems (TODS)*, 42(2):1–49, 2017.

[59] John Herring et al. *Opengis® implementation standard for geographic information-simple feature access-part 1: Common architecture [corrigendum]*. Open Geospatial Consortium, 2011.

[60] Te C Hu and Alan C Tucker. Optimal computer search trees and variable-length alphabetical codes. *SIAM Journal on Applied Mathematics*, 21(4):514–532, 1971.

[61] Ilya Ilyankou. Comparison of jaro-winkler and ratcliff/obershelp algorithms in spell check. *IB Extended Essay Computer Science*, 1(2):3, 2014.

[62] Hoyoung Jeung, Qing Liu, Heng Tao Shen, and Xiaofang Zhou. A hybrid prediction model for moving objects. In *2008 IEEE 24th international conference on data engineering*, pages 70–79. IEEE, 2008.

[63] Shunsuke Kanda, Koh Takeuchi, Keisuke Fujii, and Yasuo Tabei. Succinct trit-array trie for scalable trajectory similarity search. In *Proceedings of the 28th International Conference on Advances in Geographic Information Systems*, pages 518–529, 2020.

[64] Georgios Kellaris, Nikos Pelekis, and Yannis Theodoridis. Trajectory compression under network constraints. In *International Symposium on Spatial and Temporal Databases*, pages 392–398. Springer, 2009.

[65] Ahmed Kharrat, Iulian Sandu Popa, Karine Zeitouni, and Sami Faiz. Clustering algorithm for network constraint trajectories. In *Headway in Spatial Data Handling*, pages 631–647. Springer, 2008.

[66] Donald E Knuth, James H Morris, Jr, and Vaughan R Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.

[67] Satoshi Koide, Yukihiro Tadokoro, Chuan Xiao, and Yoshiharu Ishikawa. Cinct: Compression and retrieval for massive vehicular trajectories via relative movement labeling. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1097–1108. IEEE, 2018.

[68] Satoshi Koide, Yukihiro Tadokoro, and Takayoshi Yoshimura. Snt-index: Spatio-temporal index for vehicular trajectories on a road network based on substring matching. In *Proceedings of the 1st International ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*, pages 1–8, 2015.

[69] Benjamin Krogh, Christian S Jensen, and Kristian Torp. Efficient in-memory indexing of network-constrained trajectories. In *Proceedings of the 24th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 1–10, 2016.

[70] Gregory Kucherov, Kamil Salikhov, and Dekel Tsur. Approximate string matching using a bidirectional index. In *Symposium on Combinatorial Pattern Matching*, pages 222–231. Springer, 2014.

[71] Bart Kuijpers and Walied Othman. Trajectory databases: Data models, uncertainty and complete query languages. *Journal of Computer and System Sciences*, 76(7):538–560, 2010.

[72] Yohei Kurata and Max J Egenhofer. The head-body-tail intersection for spatial relations between directed line segments. In *International Conference on Geographic Information Science*, pages 269–286. Springer, 2006.

[73] Albert K Kurtz, Samuel T Mayo, Albert K Kurtz, and Samuel T Mayo. Pearson product moment coefficient of correlation. *Statistical Methods in Education and Psychology*, pages 192–277, 1979.

[74] Susana Ladra, José R Paramá, and Fernado Silva-Coira. Compact and queryable representation of raster datasets. In *Proceedings of the 28th International Conference on Scientific and Statistical Database Management*, pages 1–12, 2016.

[75] Susana Ladra, José R Paramá, and Fernando Silva-Coira. Scalable and queryable compressed storage structure for raster data. *Information Systems*, 72:179–204, 2017.

[76] N Jesper Larsson and Alistair Moffat. Off-line dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732, 2000.

[77] Jay Lee and Shengwen Li. Extending moran's index for measuring spatiotemporal clustering of geographic events. *Geographical Analysis*, 49(1):36–57, 2017.

[78] Joseph Lee Rodgers and W Alan Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1):59–66, 1988.

[79] Pierre Legendre. Spatial autocorrelation: trouble or new paradigm? *Ecology*, 74(6):1659–1673, 1993.

[80] Maria Liatsikou, Symeon Papadopoulos, Lazaros Apostolidis, and Ioannis Kompatsiaris. A denoising hybrid model for anomaly detection in trajectory sequences. In *EDBT/ICDT Workshops*, 2021.

[81] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing sax: a novel symbolic representation of time series. *Data Mining and knowledge discovery*, 15(2):107–144, 2007.

[82] Jessica Lin and Yuan Li. Finding structural similarity in time series data using bag-of-patterns representation. In *International conference on scientific and statistical database management*, pages 461–477. Springer, 2009.

[83] Tsong-Wuu Lin. Set operations on constant bit-length linear quadtrees. *Pattern Recognition*, 30(7):1239–1249, 1997.

[84] Nikos Mamoulis, David W Cheung, and Wang Lian. Similarity search in sets and categorical data using the signature tree. In *Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405)*, pages 75–86. IEEE, 2003.

[85] Vasileios Megalooikonomou, Qiang Wang, Guo Li, and Christos Faloutsos. A multiresolution symbolic representation of time series. In *21st International Conference on Data Engineering (ICDE'05)*, pages 668–679. IEEE, 2005.

[86] Shin-ichi Minato and Hiroki Arimura. Efficient method of combinatorial item set analysis based on zero-suppressed bdds. In *International Workshop*

on Challenges in Web Information Retrieval and Integration, pages 4–11. IEEE, 2005.

[87] Fabio Miranda, Marcos Lage, Harish Doraiswamy, Charlie Mydlarz, Justin Salamon, Yitzchak Lockerman, Juliana Freire, and Claudio T Silva. Time lattice: A data structure for the interactive visual analysis of large time series. In *Computer Graphics Forum*, volume 37, pages 23–35. Wiley Online Library, 2018.

[88] Hoda Mokhtar and Jianwen Su. A query language for moving object trajectories. In *SSDBM*, pages 173–182. Citeseer, 2005.

[89] Patrick AP Moran. Notes on continuous stochastic phenomena. *Biometrika*, 37(1/2):17–23, 1950.

[90] Mikołaj Morzy. Mining frequent trajectories of moving objects for location prediction. In *International workshop on machine learning and data mining in pattern recognition*, pages 667–680. Springer, 2007.

[91] Jonathan Muckell, Jeong-Hyon Hwang, Vikram Patil, Catherine T Lawson, Fan Ping, and SS Ravi. Squish: an online approach for gps trajectory compression. In *Proceedings of the 2nd international conference on computing for geospatial research & applications*, pages 1–8, 2011.

[92] Joong Chae Na, Heejin Park, Maxime Crochemore, Jan Holub, Costas S Iliopoulos, Laurent Mouchard, and Kunsoo Park. Suffix tree of alignment: An efficient index for similar data. In *Combinatorial Algorithms: 24th International Workshop, IWOCA 2013, Rouen, France, July 10-12, 2013, Revised Selected Papers 24*, pages 337–348. Springer, 2013.

[93] Mirco Nanni and Dino Pedreschi. Time-focused clustering of trajectories of moving objects. *Journal of Intelligent Information Systems*, 27(3):267–289, 2006.

[94] Gonzalo Navarro. Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. *ACM Computing Surveys (CSUR)*, 46(4):1–47, 2014.

[95] Gonzalo Navarro. *Compact data structures: A practical approach*. Cambridge University Press, 2016.

[96] Jinfeng Ni and Chinya V Ravishankar. Indexing spatio-temporal trajectories with efficient polynomial approximations. *IEEE Transactions on Knowledge and Data Engineering*, 19(5):663–678, 2007.

[97] Aiden Nibali and Zhen He. Trajic: An effective compression system for trajectory data. *IEEE Transactions on Knowledge and Data Engineering*, 27(11):3138–3151, 2015.

[98] Enno Ohlebusch, Johannes Fischer, and Simon Gog. Cst++. In *International*

symposium on string processing and information retrieval, pages 322–333. Springer, 2010.

[99] Daisuke Okanohara and Kunihiko Sadakane. Practical entropy-compressed rank/select dictionary. In *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 60–70. SIAM, 2007.

[100] David O'sullivan and David Unwin. *Geographic information analysis*. John Wiley & Sons, 2003.

[101] Jan Paredaens and Bart Kuijpers. Data models and query languages for spatial databases. *Data & Knowledge Engineering*, 25(1-2):29–53, 1998.

[102] Christine Parent, Stefano Spaccapietra, Chiara Renso, Gennady Andrienko, Natalia Andrienko, Vania Bogorny, Maria Luisa Damiani, Aris Gkoulalas-Divanis, Jose Macedo, Nikos Pelekis, et al. Semantic trajectories modeling and analysis. *ACM Computing Surveys (CSUR)*, 45(4):1–32, 2013.

[103] Donna J Peuquet. A conceptual framework and comparison of spatial data models. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 21(4):66–113, 1984.

[104] Dieter Pfoser and Christian S Jensen. Indexing of network constrained moving objects. In *Proceedings of the 11th ACM international symposium on Advances in geographic information systems*, pages 25–32, 2003.

[105] Dieter Pfoser, Christian S Jensen, Yannis Theodoridis, et al. Novel approaches to the indexing of moving object trajectories. In *VLDB*, volume 2000, pages 395–406. Citeseer, 2000.

[106] Alejandro Pinto, Diego Seco, and Gilberto Gutiérrez. Improved queryable representations of rasters. In *2017 Data Compression Conference (DCC)*, pages 320–329. IEEE, 2017.

[107] Ruzica Piskac, Leonardo de Moura, and Nikolaj Bjørner. Deciding effectively propositional logic using dpll and substitution sets. *Journal of Automated Reasoning*, 44:401–424, 2010.

[108] OGC Query. Language for rdf data. *Open Geopatial Consortium*, 2012.

[109] Carlos Quijada-Fuentes, M Andrea Rodríguez, and Diego Seco. Trgst: An enhanced generalized suffix tree for topological relations between paths. *Information Systems*, 125:102406, 2024.

[110] Carlos Quijada-Fuentes, M Andrea Rodríguez, and Diego Seco. Formalization and scalable processing of spatially-embedded time series. *Geo-spatial Information Science*, 2025. under submission.

[111] Vincent Rabaud and Serge Belongie. Counting crowded moving objects. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 705–711. IEEE, 2006.

[112] David A Randell, Zhan Cui, and Anthony G Cohn. A spatial logic based on regions and connection. *KR*, 92:165–176, 1992.

[113] Roonak Rezvani, Payam Barnaghi, and Shirin Enshaeifar. A new pattern representation method for time-series data. *IEEE Transactions on Knowledge and Data Engineering*, 33(7):2818–2832, 2019.

[114] Konrad Rieck, Pavel Laskov, and Sören Sonnenburg. Computation of similarity measures for sequential data using generalized suffix trees. *Advances in neural information processing systems*, 19, 2006.

[115] Valery I Rupasov, Mikhail A Lebedev, Joseph S Erlichman, Stephen L Lee, James C Leiter, and Michael Linderman. Time-dependent statistical and correlation properties of neural signals during handwriting. *PloS one*, 7, 09 2012.

[116] Yukio Sadahiro, Raymond Lay, and Tetsuo Kobayashi. Trajectories of moving objects on a network: detection of similarities, visualization of relations, and classification of trajectories. *Transactions in GIS*, 17(1):18–40, 2013.

[117] Kunihiko Sadakane. Compressed suffix trees with full functionality. *Theory of Computing Systems*, 41(4):589–607, 2007.

[118] JL Schafer. Multiple imputation: a primer. stat methods in med., 8 (1), 3–15. *Scientific Data Management, 1, 32*, 38, 1999.

[119] Jean-Guy Schneider, Peter Mandile, and Steve Versteeg. Generalized suffix tree based multiple sequence alignment for service virtualization. In *2015 24th Australasian Software Engineering Conference*, pages 48–57. IEEE, 2015.

[120] Markus Schneider and Thomas Behr. Topological relationships between complex spatial objects. *ACM Transactions on Database Systems (TODS)*, 31(1):39–81, 2006.

[121] Claude E Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[122] Jingwei Shen, Kaifang Shi, and Min Chen. Topological relations between directed line segments in the cyclic space. *Journal of Geographical Systems*, 22(4):497–518, 2020.

[123] Jingwei Shen, Dongzhe Zhao, Kaifang Shi, and Mingguo Ma. A model for representing topological relations between lines considering metric details. *Journal of Geographical Systems*, pages 1–18, 2021.

[124] Fernando Silva-Coira, José R Paramá, Guillermo de Bernardo, and Diego Seco. Space-efficient representations of raster time series. *Information Sciences*, 566:300–325, 2021.

[125] Stefano Spaccapietra, Christine Parent, Maria Luisa Damiani, Jose Antonio de Macedo, Fabio Porto, and Christelle Vangenot. A conceptual view on trajectories. *Data & knowledge engineering*, 65(1):126–146, 2008.

[126] Yufei Tao and Dimitris Papadias. The mv3r-tree: A spatio-temporal access method for timestamp and interval queries. In *Proceedings of Very Large Data Bases Conference (VLDB), 11-14 September, Rome*, 2001.

[127] Waldo R Tobler. A computer movie simulating urban growth in the detroit region. *Economic geography*, 46(sup1):234–240, 1970.

[128] Goce Trajcevski, Ouri Wolfson, Fengli Zhang, and Sam Chamberlain. The geometry of uncertainty in moving objects databases. In *International Conference on Extending Database Technology*, pages 233–250. Springer, 2002.

[129] Nico Van de Weghe. *Representing and reasoning about moving objects: A qualitative approach*. PhD thesis, Ghent University, 2004.

[130] Nico Van de Weghe, Bart Kuijpers, Peter Bogaert, and Philippe De Maeyer. A qualitative trajectory calculus and the composition of its relations. In *International Conference on GeoSpatial Sematics*, pages 60–76. Springer, 2005.

[131] Bálint Márk Vásárhelyi. *Suffix trees and their applications*. PhD thesis, MSc Thesis, Faculty of Science, Eötvös Loránd University, Budapest, 2013.

[132] Michalis Vazirgiannis and Ouri Wolfson. A spatiotemporal model and language for moving objects on road networks. In *International Symposium on Spatial and Temporal Databases*, pages 20–35. Springer, 2001.

[133] Laure Vieu. Spatial representation and reasoning in artificial intelligence. In *Spatial and temporal reasoning*, pages 5–41. Springer, 1997.

[134] Sebastiano Vigna. Broadword implementation of rank/select queries. In *International Workshop on Experimental and Efficient Algorithms*, pages 154–168. Springer, 2008.

[135] Xiaogang Wang, Kinh Tieu, and Eric Grimson. Learning semantic scene models by trajectory analysis. In *European conference on computer vision*, pages 110–123. Springer, 2006.

[136] Niklaus Wirth. *Algorithms & data structures*. Prentice-Hall, Inc., 1985.

[137] Youlong Xia, Kenneth Mitchell, Michael Ek, Justin Sheffield, Brian Cosgrove, Eric Wood, Lifeng Luo, Charles Alonge, Helin Wei, Jesse Meng, et al. Continental-scale water and energy flux analysis and validation for the north american land data assimilation system project phase 2 (nldas-2): 1. intercomparison and application of model products. *Journal of Geophysical Research: Atmospheres*, 117(D3), 2012.

[138] Lin Xu, Yang Yue, and Qingquan Li. Identifying urban traffic congestion pattern from historical floating car data. *Procedia-Social and Behavioral Sciences*, 96:2084–2095, 2013.

[139] Shengxun Yang, Zhen He, and Yi-Ping Phoebe Chen. Gcotraj: A storage approach for historical trajectory data sets using grid cells ordering. *Information Sciences*, 459:1–19, 2018.

[140] Liang Zuopeng, Hu Kongfa, Ye Ning, and Dong Yisheng. An efficient index structure for xml based on generalized suffix tree. *Information Systems*, 32(2):283–294, 2007.