Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Uso de t-Spanners para Búsqueda en Espacios Métricos

Rodrigo Andrés Paredes Moraleda

2002

Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Uso de t-Spanners para Búsqueda en Espacios Métricos

Rodrigo Andrés Paredes Moraleda

Comision examinadora		Nota (n°)	Calificaciones (letras)	Firma
Profesor Guía Sr. Gonzalo Navarro	:	.....	.....	.....
Profesor Integrante Sr. Edgar Chávez González	:	.....	.....	.....
Profesor Integrante Sra. María Cecilia Rivara	:	.....	.....	.....
Profesor Invitado Sr. Luis Dissett Vélez	:	.....	.....	.....
Nota Final Examen de Grado	:	.....	.....	.....

Tesis para optar al
Título de Ingeniero Civil en Computación y
Grado de Magíster en Ciencias, Mención Computación

Septiembre, 2002

RESUMEN DE LA TESIS PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN Y GRADO DE MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN
POR : RODRIGO PAREDES MORALEDA
FECHA : 05 DE SEPTIEMBRE DE 2002
PROF. GUÍA : PhD GONZALO NAVARRO

Uso de t-Spanners para Búsqueda en Espacios Métricos

Dada una colección de n objetos, el problema de buscar los elementos más cercanos, bajo algún criterio de similitud, a una consulta dada, tiene un gran número de aplicaciones en muchas áreas de la computación, que van desde el reconocimiento de patrones hasta la recuperación de información en bases de datos textuales o multimediales. Estas aplicaciones tienen en común que la colección de objetos junto al criterio de similitud ("distancia" entre los objetos), conforman un *Espacio Métrico* (EM).

Entre los algoritmos para buscar objetos en un EM, AESA (Approximating Eliminating Search Algorithm) es uno de los que presenta mejor desempeño. El problema de AESA es que requiere precalcular y almacenar las $n(n-1)/2$ distancias entre pares de objetos, necesitando $O(n^2)$ memoria, lo que lo hace impracticable. Sin embargo, si fuese posible disminuir el uso de memoria de AESA, éste se podría utilizar en aplicaciones reales. Con esta intención, se considera la alternativa de utilizar *t-spanners* para representar la estructura de distancias entre objetos del EM, conformando de este modo la base de datos métrica.

Durante la tesis se desarrollaron cinco algoritmos de construcción de t-spanners, un algoritmo de inserción y otro de borrado de objetos en el t-spanner, un algoritmo reconstrucción de t-spanners y un algoritmo para resolver consultas de recuperación de objetos. Adicionalmente se realizaron experimentos de construcción de t-spanners en EMs de interés, vale decir, espacios vectoriales con la distancia euclidiana, espacios de strings con distancia de edición y espacios de documentos con la distancia coseno. También se realizaron experimentos de búsqueda en dichos EMs, comparando el método basado en t-spanners contra AESA y contra un algoritmo basado en pivotes.

En EMs construidos con objetos del mundo real, se muestra que el método basado en t-spanners presenta resultados mucho mejores que los obtenidos con algoritmos basados en pivotes, y por otro lado, que es altamente competitivo con respecto a AESA. Por ejemplo, en el espacio de los documentos, utilizando el método basado en t-spanners, se requiere de un 3.84 % de la memoria que utiliza AESA y se paga un exceso de sólo un 9 % más de las evaluaciones de distancia que utiliza AESA para recuperar objetos. Se conjetura y demuestra experimentalmente que la propiedad del espacio métrico que resulta esencial para el buen comportamiento de los t-spanners, es la existencia de *clusters* de elementos. Esta propiedad se presenta en la mayoría de los espacios de la vida real, por lo que se espera que los t-spanners se comporten bien en la práctica.

Además de los resultados concretos de esta tesis, se abre la puerta a varios temas futuros de investigación. Uno es el desarrollo de algoritmos de construcción de t-spanners que consideren valores de t dependientes de la distancia entre los nodos, de manera que la aproximación sea mejor para la vecindad de los nodos que para los nodos alejados. Otro es utilizar el t-spanner como una estructura de navegación sobre el EM. De esta manera se podría considerar la búsqueda como la combinación de una fase de acercamiento hacia la consulta, y luego una fase de eliminación de candidatos.

a mi padre,
quien me enseñó a vivir
haciendo sólo lo que a uno le gusta

mi papá era un gran artesano

Agradecimientos

Para ahorrarme trabajo, sólo pensaba agradecer a aquellas personas que me ayudaron en estos últimos dos años, luego me encontré con la sorpresa que no sólo eran los mismos de hace dos años atrás, eran muchas más personas. Así que ... empecemos por el comienzo:

En primer lugar le agradezco a mi madre, Isa, la mujer más valiente que he conocido. Tu incondicinal apoyo ha sido muy importante para que tu guagua (para las mamás siempre somos sus guaguas :)) cumpla con sus metas, a tí van mis mayores agradecimientos.

También van los agradecimientos a mi padre, César, que sin entender lo que yo hacía, me apoyaba e impulsaba a ser mejor. Además, gracias a que él me mostró el camino amarillo de hacer lo que a uno le gusta, es que ahora estoy feliz, dedicado de lleno a la Universidad.

A mis hermanos Waldo y César, que siempre se han preocupado de mí, y me apoyan en todo lo que esté a su alcance. No puedo dejar pasar a mi cuñada, la Pacita y a mis amados sobrinos, Cesítar y Valentina.

Al resto de mi familia también les agradezco, en especial a Lucrecia, Pablo, Pola, Jorge y Sara.

A Paula Miño, mi polola, por su abnegación, paciencia y gran amor hacia mí.

Uhhh, cuando me acuerdo de la cantidad de amigos y amigas que he tenido también veo la lista larga:

A Jorge León, y Juan Manuel Vargas, y sus familias y su eterna amistad de 15 años, ha pasado tanto tiempo.

A Estrella Callejas, mi amiga, por su compañía, su tiempo y su forma de ver las cosas.

A mis amigos del preu JCT, en especial a LAF. No puedo dejar de nombrar a Carlos Lillo y Ana Cáceres, amigos de toda la vida y en cualquier situación, a Paola Torres y Javier Aguilera, a Luis Carrasco, a Teresa Villarroel, a Paulina Contreras, a Mauricio Arriaza, a Claudio Araya, a Guillermo Olivares, y tanto otros que no recuerdo en mi memoria, pero sí en mi corazón.

A mis compañeros de escuela, y actualmente colegas Eléctricos, Juan Pérez, Eduardo Morales y Fernando Flattow, mucho éxito en sus vidas profesionales y personales también.

A mis colegas de AT&T LA, en especial a Alfonso Ehijo, quien fue mi profesor guía en electricidad, gracias por tu incansable apoyo y por ser mi mentor, él me acogió en la empresa para que pueda seguir mis estudios en computación, y el magíster, y después aceptó que siga estudiando el doctorado. A Juan Ocampos, mi jefe, por su tiempo, consejos y trucos

en el trabajo. A mis compañeros Arturo Nera, Esteban Troncoso, Luis Palma y José Magni, por su apoyo incondicional y por levantarme el ánimo cuando tuve dudas en mi vocación académica. Al resto del equipo de ingeniería, Juan Loza, Jorge Sepúlveda, Jorge Rodriguez, José Alvarado, Pablo Burgos, Carlos Góndola, Andrés Sandoval, Ricardo Pedraza Luis Mansilla, Simón Mancisidor, Oscar Muñoz y Claudio Gutiérrez, que me aceptaron tal como soy. Por último a Carlos Fernández y a la compañía completa, que me brindaron una cálida bienvenida, un grato lugar laboral y la oportunidad de hacer bien mi trabajo.

A mis profesoras Carmen Ortiz y Nancy Lacourly.

A la gente del DCC, en especial a Angélica Aguirre, gracias, muchas gracias, y perdona ser tan molestoso. A Magaly, a Cuauhtémoc, a Benjamín, al resto de compañeros de la familia de post-grado y a los profesores que me aguantaron en sus cursos.

A los profesores de mi comisión, Edgar Chávez, Luis Dissett y María Cecila Rivara. Edgar, nos vemos en México.

Por último, pero no menos importante, a Gonzalo Navarro, mi profesor guía en computación. Muchas gracias por tu paciencia, apoyo, exigencia, rigurosidad, confianza y tiempo.

A todos Ustedes, muchas gracias.

Muchos han hecho notar la rapidez con que Muad'Dib aprendió las necesidades de Arrakis. Las Bene Gesserit, por supuesto, conocen los fundamentos de esta rapidez. Para los demás, diremos que Muad'Dib aprendió rápidamente porque la primera enseñanza que recibió fue la certeza básica de que podía aprender. Es horrible pensar cómo tanta gente cree que no puede aprender, y cómo más gente aún cree que el aprender es difícil. Muad'Dib sabía que cada experiencia lleva en sí misma su lección.

De *La humanidad de Muad'Dib*, por la Princesa Irulan.
Dune, Frank Herbert

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Marco General

Buscar es un problema fundamental en ciencias de la computación, que está presente en la mayoría de las aplicaciones. La noción tradicional, "búsqueda exacta", consiste en encontrar un elemento cuyo identificador corresponda exactamente a una llave de búsqueda definida. La extensión natural de esta noción es la "búsqueda en proximidad", que pretende encontrar un elemento de la base de datos (BD) suficientemente cercano a dicha llave de búsqueda.

Las BD tradicionales se diseñan con el objetivo de resolver búsquedas exactas eficientemente realizando comparaciones entre elementos simples. Los nuevos desarrollos tecnológicos han originado BD que contienen información no estructurada, como lo son imágenes, sonidos, texto libre, etc., en donde no es posible definir claramente un identificador para cada registro de la BD. Incluso, si fuese posible definir el identificador, al momento de realizar la búsqueda puede ser necesario comparar la llave con todos los campos del registro. Por otro lado, en ciertas ocasiones se desea consultar por un elemento que no pertenece a la BD, en cuyo caso es necesario recuperar algún(algunos) elemento(s) que sea(n) similar(es) a la llave de búsqueda.

Existen variadas aplicaciones prácticas de las herramientas de búsqueda en proximidad, entre ellas: bases de datos multimediales, clasificación, compresión y transmisión de video, recuperación de textos, biología computacional, reconocimiento de patrones, etc.

Un ejemplo del uso de búsqueda en proximidad corresponde al proceso de autenticación biométrica. En este caso el sujeto se presenta al dispositivo de autenticación y éste debe determinar si efectivamente el sujeto es quien dice ser, considerando que la muestra que obtiene el sensor en general no corresponde a ningún elemento almacenado en la BD puesto que la medición está sujeta a efectos de rotación y escalamiento, variaciones en la intensidad promedio, defectos debidos al uso del dispositivo sensor, variaciones propias de todo organismo vivo, etc.

Estas aplicaciones tienen en común que los elementos pertenecientes a la BD conforman un *Espacio Métrico* [CNBYM01]. Esto significa que es posible definir una función real no negativa "d" entre los elementos, llamada distancia o métrica, que satisface las propiedades de positividad estricta, simetría, reflexividad, y desigualdad triangular.

En muchas aplicaciones la función de evaluación de distancias (f.e.d.) d tiene un costo de cálculo tan alto que los otros costos se consideran despreciables, luego lo que pretenden las técnicas de búsqueda en espacios métricos es minimizar la cantidad de evaluaciones de distancia (e.d.).

La solución ingenua para resolver las consultas en espacios métricos corresponde a comparar exhaustivamente la llave de búsqueda con todos los elementos de la BD, lo que significa realizar tantas e.d. como elementos tenga la BD (que se denotará n). Si la f.e.d. es costosa, este mecanismo resulta impracticable para la mayoría de las aplicaciones reales.

1.2. Presentación del Tema

Para enfrentar el problema de buscar objetos en un espacio métrico se consideran algoritmos que construyen un índice a partir de los elementos de la BD, y en conjunto a la desigualdad triangular reducen la cantidad de e.d. al momento de realizar las búsquedas. Actualmente, las estrategias utilizadas consideran algoritmos basados en pivotes y algoritmos basados en clustering. Ambas estrategias pierden efectividad cuando la dimensión del espacio aumenta. Este hecho se conoce como la *maldición de la dimensionalidad*.

Vidal propone AESA [VR86], que es un algoritmo que presenta un buen desempeño incluso en dimensiones altas. Experimentalmente AESA muestra que tiene un tiempo de búsqueda O(1). El problema de este algoritmo es que la estructura donde realiza las búsquedas corresponde a una matriz en donde están precalculadas las $n(n-1)/2$ distancias entre pares de objetos, con lo que el costo en memoria del algoritmo llega a ser $O(n^2)$, lo que lo hace impracticable. Sin embargo, si fuese posible disminuir el uso de memoria de AESA, sería factible su uso en aplicaciones reales. El primer objetivo de esta tesis consiste en construir una estructura que permita estimar las $n(n-1)/2$ distancias con un error de estimación acotado, utilizando la menor cantidad de memoria posible.

Uno de los problemas estudiados en la teoría de grafos consiste en, dado un grafo G con aristas con pesos no negativos, encontrar un subgrafo G', tal que la distancia calculada entre cualquier par de nodos en G' sea a lo más t veces la distancia correspondiente en G. Esta estructura se denomina *t-Spanner* y se ajusta a los requerimientos de espacio en memoria y de error acotado en la estimación. El t-spanner necesita espacio subcuadrático de memoria para almacenar la información de las distancias que efectivamente guarda en la estructura de datos; por otro lado a partir de la información de estas distancias almacenadas, puede calcular una estimación del resto de las distancias, con una cota de error conocida t.

Los t-spanners tienen diversas aplicaciones, entre ellas: sistemas distribuidos, redes de comunicaciones, arquitectura de máquinas paralelas, robótica, geometría computacional y otros [PS89]. En este trabajo se utilizan los t-spanners como una herramienta para aproximar las distancias entre los elementos del espacio métrico.

Con esto, dado un conjunto de elementos pertenecientes a un espacio métrico, se puede considerar la matriz de todas las distancias como un grafo completo G a partir del cual se pretende construir un t-spanner G' que se constituya como una estructura apropiada para almacenar la información de los elementos y las distancias, con el fin de poder efectuar

consultas de recuperación de elementos minimizando la cantidad de e.d. en el momento de realizar las búsquedas.

Una vez desarrollada la estructura de búsqueda, se diseñarán e implementarán algoritmos para resolver búsquedas por rango. En una primera etapa, se pretende utilizar el t-spanner para aproximar al algoritmo de Vidal, en estudios posteriores se pretende utilizar el t-spanner como una herramienta para navegar en el espacio métrico y acercarse espacialmente a la consulta.

1.3. Objetivos

En esta tesis se propone una nueva metodología para enfrentar el problema de Búsqueda en Espacios Métricos, la cual considera un mapeo en un grafo $G'(V, E)$. En G', el conjunto de vértices V corresponde a los objetos y el conjunto de aristas E corresponde a una selección reducida de las distancias entre pares de objetos. Esto permite construir una estructura de datos con la cual se pueden resolver las búsquedas típicas en un espacio métrico a través de un mecanismo eficiente tanto para la navegación como para la recuperación de la información en el grafo. En adelante esta estructura se denominará t-spanner.

A continuación se presentan los objetivos generales y específicos que se formulan de modo de resolver los requerimientos de este trabajo de tesis.

1.3.1. Objetivos Generales

- Diseñar, implementar y medir la eficiencia en tiempo y memoria de algoritmos para construir una estructura de datos idónea para el problema de espacios métricos utilizando algoritmos de producción de t-spanners.
- Diseñar, implementar y medir la eficiencia en tiempo y memoria de algoritmos que resuelvan las consultas por rango, que utilicen el t-Spanner como estructura de base para la búsqueda.

1.3.2. Objetivos Específicos

- Proponer diversos métodos de construcción de t-spanners, analizarlos, y comparar su rendimiento en tiempo de procesamiento, número de e.d. y cantidad de arcos generados.
- Proponer mecanismos de solución de consultas por rango basados en el t-spanner como estructura de datos, y estudiar su performance en distintos tipos de espacios métricos, con énfasis en los de dimensión alta.
- Comparar el rendimiento de los algoritmos propuestos en número de e.d. con soluciones ya existentes [CNBYM01], en particular la de Vidal.
- Convertir el t-spanner en una estructura dinámica, que permita insertar y eliminar elementos del espacio métrico y se mantenga actualizada a bajo costo. Junto con esto,

diseñar mecanismos de optimización periódica de la estructura para restituir la calidad del t-spanner luego de inserciones y borrados sucesivos.

1.4. Descripción de los Contenidos

En el capítulo *Antecedentes*, se muestra una breve reseña teórica del problema de búsqueda en espacios métricos y del estado del arte de los algoritmos de construcción de t-spanners.

En el capítulo *Metodología*, se describe la planificación de las actividades realizadas durante el estudio, los criterios empleados durante el diseño de los algoritmos desarrollados en la tesis y la planificación de las pruebas experimentales.

En el capítulo *Algoritmos para Construcción de t-Spanners*, se describen los algoritmos para cumplir esta tarea que fueron diseñados durante esta tesis.

En el capítulo *El t-Spanner como Estructura para Búsquedas en Espacios Métricos*, se describe cómo se puede usar el t-spanner como la base de datos del espacio métrico, lo que permite insertar y borrar objetos, reconstruir la base de datos y buscar un objeto dentro de la base de datos.

En el capítulo *Resultados Experimentales*, se muestran las mediciones efectuadas para los diferentes algoritmos implementados y se entrega evidencia empírica que valida la utilidad de esta metodología en espacios métricos obtenidos del mundo real.

En el capítulo *Conclusiones*, se realiza una revisión global de los resultados con el fin de analizar la aplicabilidad de la metodología propuesta y se muestran las principales deducciones obtenidas de los resultados experimentales. Por último, se entregan recomendaciones para nuevos desarrollos en uso de t-Spanners para Búsqueda en Espacios Métricos.

Capítulo 2

Antecedentes

2.1. Espacios Métricos

A continuación se presentan conceptos básicos sobre *Espacios Métricos*; la referencia de esta sección es [CNBYM01].

Se define un espacio métrico como el par $(\mathbb{X}, d)$, donde $\mathbb{X}$ es el universo de los objetos, y d y una función $d : \mathbb{X} \times \mathbb{X} \longrightarrow \mathbb{R}$ que denotará una medida de distancia entre dichos objetos.

La función de distancia satisface las siguientes propiedades:

p1	$\forall x, y \in \mathbb{X}, d(x, y) \geq 0$	positividad,
p2	$\forall x, y \in \mathbb{X}, d(x, y) = d(y, x)$	simetría,
p3	$\forall x \in \mathbb{X}, d(x, x) = 0$	reflexividad,
p4	$\forall x, y \in \mathbb{X}, x \neq y \Rightarrow d(x, y) > 0$	positividad estricta,
p5	$\forall x, y, z \in \mathbb{X}, d(x, y) \leq d(x, z) + d(z, y)$	desigualdad triangular.

Sea $\mathbb{U}$ un conjunto finito de objetos, $\mathbb{U} \subseteq \mathbb{X}$ de tamaño $n = |\mathbb{U}|$, el conjunto de elementos de interés.

$\mathbb{U}$ se conoce como *diccionario*, *base de datos* o simplemente *conjunto de objetos* o *conjunto de elementos* y d se conoce como *distancia* o *métrica*.

2.1.1. Consultas de Proximidad

Existen dos tipos de consultas de interés en espacios métricos:

c1 **Consulta de rango** $(q, r)_d$: Recupera todos los elementos que estén dentro de la bola de radio r centrada en q, de acuerdo a la métrica d. Esto es $\{u \in \mathbb{U} / d(q, u) \leq r\}$.

c2 **Consulta de k vecinos más cercanos $NN_k(q)_d$:** Recupera los k elementos más cercanos a q, de acuerdo a la métrica d. Esto es recuperar un conjunto $A \subseteq \mathbb{U}$ tal que $|A| = k$ y $\forall u \in A, v \in \mathbb{U} - A, d(q, u) \leq d(q, v)$. Note que en caso de empate se escogerá cualquier conjunto de k elementos que satisfaga la condición.

El tipo de consulta más básico es la consulta de rango, la que en adelante se denotará como (q, r). En la Figura 2.1 se muestra una consulta de rango sobre un conjunto de elementos,

usando $\mathbb{R}^2$ en conjunto a la distancia euclidiana como espacio métrico. En ella se aprecia que del conjunto de elementos, sólo los azules satisfacen la consulta (q, r).

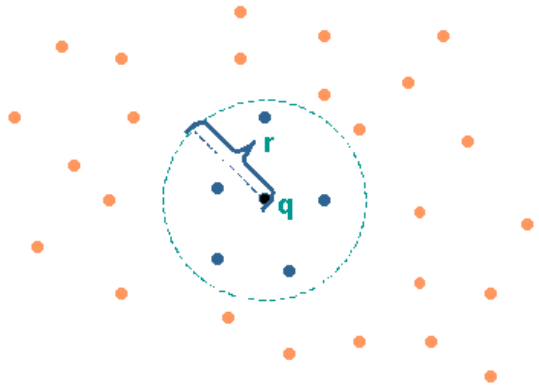


Figura 2.1: Ejemplo de consulta por rango.

El tiempo total para resolver una consulta se puede descomponer en:

$T = \#$ evaluaciones de distancia $\times$ costo de $d()$ + tiempo CPU extra + tiempo I/O

En muchas aplicaciones, la función de evaluación de distancias d tiene un costo de cálculo tan alto que los otros costos se consideran despreciables, luego lo que pretenden las técnicas de búsqueda en espacios métricos es minimizar la cantidad de e.d.

La solución ingenua para resolver las consultas en espacios métricos corresponde a comparar exhaustivamente la llave de búsqueda con todos los elementos de la BD, lo que significa realizar tantas e.d. como elementos tenga la BD. Si la f.e.d. es costosa, este mecanismo resulta impracticable para la mayoría de las aplicaciones reales.

Para enfrentar el problema de buscar objetos en un espacio métrico se consideran algoritmos que construyen un índice a partir de los elementos de la BD, y en conjunto a la desigualdad triangular reducen la cantidad de e.d. Actualmente, las estrategias utilizadas consideran algoritmos basados en pivotes y algoritmos basados en clustering. En la sección 2.1.3 se muestran algunas soluciones conocidas en la actualidad.

2.1.2. El Espacio Métrico de los Vectores

Un ejemplo de espacio métrico es $\mathbb{R}^k$, sobre el cual se puede definir una gran variedad de funciones de distancia, siendo frecuente el uso de la familia de métricas L_s definidas como:

$$L_s((x_1, \ldots, x_k), (y_1, \ldots, y_k)) = \left(\sum_{i=1}^{k} |x_i - y_i|^s \right)^{1/s}, \text{con } s \in \mathbb{R}, s \geq 1$$

Ejemplo de métricas pertenecientes a esta familia son: L_1, conocida como distancia Manhattan y L_2 o distancia Euclidiana. Además se puede definir L_∞ o distancia infinito como:

$$L_\infty((x_1, \ldots, x_k), (y_1, \ldots, y_k)) = \max_{i=1..k} |x_i - y_i|$$

Un espacio vectorial permite el uso de mayor información que un espacio métrico, puesto que se puede acceder directamente a la información geométrica y de las coordenadas, la cual no está disponible en los espacios métricos generales. En esta tesis sólo se utilizará la información de distancias entre vectores, sin utilizar la información de las coordenadas.

En espacios vectoriales existen algoritmos óptimos (en tamaño de la BD) tanto para el caso promedio como para el peor caso [BWY80] para búsqueda de puntos cercanos. Estructuras que abordan este problema son *kd*-trees [Ben75, Ben79], *R*-trees [Gut84], quad-trees [Sam84] y los más recientes *X*-trees [BKK96]. Estas técnicas hacen uso intensivo de la información de las coordenadas para agrupar y clasificar puntos en el espacio. Por ejemplo, el *kd*-tree divide el espacio entre las diferentes coordenadas y el *R*-tree agrupa los puntos en hiper-rectángulos. Desafortunadamente las técnicas existentes son muy sensibles a la dimensión del espacio vectorial. En efecto, los algoritmos de búsqueda de puntos cercanos tienen una dependencia exponencial con la dimensión del espacio (esto es lo que se conoce como la maldición de la dimensionalidad).

Las técnicas específicas para espacios vectoriales son un tema de investigación en sí, el cual no se intentará cubrir en este estudio, para mayor referencia sobre espacios vectoriales consulte [Sam84, WJ96, GG98, BBK02]. Esta tesis sólo se focalizará en espacios métricos generales.

2.1.3. Soluciones Actuales para Espacios Métricos

En la actualidad existen muchos algoritmos de búsqueda que indexan un espacio métrico. Se pueden diferenciar dos familias de algoritmos, la primera basada en el uso de pivotes, y la segunda en el uso de clusters [JD88]. Una aproximación distinta para enfrentar el problema de búsqueda en espacios métricos se muestra en [SW90], siendo este trabajo lo más cercano a la metodología propuesta en esta tesis; en la sección 2.1.6 se presenta esta aproximación.

Algoritmos basados en pivotes: La idea es usar un conjunto de k elementos distinguidos ("pivotes") $p_1, \ldots, p_k \in \mathbb{U}$ y almacenar para cada elemento u de la base de datos, la distancia hacia los k pivotes $(d(u, p_1), \ldots, d(u, p_k))$. Luego, dada la consulta q, se calcula la distancia hacia los k pivotes $(d(q, p_1), \ldots, d(q, p_k))$, y utilizando la desigualdad triangular, se eliminan los candidatos u tales que $|d(q, p_i) - d(u, p_i)| > r$, sin calcular explícitamente $d(q, u)$. Todos los otros elementos que no puedan ser eliminados utilizando esta regla se comparan directamente contra la consulta. Dentro de los algoritmos basados en pivotes se cuenta con:

- **BKT** Burkhard-Keller Tree [BK73],
- **FQT** Fixed-Queries Tree [BYCMW94],
- **FHQT** Fixed-Height FQT [BYCMW94],
- **FQA** Fixed-Queries Array [CMN99],
- **VPT** Vantage-Point Tree [Yia93],
- **MVPT** Multi-Vantage-Point Tree [BO97],
- **VPF** Excluded Middle Vantage Point Forest [Yia98].

- **AESA** Approximating Eliminating Search Algorithm [VR86],
- **LAESA** Linear AESA [MOV94].

Algoritmos basados en clusters: Esta idea consiste en dividir el espacio en zonas tan compactas como sea posible, y almacenar un punto representativo para cada zona ("centroide") e información adicional que permita descartar rápidamente la zona al momento de realizar la búsqueda. Luego de la primera división por zonas, cada zona se subdivide recursivamente, con lo que se produce una jerarquía de zonas o clusters. Con esto, dada la consulta q, se calcula la distancia hacia los centroides, y se descartan zonas, luego la búsqueda continua recursivamente en las zonas no descartadas. Dentro de los algoritmos basados en clustering se cuenta con:

- **BSTs** Bisector Trees [KM83],
- **GHT** Generalized-Hyperplane Tree [Uhl91],
- **GNAT** Geometric Near-neighbor Access Tree [Bri95],
- **VT** Voronoi Tree [DH87],
- **MT** M-tree [CPZ97],
- **SAT** Spatial Approximation Tree [Nav99].

Todos estos algoritmos pierden efectividad cuando la dimensión intrínseca del espacio aumenta (maldición de la dimensionalidad).

2.1.4. Algoritmo Básico de Búsqueda en Proximidad Usando Pivotes

Dada una consulta (q, r) y un conjunto de k pivotes $p_i \in \mathbb{U}$, por la desigualdad triangular se tiene que $d(p_i, u) \leq d(p_i, q) + d(q, u)$, con $u \in \mathbb{U}$, y de la misma forma se tiene que $d(p_i, q) \leq d(p_i, u) + d(q, u)$. De las inecuaciones anteriores se obtiene que una cota inferior para la distancia entre q y u es $d(q, u) \geq |d(p_i, u) - d(p_i, q)|$. Como los elementos u que interesan son aquellos en donde $d(q, u) \leq r$, entonces se pueden excluir todos los elementos que no cumplan la condición 2.1.

Ecuación 2.1 (Condición de exclusión para pivotes) *:*

$$d(u, q) \leq r \quad \Rightarrow \quad |d(q, p_i) - d(u, p_i)| \leq r, \forall i = 1 \ldots k$$

Si el espacio $\mathbb{U}$ posee n elementos, se construye un índice con las nk distancias $d(u, p_i)$, y por lo tanto al momento de realizar la consulta (q, r) en la primera fase de la búsqueda se calculan las k distancias $d(q, p_i)$, y se descartan los objetos u que no satisfagan la condición 2.1. A esto se le denomina **complejidad interna** de la consulta (q, r).

En una segunda fase, los elementos u no descartados por la condición 2.1 deben verificarse directamente con q y comprobar si verdaderamente se cumple la condición descrita en la consulta $c1$. A este cálculo de distancias adicionales se denomina **complejidad externa** de la consulta (q, r).

Por lo tanto, la complejidad total de la consulta (q, r) considerando algoritmos basados en pivotes, es la suma de sus complejidades interna y externa. En la práctica conviene hacer crecer k, de modo que para la segunda fase llegue la menor cantidad posible de candidatos.

2.1.5. Algoritmo AESA

AESA es un algoritmo que pretende aproximar la bola de la consulta (q, r) mediante la eliminación sucesiva de candidatos. Para esto escoge un elemento p en el conjunto $\mathbb{U}$ que se denomina pivote, calcula la distancia $d = d(p, q)$, y elimina a todos aquellos elementos u tal que $d(u, p)$ esté fuera del rango $[d-r, d+r]$. Luego toma un nuevo pivote dentro del conjunto de los que no son eliminados y repite el proceso hasta que se obtiene el conjunto de elementos dentro de la bola de la consulta (q, r). De esta forma AESA puede ser visto como un método basado en pivotes donde $k \to n$.

La selección del primer pivote es al azar, en cambio la selección del segundo pivote y de los siguientes sigue el criterio de escoger el pivote más promisorio dentro del conjunto de candidatos. La promisoriedad se define según la suma de las cotas inferiores de la distancia entre los elementos y la consulta (conocidas hasta el momento). Esto intenta elegir pivotes cercanos a q. Este criterio se muestra en la ecuación 2.2:

Ecuación 2.2 (Criterio de selección de pivotes en AESA) *:*

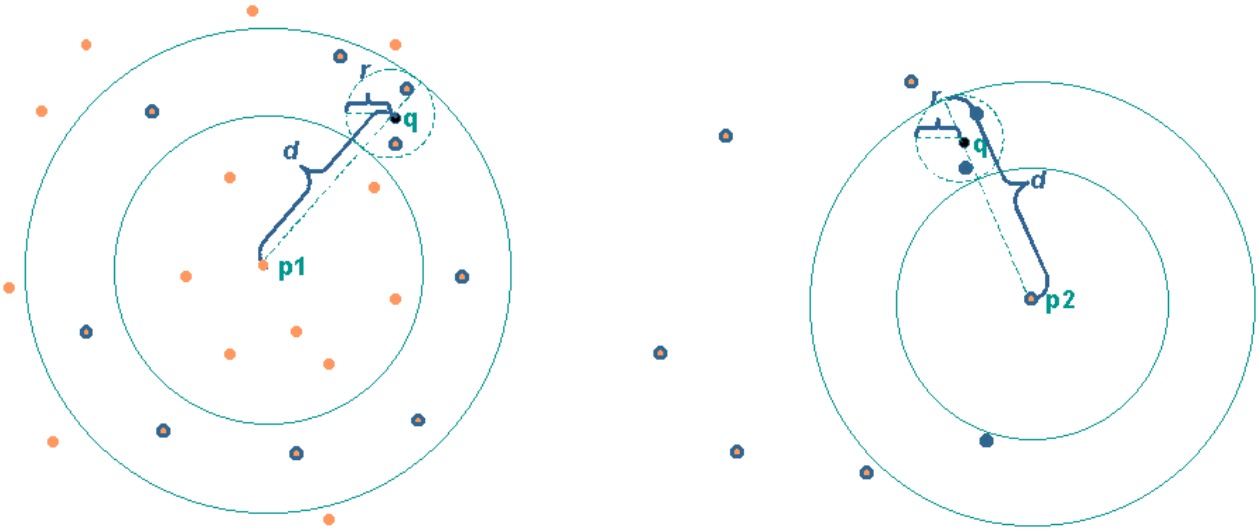
$$\text{Sea } sumLB(u) = \sum_{i=0}^{k-1} \Big| d(p_i, q) - d(p_i, u) \Big|$$

$$\text{Luego el pivote } p_k \leftarrow \text{argmin}_{u \in \, candidatos} \Big\{ sumLB(u) \Big\}$$

En la Figura 2.2 se muestran dos iteraciones de un ejemplo de ejecución de AESA. En (a) se muestra la primera eliminación de candidatos, para esto se escoge al pivote p_1, se calcula la distancia $d = d(p_1, q)$ y se establece el cascarón de exclusión de acuerdo al rango de distancias $[d - r, d + r]$. Con esto los elementos que pasan a la segunda iteración son los que están dentro del cascarón. En (b) se escoge un nuevo pivote p_2 dentro de los objetos que no fueron excluidos y que optimiza la promisoriedad entre los candidatos no eliminados, se calcula la distancia $d = d(p_2, q)$ y se establece el cascarón. Para la siguiente iteración sólo clasifican los tres objetos azules. En la Figura 2.3 se muestra el algoritmo AESA.

2.1.6. Algoritmo de Shasha y Wang

En [SW90] se muestra una metodología que emplea un grafo para buscar objetos en un espacio métrico. Este grafo contiene una colección arbitraria de distancias precalculadas entre



(a) (b)

Figura 2.2: Ejemplo de AESA.

Algoritmo 2.1 (AESA) *:*

```
AESA (Query q, Radio r, Objectos 𝕌)

Inicializaciones:
  distReal es la matriz de distancias entre todos los objetos de 𝕌
  candidatos ← 𝕌
  encontrados ← ∅
  for u ∈ 𝕌 sumLB(u) ← 0

Ciclo de eliminación de candidatos:
  while (candidatos ≠ ∅)
    p ← argmin_{u ∈ candidatos} {sumLB(u)}
    distPQ ← d(p,q)
    candidatos ← candidatos − {p}
    if (distPQ ≤ r) encontrados ← encontrados ∪ {p}
    for (u ∈ candidatos)
      if (distReal(p,u) ∉ [distPQ − r, distPQ + r])
        candidatos ← candidatos − {u}
      else sumLB(u) ← sumLB(u) + |distPQ − distReal(p,u)|
```

Figura 2.3: Algoritmo AESA.

objetos y se utiliza para estimar el resto de las distancias. Con esta idea, tras calcular algunas distancias entre pares de objetos, se pueden estimar el resto de ellas.

Para el realizar el cálculo de las estimaciones de distancia se utilizan dos matrices de tamaño $n \times n$. En la primera de ellas, *ADM*, se mantiene la estimación del valor mínimo de la distancia entre un par de nodos (obtenida utilizando la desigualdad triangular), y en la segunda, *MIN*, el valor máximo de la distancia (que corresponde al camino de costo mínimo entre el par de nodos). Para evitar confusiones en el uso de las matrices se reemplazarán los nombres originales ADM y MIN por *LB* (lower bound) y *UB* (upper bound) respectivamente.

Para obtener la matriz LB, para cada celda (i, j) de LB, se calcula el máximo, sobre todos los caminos, del mayor arco del camino menos la suma de los arcos menores restantes; y para calcular UB, se utiliza el algoritmo de Floyd para calcular las distancias mínimas entre todos los pares de nodos (ver sección 2.2.1). Debido a que el cálculo de LB puede llegar a ser muy costoso (pues tiene que revisar todos los caminos entre un par de nodos), y que tienen que usar el algoritmo de Floyd para calcular UB, proponen un algoritmo de programación dinámica de tiempo cúbico en n para obtener LB.

Con la matriz UB se puede excluir a los elementos cuya estimación máxima de distancia sea menor que la distancia entre el elemento y la bola de la consulta, lo cual es equivalente a eliminar los candidatos que están dentro del anillo interno del cascarón de exclusión (ver Figura 2.2). A su vez, con la matriz LB se puede excluir a los elementos cuya estimación mínima de distancia sea mayor que la distancia entre el elemento y la consulta, lo que corresponde a excluir a algunos de los elementos que están fuera del anillo externo del cascarón de exclusión.

Uno de los problemas de [SW90] es que requiere de memoria $O(n^2)$, lo que lo hace impráctico para aplicaciones reales, puesto que sólo es útil con bases de datos pequeñas. Por otro lado, los autores presentan resultados favorables en espacios métricos sintéticos en donde la distancia entre los pares de objetos sigue una distribución uniforme, lo que corresponde a espacios métricos de dimensión bajísima (en efecto, es un espacio métrico de dimensión 1, el cual no presenta utilidad práctica), y en los experimentos en dimensiones medias muestran que cuando la densidad de arcos considerada en el grafo es de un 50 %, se necesita inspeccionar un 20 % de la base de datos para resolver la consulta, y para una densidad de un 1 %, se necesita inspeccionar el 90 % de la base de datos para resolver la consulta, resultados que no son interesantes.

Las ideas de Shasha y Wang se acercan a las presentadas en esta tesis en el sentido de utilizar un grafo como un mapa aproximado de las distancias de la base de datos. Una diferencia importante es que ellos parten de un grafo arbitrario: si bien al final consideran qué grafos son mejores que otros, no usan ninguna propiedad particular de los grafos. Un elemento esencial de esta tesis es que se construye un grafo que satisface una determinada propiedad (la de t-spanner), que se explota para eliminar elementos al momento de la búsqueda. No todos estos elementos se podrían eliminar si no se supiera que el grafo tiene la propiedad de ser un t-spanner. Los buenos resultados experimentales obtenidos en esta tesis contrastan con los pobres resultados de Shasha y Wang.

2.2. t-Spanners

2.2.1. Definiciones Básicas para Grafos

A continuación se presentan conceptos básicos sobre *Teoría de Grafos*; la referencia de esta sección es [Wei95].

Un *grafo* $G = (V, A)$ consta de un conjunto de *vértices*, V, y un conjunto de *aristas*, A. Cada arista es un par (v, w), donde $v, w \in V$. A los vértices también se les llama *nodos* y a las aristas también se les llama *arcos*. Si el par es ordenado, entonces el grafo es dirigido (en esta tesis sólo se consideran grafos no dirigidos). El vértice w es *adyacente* a v, o *vecino* a v, si y sólo si $(v, w) \in A$. En un grafo no dirigido con arista (v, w), y por lo tanto (w, v), w es adyacente a v y v es adyacente a w. En ocasiones se tiene que una arista tiene una tercera componente, llamada *peso* o *costo*, en cuyo caso se habla de un *grafo ponderado*.

Un *camino* en un grafo es una secuencia de vértices $w_1, w_2, ..., w_n$ tal que $(w_i, w_{i+1}) \in A$ para $1 \leq i \leq n$. Para un grafo con aristas sin peso, la *longitud* de tal camino es el número de aristas del camino, que es igual a $n - 1$; por otro lado, cuando las aristas tienen peso, la longitud del camino es igual a la suma de los pesos de las aristas que lo forman; estos caminos se conocen como *caminos ponderados*. Se permiten caminos de un vértice a sí mismo; si este camino no contiene aristas, entonces el camino tiene longitud cero.

Un grafo no dirigido es *conexo* si hay un camino desde cualquier vértice a cualquier otro. Un grafo es *completo* si existen aristas entre cualquier par de vértices.

Un grafo es *disperso* (o *ralo*) cuando tiene $O(|V|)$ aristas y es *denso* cuando tiene $\Theta(|V|^2)$ aristas.

Se define la *adyacencia de un nodo* $u \in V$, como el conjunto formado por todos los nodos vecinos a u, formalmente $adyacencia(u) = \{v \in V \ / \ (u, v) \in E\}$. Note que la adyacencia de un nodo u no contiene al nodo u. Además se define la *adyacencia de un conjunto de nodos* $U \subseteq V$, como el conjunto formado por todos los nodos vecinos a U, formalmente $adyacencia(U) = \{v \in V \ / \ \exists \ u \in U, (u, v) \in E\} - U$.

Representaciones de grafos

Considere el grafo no dirigido de la Figura 2.4. Observe que los nodos están numerados iniciando en 1, y está compuesto por 7 vértices y 11 aristas.

Una forma sencilla de representar un grafo es con un arreglo bidimensional. Esta representación se denomina *matriz de adyacencia*. Para cada arista (u, v), se pone $a[u, v] = 1$ si $(u, v) \in A$, $a[u, v] = 0$ si no; para aristas con pesos no negativos $a[u, v] = peso_{u,v}$ si $(u, v) \in A$, $a[u, v] = \infty$ si no.

Cuando el grafo es no dirigido la matriz de adyacencia es simétrica, luego basta guardar los costos de la matriz triangular superior. Esta representación es extremadamente sencilla, requiere memoria $\Theta(|V|^2)$ y es útil para el caso de grafos densos. Para grafos dispersos esta representación no resulta adecuada puesto que reserva espacio para todos los arcos, con lo que se tiene un exceso de reserva de memoria.

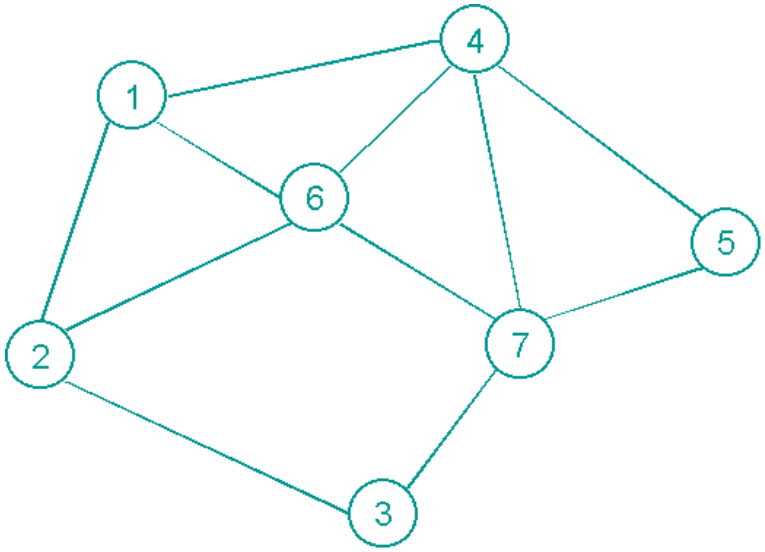


Figura 2.4: Grafo no dirigido.

Para el caso de los grafos dispersos se recomienda el uso de una *lista de adyacencia*. Para cada vértice $v \in V$, se mantiene la lista $adyacencia(v)$, con lo que el uso de memoria es $O(|V| + |A|)$. Cuando se tiene aristas con peso, junto con guardar el vértice adyacente se guarda el peso de la arista. Cuando el grafo es no dirigido, dado que $\forall (u, v) \in A, (v, u) \in A$, es necesario guardar, para cada arista, su arista simétrica.

En el Cuadro 2.1 se muestra la matriz de adyacencia del grafo de la Figura 2.4 (pero esta vez se le asignan pesos a las aristas del grafo) y en la Figura 2.5 la lista de adyacencia para el mismo grafo (considerando los pesos de las aristas).

	1	2	3	4	5	6	7
1	0	7.5	∞	6.2	∞	4	∞
2		0	10	∞	∞	7.5	∞
3			0	∞	∞	∞	4
4				0	8.5	5.2	8.3
5					0	∞	5.5
6						0	6
7							0

Cuadro 2.1: Matriz de adyacencia del grafo de la figura 2.4.

Algoritmo de Dijkstra

Una descripción extensa de Dijkstra se muestra en [Wei95]. Este algoritmo resuelve el problema de, dado un grafo ponderado, $G(V, A)$, y un vértice distinguido, s, encontrar el camino de mínimo costo de s a cada uno de los demás vértices de G. El algoritmo de Dijkstra (ver Figura 2.6) está basado en insertar nodos al conjunto de vértices conocidos (que inicialmente es vacío), hasta recorrer todos los nodos de V. Para esto el algoritmo realiza

1	-	(2, 7.5)	-	(4, 6.2)	-	(6, 4)		
2	-	(1, 7.5)	-	(3, 10)	-	(6, 7.5)		
3	-	(2, 10)	-	(7, 4)				
4	-	(1, 6.2)	-	(5, 8.5)	-	(6, 5.2)	-	(7, 8.3)
5	-	(4, 8.5)	-	(7, 5.5)				
6	-	(1, 4)	-	(2, 7.5)	-	(4, 5.2)	-	(7, 6)
7	-	(3, 4)	-	(4, 8.3)	-	(5, 5.5)	-	(6, 6)

Figura 2.5: Lista de adyacencia del grafo de la figura 2.4, la notación es (vértice, costo).

dos ciclos anidados, el primero para recorrer todos los vértices del grafo, y el segundo para encontrar la distancia más corta entre los nodos conocidos y el resto de los vértices y luego para actualizar los cálculos de distancia hacia los nodos desconocidos. La implementación típica de Dijkstra toma tiempo $O(n^2)$, con $n = |V|$, y realiza una búsqueda secuencial sobre el arreglo de distancias para determinar la de menor costo. Una implementación alternativa de este algoritmo utiliza una cola de prioridad para mantener las distancias hacia el resto de los nodos, lo que permite encontrar el vértice con la distancia más corta en tiempo $O(\log n)$. Con esto se obtienen tiempos de cálculo $O(m \log n)$, con $m = |A|$ y $n = |V|$ lo que constituye una disminución importante de tiempo para el caso de grafos dispersos. En esta tesis se utiliza la versión de Dijkstra optimizada para grafos dispersos.

Algoritmo 2.2 (Dijkstra) *:*

```
Dijkstra (Grafo G, Nodo inicio, Tabla T)

Inicializaciones:
  for (i ← 0 .. n − 1)
    T[i].conocido ← Falso
   †T[i].distancia ← ∞
    T[i].camino ← null
  T[inicio].distancia ← 0

Cálculo de distancias:
  for (i ← 0 .. n − 1)
    v ← argmin_{v, T[v].conocido = Falso} T[v].distancia
    T[v].conocido ← Verdadero
    for (cada w adyacente a v)
      if (T[w].conocido = Falso)
       ‡if (T[v].distancia + costo(v,w) < T[w].distancia)
       ‡  T[w].distancia ← T[v].distancia + costo(v,w)
          T[w].camino ← v
```

Figura 2.6: Algoritmo de Dijkstra.

Algoritmo de Floyd

El algoritmo de Floyd [Wei95] (ver Figura 2.7) se utiliza para calcular los caminos más cortos entre todos los pares de vértices, toma tiempo $O(n^3)$ y se recomienda para grafos densos. Para grafos dispersos, se recomienda ejecutar n iteraciones de Dijkstra considerando cada nodo como vértice de origen de los cálculos, lo que toma un costo en tiempo $O(n \cdot m \log n)$.

Algoritmo 2.3 (Floyd) *:*

```
Floyd (Grafo G, Matriz D)

Inicializaciones:
  for (i ← 0 .. n − 1) D[i, i] ← 0
  for (i, j ← 0 .. n − 1, i ≠ j) D[i, j] ← ∞

Cálculo de distancias:
  for (k ← 0 .. n − 1)
    for (i, j ← 0 .. n − 1) D[i, j] ← mín(D[i, j], D[i, k] + D[k, j])
```

Figura 2.7: Algoritmo de Floyd.

Árbol de cobertura de costo mínimo

Un árbol de cobertura de costo mínimo (MST, del inglés Minimum Spanning Tree) de un grafo no dirigido G, es un árbol formado a partir de las aristas que conectan todos los vértices de G con un costo total mínimo.

El MST puede no ser único para un grafo G dado. Note que el número de aristas del MST es $|V| - 1$. El MST es un árbol porque es acíclico, es de cobertura porque cubre todos los vértices y es de costo mínimo por que la suma de los costos de las aristas del MST es la menor posible. Los álgoritmos básicos para resolver este problema son de tipo *greedy* (algoritmos avaros), y los más conocidos son el algoritmo de Kruskal y el de Prim ([Wei95]). El cálculo del MST toma tiempo $O(n^2)$ para grafos densos y $O(m \log m)$ para grafos dispersos, y tanto Prim como Kruskal admiten ambas implementaciones. En esta tesis se utiliza Prim de orden $O(n^2)$, por dos razones:

1. Prim necesita memoria $O(n)$ para calcular el MST, Kruskal necesita memoria $O(m)$ que para el caso de grafos completos es $O(n^2)$.

2. Los MST utilizados en esta tesis se construyen sobre grafos completos.

El algoritmo de Prim es esencialmente idéntico al de Dijkstra para caminos más cortos y sólo difiere en la condición de actualización. Prim actualiza si el peso del arco que une el nodo que se está agregando al conjunto de conocidos, u, con algún nodo desconocido, v, es menor que el costo que se tenía para v previamente. Esto se obtiene modificando las líneas marcadas con ‡ en la Figura 2.6 por:

```
if (costo(v,w) < T[w].distancia)
  T[w].distancia ← costo(v,w)
```

2.2.2. Definición de t-Spanner

Considere un grafo $G = G(V, A)$ formado por un conjunto de vértices V y un conjunto de arcos con pesos no negativos A. En este grafo es posible determinar los caminos más cortos entre todos los pares de vértices. Este problema se puede resolver utilizando $|V|$ iteraciones del algoritmo de cálculo de caminos más cortos (Dijkstra), considerando todos los vértices como nodo de origen; o con el algoritmo de Floyd [Wei95].

Un t-spanner es un grafo $G' = G'(V, E), E \subseteq A$ (note que utiliza el mismo conjunto de vértices y un subconjunto de los arcos), construido a partir del grafo G original, considerando una versión relajada de la condición de encontrar los caminos más cortos entre pares de nodos. La principal propiedad de un t-spanner es que garantiza que el costo del camino entre cualquier par de vértices del grafo G' es a lo sumo t veces la distancia del camino mínimo obtenido en el grafo original G [PS89], lo que se conoce como *condición de t-spanner*. Note que al menos necesita $|V| - 1$ arcos, para que el grafo G' sea conexo y que a lo más necesita insertar los A arcos del grafo original G.

2.2.3. Ejemplo de Construcción de un t-Spanner

Para ilustrar el concepto de t-spanner, se muestra un ejemplo de construcción de un t-spanner dado un grafo completo inicial definido sobre puntos en $\mathbb{R}^2$. Considere el grafo de la Figura 2.8 y la matriz de costos reales mostrada en el Cuadro 2.2. Como es una matriz simétrica sólo se muestran los valores de la matriz triangular superior. Se habla de costos reales en el sentido que son efectivamente el costo del arco entre dos nodos.

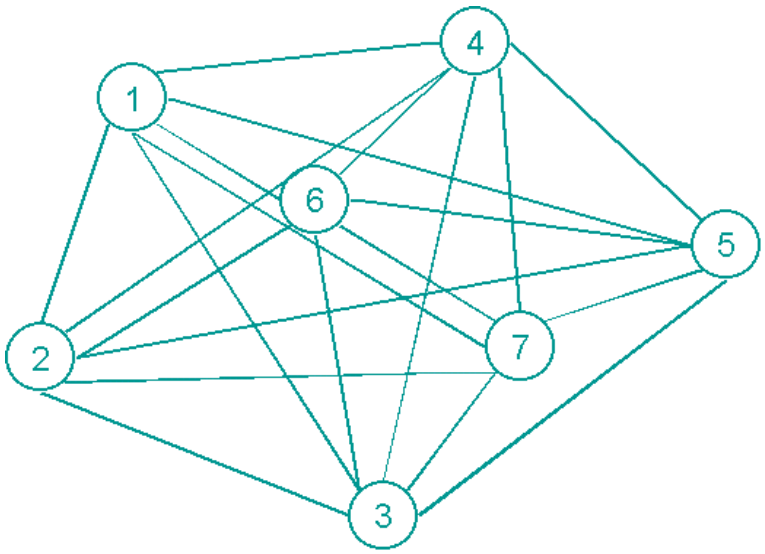


Figura 2.8: Ejemplo de construcción de un t-spanner (grafo inicial).

	1	2	3	4	5	6	7
1	0	7.5	12.5	6.2	13.5	4	10
2		0	10	12.5	16	7.5	10.5
3			0	12	9	8.5	4
4				0	8.5	5.2	8.3
5					0	9.7	5.5
6						0	6
7							0

Cuadro 2.2: Matriz de costos reales del grafo de la Figura 2.8.

Suponiendo un valor de $t = 1.3$, lo cual significa que se permite que el valor de la estimación de la distancia sea a lo más un 30 % superior que el valor real, se tienen los valores máximos permitidos para la estimación de distancias mostrados en el Cuadro 2.3.

	1	2	3	4	5	6	7
1	0	9.75	16.25	8.06	17.55	5.2	13
2		0	13	16.25	20.08	9.75	13.65
3			0	15.6	11.7	11.05	5.2
4				0	11.05	6.76	10.79
5					0	12.61	7.15
6						0	7.8
7							0

Cuadro 2.3: Matriz de estimaciones máximas del grafo de la Figura 2.8 considerando un valor de $t = 1.3$.

El t-spanner inicial está compuesto sólo por los vértices, a partir de éste se agregan arcos hasta cumplir la condición de estimación de distancias del t-spanner. Una buena idea es insertar los arcos que conforman el árbol de expansión mínimo (MST) del grafo de la Figura 2.8. Esto se ve en la Figura 2.9 y las estimaciones de este t-spanner en el Cuadro 2.4. Los arcos reales están escrito en tipografía normal, las estimaciones correctas en *cursiva* y las estimaciones incorrectas en **negritas**.

	1	2	3	4	5	6	7
1	0	7.5	*14*	**9.2**	*15.5*	4	*10*
2		0	**21.5**	**16.7**	**23**	**11.5**	**17.5**
3			0	*15.2*	*9.5*	*10*	4
4				0	**16.7**	5.2	**11.2**
5					0	*11.5*	5.5
6						0	6
7							0

Cuadro 2.4: Matriz de estimación de costos del grafo de la Figura 2.9.

Como no es suficiente con los arcos del MST es necesario insertar más arcos. Otra buena idea

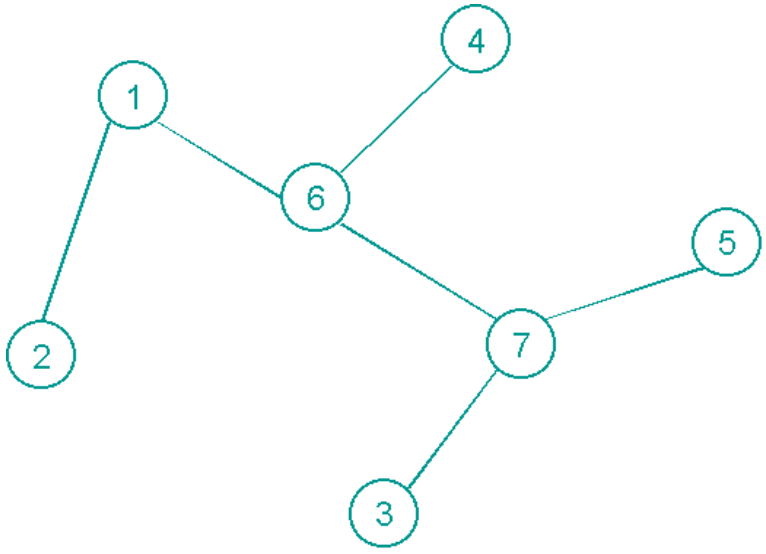


Figura 2.9: Ejemplo de construcción de un t-spanner. Se adicionaron los arcos que conforman el MST del grafo de la Figura 2.8.

es insertar arcos que estén mal estimados, pues esto asegura que al menos un arco (el que se inserta) mejora su estimación. En esta oportunidad se insertan los arcos (1,4), (2,6) y (4,7), lo que se ve en la Figura 2.10, y las estimaciones de este t-spanner en el Cuadro 2.5.

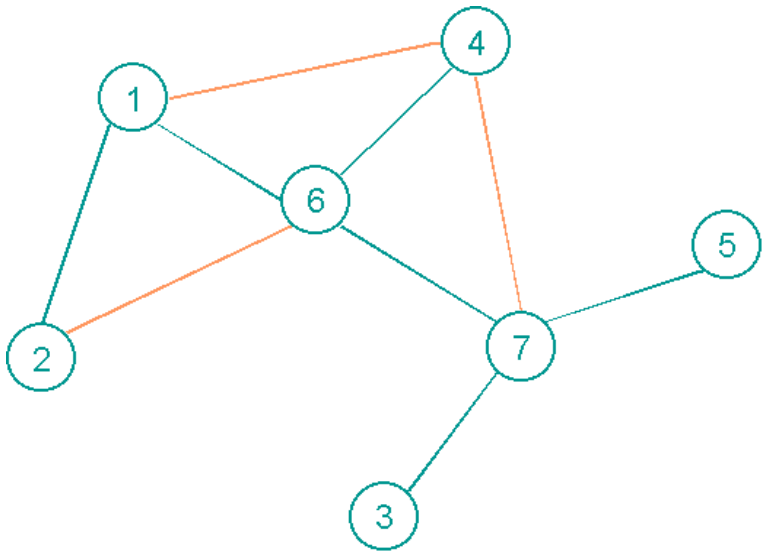


Figura 2.10: Ejemplo de construcción de un t-spanner. Se adicionaron los arcos (1,4), (2,6) y (4,7).

Con los arcos actualmente insertados se t-estiman las distancias restantes, salvo dos arcos que están mal estimados: (2,3) y (4,5). En estos casos, cuando son muy pocos los arcos que faltan por estimar, simplemente se insertan y con esto termina la construcción. El 1.3-spanner resultante se ve en la Figura 2.11 y su matriz de estimación de costos en el Cuadro 2.6.

	1	2	3	4	5	6	7
1	0	7.5	*14*	6.2	*15.5*	4	*10*
2		0	**17.5**	*12.7*	*19*	7.5	*13.5*
3			0	*15.2*	*9.5*	*10*	4
4				0	**13.8**	5.2	8.3
5					0	*11.5*	5.5
6						0	6
7							0

Cuadro 2.5: Matriz de estimación de costos del grafo de la Figura 2.10.

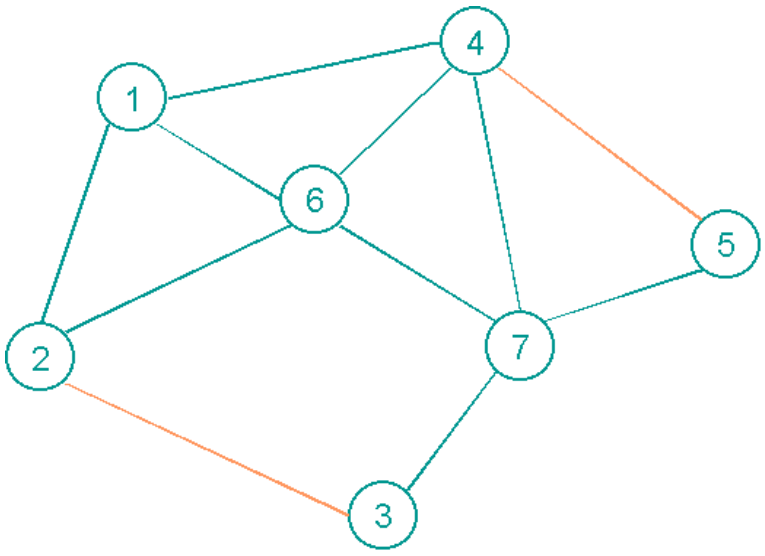


Figura 2.11: Ejemplo de construcción de un t-spanner. Se adicionaron los arcos (2,3) y (4,5).

	1	2	3	4	5	6	7
1	0	7.5	*14*	6.2	*15.5*	4	*10*
2		0	10	*12.7*	*19*	7.5	*13.5*
3			0	*15.2*	*9.5*	*10*	4
4				0	8.5	5.2	8.3
5					0	*11.5*	5.5
6						0	6
7							0

Cuadro 2.6: Matriz de estimación de costos del grafo de la Figura 2.11.

Es importante destacar tres hechos:

- Sólo se utilizaron 11 arcos para construir el 1.3-spanner, con lo que se consigue un ahorro de memoria de un 47.6 %.
- La inserción de un arco puede mejorar la estimación de costos de otros arcos en el grafo, pues puede reducir el costo a caminos entre otros nodos.
- Dado un grafo $G = G(V, A)$ el t-spanner $G' = G'(V, E)$ no es único. Debido a esto, los algoritmos de producción de t-spanners pretenden obtener resultados de la mejor calidad posible, es decir, pretenden minimizar la cantidad de arcos contenidos en el t-spanner.

2.2.4. Relación entre Espacios Métricos y t-Spanners

Dado un conjunto de objetos pertenecientes a un espacio métrico, es posible construir un grafo completo $G(V, A)$ en el cual el conjunto de vértices V corresponde a los objetos del espacio métrico, y el conjunto de arcos A corresponde a las $|V|\left(|V|-1\right)/2$ distancias entre dichos objetos. El grafo así construido, similar a la estructura de datos de AESA (sección 2.1.5), requiere $O(|V|^2)$ memoria para almacenar la información asociada a las distancias, siendo esto un costo de memoria muy alto para la mayoría de las aplicaciones prácticas. Aplicando los algoritmos de producción de t-spanners se pretende construir un subgrafo $G'(V, E)$ que represente a los objetos del espacio métrico, considerando un pequeño número de arcos $E \subseteq A$, de modo de reducir la memoria necesaria para almacenar la estructura del grafo, y aún así aproximar las distancias entre objetos con un error máximo t, es decir:

Ecuación 2.3 (Aproximación de distancias en el t-spanner) *:*

$$d(u, v) \quad \leq \quad d_{t-\text{Spanner}}(u, v) \quad \leq \quad t \cdot d(u, v)$$

La primera desigualdad es clara, y la segunda, corresponde a la condición de t-spanner.

Uno de los efectos de la maldición de la dimensionalidad es que a medida que la dimensión del espacio aumenta, el histograma de distancias entre todos los nodos se estrecha. De esto nace una restricción en el diseño de los algoritmos de construcción de t-spanners: sólo se permitirán valores $t \leq 2$. Esto se debe a que valores de $t > 2$ disminuyen fuertemente la capacidad de eliminación de objetos al utilizar el t-spanner como estructura de soporte para realizar las búsquedas.

El hecho de que los vértices pertenecen a un espacio métrico y, en particular, que la métrica satisface la desigualdad triangular, se traduce en que la distancia más corta entre dos vértices cualesquiera utilizando el grafo, es precisamente el arco directo que conecta ambos vértices, y que tiene como costo asociado la distancia medida según la métrica del espacio. Esto entrega un mecanismo directo para conocer la distancia mínima entre dos objetos y el valor máximo que se permite en la estimación de distancias.

2.2.5. Estado del Arte

Un t-Spanner se construye agregando aristas a un grafo $G' = G'(V, \emptyset)$ hasta que G' cumple la condición de t-spanner. La tarea más costosa dentro este proceso es la verificación de la condición de t-spanner.

Se han desarrollado estudios sobre el tema de los t-spanners generales [Epp99, PS89, PU89]. En la mayoría de ellos proponen el algoritmo básico presentado en la sección 2.2.6, que tiene costo $O(mn^2)$, donde $n = |V|$ y $m = |E|$, ambos valores referidos al t-spanner resultante. En [ADDJ90, ADD$^+$93] se muestran técnicas que producen t-spanners con $n^{1+O(\frac{1}{t-1})}$ arcos, para grafos generales con n nodos, sin embargo, para $t < 2$ este resultado no es de interés.

Construir un t-spanner óptimo es un problema computacionalmente difícil, en [PS89] los autores prueban que el problema de construir un t-spanner con a lo más m arcos es NP-completo en el caso en que G es un grafo compuesto por arcos con peso unitario; también muestran que dado un grafo G y dos enteros t y m, el problema de determinar si G tiene o no un t-spanner con a lo más m arcos es NP-completo, e incluso muestran que para un valor de t fijo, el problema sigue siendo NP-completo.

Una versión relajada del problema es encontrar un algoritmo $\mathcal{A}$ con alguna garantía de rendimiento, es decir, para cualquier grafo G y cualquier t, $\mathcal{A}$ construye un t-spanner G' de G tal que $size(G') \leq c(n) \cdot e_{opt}$ o $wt(G') \leq c(n) \cdot wt_{opt}$, donde $size(G')$ es la cantidad de arcos del t-spanner producido, $wt(G')$ es la suma de los pesos de los arcos del t-spanner producido, $c(n)$ es una constante o una función de n de crecimiento lento, e_{opt} es la cantidad de arcos del t-spanner óptimo y wt_{opt} la suma de los pesos de los arcos del t-spanner óptimo. Este problema relajado aún sigue abierto. Para el caso especial de $t = 2$ y grafos con arcos de peso unitario se tienen algoritmos con $c(n) = \log(\frac{|E|}{|V|})$ [KP94].

Otros trabajos en grafos generales ([CDNS95, Coh98]) presentan algoritmos que construyen t-spanners utilizando distintas técnicas de verificación.

En [Coh98] se presentan algoritmos sofisticados que construyen t-spanners con costo $O(n^{1+(2+\varepsilon)(1+\log_n m)/t})$ arcos y $O(mn^{(2+\varepsilon)(1+\log_n m)/t})$ en tiempo, con $n = |V|$ y $m = |A|$ (A es el conjunto de todas las distancias entre pares de objetos), para cualquier $\varepsilon > 0$. En un espacio métrico $m = \Theta(n^2)$, lo cual significa que se obtienen t-spanners con al menos $O(n^{1+6/t})$ arcos en tiempo $O(n^{2+6/t})$. Adicionalmente, el interés de este estudio es el caso $t < 2$, y los algoritmos propuestos en [Coh98] consideran que $t \in [2, \log n]$, lo cual inhabilita a estos algoritmos en el ámbito de estudio de los espacios métricos.

En el ámbito de los grafos euclidianos, que corresponde a la subclase de espacios métricos donde los objetos son puntos en un espacio D-dimensional utilizando la distancia euclidiana, se tienen mejores resultados. En [Epp99, ADDJ90, ADD$^+$93, Kei88, GLN00, RS91] se presentan algoritmos para grafos euclidianos que obtienen un costo $O(n \log^{D-1} n)$ en tiempo y $O(n)$ en el número de arcos (donde $n = |V|$), utilizando intensamente las propiedades geométricas de los objetos. Esta información no está disponible en el caso de objetos pertenecientes a espacios métricos generales, motivo por el cual estos algoritmos son restrictivos para el caso en estudio.

También se tienen resultados relacionados que contemplan aproximaciones probabilísticas utilizando árboles métricos [Bar98, CCG$^+$98]. En ellos la idea es construir un conjunto de

árboles métricos tales que su unión produzca un t-spanner con alta probabilidad. Sin embargo, los valores de t son de la forma $O(\log n \log\log n)$.

De esto surge la necesidad de encontrar algoritmos que permitan construir t-spanners apropiados para espacios métricos, vale decir con $t < 2$, para grafos completos, resistentes a una dimensionalidad alta, con costos de tiempo y memoria razonables y que puedan sacar partido al hecho de que en un espacio métrico los pesos respetan la desigualdad triangular. Es necesario recordar que al generar un t-spanner no sólo importan los recursos necesarios para construirlo, sino también la calidad del resultado, en el sentido de que se buscan t-spanners con la menor cantidad posible de arcos.

2.2.6. Algoritmo Básico de Construcción de t-Spanners

La idea intuitiva para resolver este problema es iterativa. Se comienza con un t-spanner inicial, que sólo contiene los vértices y que no contiene arcos, en donde se calculan las estimaciones de todas las distancias entre pares de vértices, las cuales son todas infinito en la iteración 0, salvo las distancias entre un nodo y sí mismo ($d(u, u) = 0$). Luego se insertan arcos hasta que todas las estimaciones de distancia cumplan la propiedad del t-spanner.

La revisión de los arcos se realiza ascendentemente de acuerdo al costo de cada arista. Esto significa que es necesario un procesamiento previo de las distancias para ordenarlas de menor a mayor. El hecho de utilizar primero los arcos de menor costo está de acuerdo con la idea geométrica de insertar arcos entre vecinos cercanos, lo cual a larga se traduce en que las estimaciones de distancia se realizan a través de caminos compuestos por los arcos demenor valor posible, y en que se usan menos arcos.

De este modo el algoritmo utiliza dos matrices. La primera, *real*, contiene las distancias reales entre todos los objetos y la segunda, *estim*, contiene las estimaciones de las distancias conseguidas con el t-spanner en construcción. El t-spanner se almacena en una lista de adyacencia.

El criterio de inserción es que un arco se agrega al conjunto E sólo cuando la estimación que se tiene de él no satisface el criterio del t-spanner.

Luego de insertar el arco, es necesario actualizar todas las estimaciones de distancia. El mecanismo de actualización utilizado es similar al mecanismo de cálculo de distancias de Floyd, pero considerando que se insertan arcos al conjunto de conocidos (Floyd inserta nodos al conjunto de conocidos).

La Figura 2.12 corresponde al algoritmo básico de construcción de t-spanners.

2.2.7. Análisis del Algoritmo Básico de Construcción de t-Spanners

Para este análisis y los posteriores se denotará por n al tamaño del conjunto de nodos ($|V|$), y por m al tamaño del conjunto de arcos ($|E|$), con $n - 1 \leq m \leq n(n-1)/2$.

Desde el punto de vista de la cantidad de las e.d., este algoritmo calcula en tiempo de

Algoritmo 2.4 (t-Spanner 0) *:*

```
t-Spanner0 (Tolerancia t, Vertices 𝕌)

Inicializaciones:
  real ← matriz de distancias entre todos los vértices
  estim ← matriz en donde se actualizan las estimaciones de costo.
  for (u, v ∈ 𝕌) estim(u, v) ← 0 si u = v, ∞ si no
  t-Spanner ← 0 // estructura de arcos del t-spanner
  Ordena las distancias en real de menor a mayor

Ciclo de actualización:
  for (e = (e_u, e_v) ∈ real, escogido en orden ascendente)
    if (estim(e) > t * real(e)) // e no está bien t-estimado
    t-Spanner ← t-Spanner ∪ {e}
    Es necesario recalcular todas las estimaciones de distancia
       for (v_i, v_j ∈ 𝕌)
           d_1 ← estim(v_i, e_u) + estim(v_j, e_v)
           d_2 ← estim(v_j, e_u) + estim(v_i, e_v)
           estim(v_i, v_j) ← min(estim(v_i, v_j), min(d_1, d_2)+real(e))
```

Figura 2.12: Algoritmo básico de construcción de t-spanners (t-Spanner 0).

construcción $O(n^2)$ e.d., que es el costo de construcción de AESA [VR86].

Desde el punto de vista del tiempo de procesamiento, este algoritmo inserta m arcos al t-spanner. Considerando que para cada inserción es necesario actualizar las n^2 distancias, el costo en tiempo de procesamiento es de $O(mn^2)$. Se debe recordar que para que el t-spanner sea conexo, se necesitan al menos $n-1$ arcos.

Desde el punto de vista de la memoria utilizada, requiere $O(n^2) + O(m)$, ya que necesita almacenar toda la matriz de distancias estimadas, más la estructura del t-spanner. Con respecto a las distancias reales se pueden mantener en memoria en otra matriz (tal como lo hace este algoritmo), o calcularlas a medida que se necesitan, pero en este caso se calcularán entre $O(n^3)$ y $O(n^4)$ distancias.

Aún cuando este algoritmo es cuidadoso al momento de insertar los arcos, tiene dos inconvenientes:

1. Actualiza todas las distancias, incluso aquellas en donde no tiene sentido realizar la actualización.
2. Requiere mucha memoria.

Los algoritmos de producción de t-spanners desarrollados en esta tesis pretenden mejorar estas deficiencias de tiempo de procesamiento y de memoria.

Capítulo 3

Metodología

3.1. Planificación de las Actividades

Se contemplan dos fases en el desarrollo de esta tesis. La primera se focaliza en la producción de t-spanners y la segunda en la recuperación de la información desde el t-spanner.

- La fase 1 consiste en:
  - Estudio del estado del arte de los algoritmos de construcción de t-Spanners.
  - Diseño, implementación y pruebas de algoritmos de construcción de t-Spanners apropiados para espacios métricos.
  - Comparación y selección del mejor algoritmo de construcción de t-Spanners considerando e.d., tiempo de procesamiento y memoria.
  - Diseño, implementación y pruebas de un mecanismo de inserción y borrado de nodos en el t-Spanner.
  - Diseño, implementacón y pruebas de un mecanismo de reconstrucción del t-Spanner.
- La fase 2 consiste en:
  - Diseño, implementación y pruebas del algoritmo AESA simulado sobre la estructura del t-spanner.
  - Comparación de rendimiento con soluciones ya existentes: algoritmo estándar basado en pivotes y AESA.

3.2. Construcción de t-Spanners

Se desarrollaron cinco algoritmos de producción de t-spanners. Con cada algoritmo nuevo se pretendía mejorar alguna deficiencia del algoritmo desarrollado anteriormente. Esta evolución pasa por las siguientes versiones:

1. Optimización del mecanismo de actualización de estimaciones del algoritmo básico. En esta versión se inserta un arco a la vez, pero se tiene el cuidado de evitar propagar los cálculos donde no tiene sentido recalcular la estimación de distancias.

2. Disminución del tiempo de procesamiento. Si bien es cierto que el algoritmo anterior se cuida de no propagar excesivamente los cálculos, el hecho de actualizar estimaciones para cada arco insertado significa un gran esfuerzo computacional en el procesamiento. En la segunda versión se pretende disminuir la cantidad de cálculos en tiempo de procesamiento, actualizando las estimaciones luego de insertar varios arcos, de modo de amortizar los costos, pero tratando de no insertar arcos que eventualmente estén bien t-estimados.

3. Inserción incremental de nodos. A diferencia de los dos algoritmos anteriores (que analizan el conjunto completo de objetos y distancias), en esta versión se prueba la alternativa de insertar un nodo a la vez, de modo que en cada iteración el t-spanner en crecimiento satisfaga la t-condición. Una de las ventajas de esta opción es que sólo evalúa $O(n^2)$ veces la f.e.d., en cambio, dado que no considera globalmente los nodos, forzosamente sus resultados son subóptimos en lo que se refiere al tamaño del t-spanner.

4. Algoritmo recursivo. Esta versión nace de la necesidad de permitir que el algoritmo de inserción incremental actúe en forma global. La idea es recursivamente dividir el conjunto de nodos en dos y luego mezclar las soluciones aplicando el algoritmo incremental.

5. Recursividad y optimización del mecanismo de actualización. La última versión considerada en este estudio consiste en dividir recursivamente el conjunto de nodos en dos, aplicar el algoritmo básico optimizado en el caso base y luego mezclar las soluciones con el algoritmo incremental.

Dado que los vértices pertenecen a un espacio métrico, se utilizará la propiedad de que la distancia más corta entre dos vértices cualesquiera utilizando el grafo, es precisamente el arco directo que conecta ambos vértices, y que tiene como costo asociado la distancia medida según la métrica del espacio.

La metodología de producción de t-spanners se basa en la selección de distancias de menor costo, lo que está de acuerdo con la idea geométrica de insertar arcos entre vecinos cercanos. Esto a la larga se traduce en que las estimaciones de distancia (en el t-spanner) se realizan a través de caminos compuestos por los arcos de menor valor posible, y en que sólo se utiliza un arco de ser necesario. Por otro lado cualquier parámetro que sea necesario ajustar vía prueba y error intenta no aumentar excesivamente ni el uso de memoria ni tiempo de procesamiento.

Por último se debe indicar que los algoritmos propuestos son del tipo *greedy*.

3.3. Uso de t-Spanners para Búsquedas en Espacios Métricos

Para utilizar el t-spanner como la estructura de la base de datos es necesario otorgarle cierta flexibilidad. Para ello se diseñan mecanismos de inserción y borrado de objetos y un algoritmo de reconstrucción de la estructura. Para la implementación de estos mecanismos se reutilizan algunos de los algoritmos de construcción.

Adicionalmente se diseña un algoritmo de consultas en la base de datos. Dado que AESA sigue siendo el mejor algoritmo de recuperación en espacios métricos, se pretende simular AESA sobra la estructura del t-spanner.

Para resolver las consultas es necesario calcular la estimación de la distancia desde el pivote hacia el resto de los candidatos. Esto significa que cada vez que se escoge un pivote se necesita calcular los caminos de costo mínimo con Dijkstra, que dependiendo de la versión que se utilice tiene costo entre $O(n_p \cdot |E| \log n)$ y $O(n_p \cdot n^2)$, donde n_p es la cantidad de nodos utilizados como pivotes en una búsqueda y $|E| = m = n^{1+\alpha}, \alpha \in [0, 1]$. Esto significa que el costo de cada consulta corresponde aproximadamente al costo de AESA multiplicado por $O(n^\alpha \log n), \alpha \in [0, 1]$. Se recuerda que en las aplicaciones de interés, el costo de computacional de la f.e.d. domina sobre los otros costos de CPU.

3.4. Planificación de las Pruebas

3.4.1. Espacios Métricos de Prueba

Las pruebas realizadas consideran cuatro tipos de espacios métricos: El EM de los vectores con distribución uniforme, el de los strings (cadenas de caracteres), el de los documentos y el de los vectores con distribución gaussiana.

Los espacios vectoriales con distribución uniforme, consideran elementos con componentes reales, generados aleatoriamente según una distribución uniforme en el intervalo $[-1, 1]$, y la distancia euclidiana como f.e.d. Note que en un EM abstracto, no se tiene un mecanismo directo para estimar la dimensión, en cambio, en los espacios vectoriales, se conoce a priori la dimensión representacional de los elementos. Se realizaron pruebas variando los siguientes parámetros: la cantidad de elementos, la dimensión y t. Con estos resultados experimentales se desarrollaron modelos para estimar el tiempo de CPU y la cantidad de arcos necesarios.

Para el caso de los strings, se consideró un subconjunto aleatorio de aproximadamente 23000 palabras de un diccionario inglés y la distancia de edición como f.e.d. La distancia de edición es una función discreta que mide la cantidad mínima de inserciones, borrados y reemplazos de caracteres necesarios de modo que dos strings sean iguales [NR02]. Sólo se dejó libre el parámetro t.

Para el caso de los documentos, se consideró una base de aproximadamente 1200 documentos obtenidos de la colección TREC-3 [Har95]. Para comparar documentos se utilizó la distancia coseno que mide el ángulo formado por dos documentos en el espacio de los términos

[BYRN99], siendo esta f.e.d. extremadamente costosa. Sólo se dejó libre el parámetro t.

Para el caso vectorial con distribución gaussiana, se consideran elementos con componentes reales, generados aleatoriamente según una distribución gaussiana en el intervalo $[-1, 1]$ que consideran tres valores de desviación estándar ($\sigma = 0.1, 0.3\ 0.5$), dos cantidades distintas de centroides (32 y 256) distribuidos uniformemente en el espacio, y la distancia euclidiana. Se fijó la dimensión en 20, y se dejó libre el parámetro t.

3.4.2. Metodología de las Pruebas

La fase experimental de la tesis está dividida por dos etapas:

1. Construcción de t-spanners.
2. Uso de t-spanners para búsqueda en espacios métricos.

Los EMs utilizados en la fase de pruebas tienen distintas características. El primero de ellos, el espacio de los vectores con distribución uniforme, es un espacio sintético, y permite analizar el efecto de la cantidad de elementos de la base de datos, la dimensionalidad de los datos y el factor de tolerancia t. Los espacios de strings y de documentos, pertenecen al mundo real, en donde no se puede acceder directamente a la dimensión del espacio, luego sólo se puede variar el número de elementos de la base de datos, y el factor de tolerancia t. El espacio vectorial con distribución gaussiana pretende simular un EM verdadero, en el sentido de que los EMs obtenidos del mundo real presentan zonas donde se concentran los objetos del espacio.

El principal objetivo de las pruebas realizadas sobre espacios vectoriales con distribución uniforme, es analizar el comportamiento de los algoritmos de construcción de t-spanners. Una vez realizadas la pruebas de construcción en espacios vectoriales, se seleccionará el mejor algoritmo de construcción de t-spanners, vale decir, aquel que ofrezca el mejor compromiso entre tiempo de CPU y uso de memoria.

Una vez seleccionado el mejor algoritmo de construcción de t-spanners, éste será utilizado para construir los índices de búsqueda para realizar los experimentos de recuperación de objetos en espacios métricos considerando la mayor cantidad posible de elementos en la base de datos. En esta etapa, junto con realizar pruebas con espacios vectoriales con distribución uniforme, se utilizarán los espacios de strings, de documentos y vectoriales con distribución gaussiana. Para los espacios métricos de strings, de documentos y vectorial gaussiano, se realizarán pruebas considerando distintos radios de recuperación.

Note que en los cuatro espacios métricos considerados, se realizarán tanto pruebas de construcción como de búsqueda. Sin embargo, en el espacio métrico de los vectores con distribución uniforme, tendrá mayor énfasis la construcción, en cambio, para los espacios de strings, de documentos y vectorial gaussiano, el énfasis se enfocará en la búsqueda.

Para comparar los t-spanners con otras estrategias conocidas, se realizarán experimentos de recuperación de objetos utilizando AESA, y un algoritmo de búsqueda en proximidad basado en pivotes que utiliza la misma memoria que requiere el t-spanner para construir el índice de búsqueda (los pivotes serán seleccionados aleatoriamente). Con esto se pretende mostrar

el comportamiento del t-spanner tanto frente a AESA como a otro algoritmo que también necesita una cantidad reducida de memoria para llevar a cabo búsquedas por similaridad.

Los métodos basados en pivotes resuelven las búsquedas en dos etapas: en la primera, se utiliza el índice construido sobre los pivotes para eliminar la mayor cantidad de candidatos, y en la segunda, mediante la revisión exhaustiva de los candidatos no eliminados, se determina cuál(es) de ellos resuelve(n) la consulta. Debido que en algunos casos se consideran más pivotes de los necesarios para resolver la primera etapa de la búsqueda, se decidió detener la utilización de los pivotes cuando el tamaño del conjunto de candidatos a revisar sea menor que el tamaño del conjunto de pivotes no considerados. De esta manera no se utilizan más e.d. debido al hecho de que se dispone de más pivotes. Adicionalmente, es posible que el número óptimo de pivotes sea menor que el valor asociado al t-spanner más pequeño (esto sucede cuando el problema es "fácil" de resolver), en cuyo caso se muestran los resultados de búsqueda considerando menos pivotes.

3.4.3. Rango de los Parámetros Estudiados

En el EM vectorial uniforme se hicieron experimentos para construcción y reconstrucción de t-spanners y para recuperación de objetos. Los experimentos de construcción se realizaron variando: la cantidad de nodos $|V| \in [200, 8000]$, la dimensión de los vectores $dim \in [4, 28]$ y $t \in [1.3, 2.0]$. Los experimentos de reconstrucción se realizaron con un conjunto inicial de nodos de tamaño $|V| = 2000$ al cual se le insertaban y luego borraban hasta 4000 elementos, para un rango de dimensiones $dim \in [4, 28]$ y $t \in [1.3, 2.0]$. Se escogen estos valores puesto que para EMs de $dim < 4$ existen buenas soluciones, y EMs de $dim \geq 20$ se consideran intratables, luego se pretende explorar algunos valores sobre dimensión 20. Con respecto a t, experimentos preliminares muestran que en $dim = 20$ o superior, la cantidad de arcos del t-spanner es cuadrática para $t = 1.3$. Los experimentos de recuperación, se realizan sobre t-spanners compuestos por 10000 elementos en el rango de dimensiones $[4, 24]$ con valores de $t \in [1.4, 2.0]$. Note que el rango de dimensiones y de t es más limitado, debido a los mayores requerimientos de memoria y tiempo impuestos por la mayor cantidad de objetos.

En los otros EMs utilizados, los experimentos se enfocaban escencialmente en la recuperación de objetos. Para los experimentos en el espacio de los strings se utilizó un diccionario del inglés de aproximadamente 23000 palabras, con valores de $t \in [1.4, 2.0]$. Para los experimentos en el espacio de los documentos se consideró una colección de aproximadamente 1200 documentos, con valores de $t \in [1.4, 2.0]$. Para los experimentos en el espacio vectorial gaussiano se consideró una colección 10000 nodos, con valores de $t \in [1.4, 2.0]$. En los espacios de strings y vectorial gaussiano se considera el $t \geq 1.4$ por razones de tiempo y memoria impuestos por la mayor cantidad de objetos; y en el espacio de documentos, pues con $t < 1.4$ el uso de memoria es cuadrático.

3.4.4. Descripción de la Estación de Trabajo

La máquina utilizada para las pruebas está equipada con un procesador AMD Athlon 1 GHz, 384 MB RAM DDR, sistema operativo Red Hat Linux 7.1, compilador g++ versión 2.96, opción de optimización -O9.

Capítulo 4

Algoritmos para Construcción de t-Spanners

4.1. Algoritmo Básico Optimizado

Este algoritmo, al igual que el Algoritmo Básico (sección 2.2.6), considera el uso de las matrices *real* y *estim* y escoge arcos en orden creciente. La optimización realizada apunta al mecanismo de actualización de las estimaciones de distancia.

La idea es controlar la propagación de cálculos de modo de sólo actualizar las estimaciones de distancias que se ven afectadas tras la inserción de un arco. En la Figura 4.1 se muestra la inserción de un nuevo arco. Como se ve, en la primera actualización sólo se modifica el arco que se inserta, que conecta los vértices marcados con la letra "a". Este cálculo se propaga afectando solamente a lo nodos marcados con "b", y la siguiente propagación llega a los nodos marcados con "c", y por último al nodo "d". Note que no propaga cálculos a los otros nodos. En las actualizaciones tempranas, la no propagación de cálculos se debe principalmente a la falta de conectividad entre los nodos, pero una vez que el grafo es conexo, un nodo propagará los cálculos únicamente en el caso de que mejore las estimaciones de distancia.

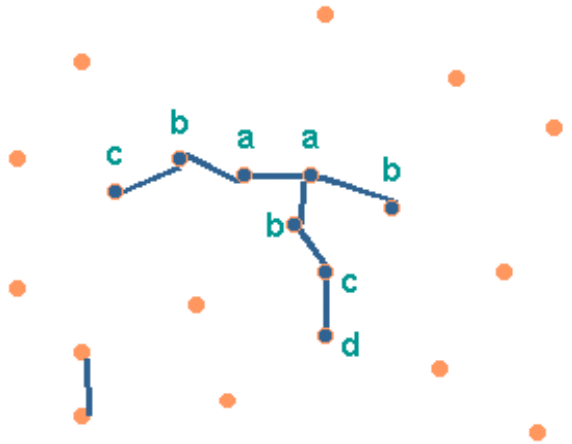


Figura 4.1: Mecanismo de actualización algoritmo t-Spanner 1.

Para llevar a cabo este control de las actualizaciones necesita manejar tres conjuntos de nodos:

- *ok*: Los nodos que ya tienen actualizados sus estimaciones de camino mínimo hacia el arco insertado.
- *check*: Corresponde a la adyacencia de los nodos de *ok*, formalmente esto es: $check = adyacencia(ok)$. Estos son los nodos que se revisan en la iteración actual.
- *adya*: Corresponde a los nodos (que aún no han sido revisados) pertenecientes a la adyacencia de los nodos de *check* que actualizan alguna estimación de distancia. Previo a la definición formal de *adya* considere la definición del conjunto *cambian* como: $cambian = \{u \in check / u$ actualiza alguna estimación$\}$, luego la definición formal de *adya* es: $adya = adyacencia(cambian) - (ok \cup check)$. Estos son los nodos que se revisarán en la iteración siguiente.

Junto al empleo de estos conjuntos, se utiliza el hecho de que sólo se propagan los cálculos cuando el arco insertado mejora la estimación de la distancia a alguno de los vértices que forman el arco. Como se ve en la Figura 4.1 los nodos marcados con "b" propagan cálculos puesto que mejora la estimación contra alguno de los extremos del arco insertado (nodos "a"). En cambio los nodos conectados por el arco solitario no mejoran su estimación de distancia hacia alguno de los nodos "a", luego no participan en la propagación. Esto puede establecer mediante la siguiente condición de propagación:

Ecuación 4.1 (Condición de propagación de cálculos (t-Spanner 1)) *:*

$$\begin{array}{l} \text{Sea } e = (e_u, e_v) \text{ el arco a insertar} \\ \forall\ x \in check, \text{si actualiza } estim(x, e_u) \text{ o } estim(x, e_v) \\ \qquad \Rightarrow \quad \text{propaga los cálculos} \end{array}$$

El ciclo completo revisa todos los arcos del grafo. Para cada arco itera hasta que no se propaguen más cálculos, es decir, hasta que *adya* sea un conjunto vacío. Para controlar el avance de la propagación se actualizan los conjuntos de modo que *ok* guarde a todos los nodos que ya fueron actualizados, *check* guarde los nodos que se tienen que revisar en la iteración, y *adya* guarde los nodos que se revisarán en la iteración próxima. En la Figura 4.2 se muestra el algoritmo básico optimizado. Se agrega el conjunto *tmpCheck* sólo con fines de evitar las revisiones simétricas (si i, $j \in check$ y si ya se revisó el arco (i, j), no tiene sentido revisar el arco (j, i)).

4.2. Análisis del Algoritmo Básico Optimizado

Desde el punto de vista de la cantidad de las e.d., este algoritmo calcula en tiempo de construcción $O(n^2)$ e.d.

Desde el punto de vista del tiempo de procesamiento, este algoritmo inserta m arcos al t-spanner. Considerando que para cada inserción es necesario actualizar a los vecinos directos y

Algoritmo 4.1 (t-Spanner 1) *:*

```
t-Spanner1 (Tolerancia t, Vértices 𝕌)

Inicializaciones:
  real ← matriz de distancias entre todos los vértices
  estim ← matriz en donde se actualizan las estimaciones de costo.
  for (u, v ∈ 𝕌) estim(u, v) ← 0 si u = v, ∞ si no
  t-Spanner ← 0 // estructura de arcos del t-spanner
  Ordena las distancias en real de menor a mayor

Ciclo de actualización:
  for (e = (e_u, e_v) ∈ real, escogido en orden ascendente)
    if (estim(e) > t * real(e)) // e no está bien t-estimado
      t-Spanner ← t-Spanner ∪ {e}
      ok ← {e_u, e_v}
      check ← ∅
      adya ← adyacencia(ok)
      while (adya ≠ ∅) // Propagación de actualizaciones
        // Actualización de los conjuntos
        ok ← ok ∪ check
        check ← adya
        tmpCheck ← adya
        adya ← ∅
        // Chequeo check contra tmpCheck ∪ ok, si actualiza algún
        // costo de un nodo, agregar su adyacencia a adya
        for (c ∈ check)
          tmpCheck ← tmpCheck − {c}
          if ((estim(c, e_v) + real(e) ≤ estim(c, e_u) or
               (estim(c, e_u) + real(e) ≤ estim(c, e_v))
            // propaga los cálculos y guarda la adyacencia
            for (o ∈ ok ∪ tmpCheck)
              d_1 ← estim(c, e_u) + estim(o, e_v)
              d_2 ← estim(c, e_v) + estim(o, e_u)
              estim(c, o) ← mín (estim(c, o), mín(d_1, d_2) + real(e))
            adya ← adya ∪ (adyacencia(c) - (ok ∪ check))
```

Figura 4.2: Algoritmo básico optimizado (t-Spanner 1).

a aquellos nodos en donde se propagan los cálculos que en el peor caso son n^2 actualizaciones, se tiene que el costo en tiempo de procesamiento en promedio es de $O(m \cdot k^2)$, con k la cantidad promedio de vecinos a actualizar, y de $O(mn^2)$ en el peor caso.

Desde el punto de vista de la memoria utilizada, requiere $O(n^2) + O(m)$, ya que necesita almacenar toda la matriz de distancias, más la estructura del t-spanner.

Este algoritmo reduce la cantidad de cálculos requeridos para actualizar, pero el hecho de actualizar para cada arco insertado significa un fuerte sobrecosto. En efecto los inconvenientes de este algoritmo son el alto esfuerzo de cálculo debido a la actualización permanente de arcos, y que requiere mucha memoria. Una ventaja de este algoritmo es que produce t-spanners de buena calidad, motivo por el cual, una vez desarrollados los experimentos de t-Spanner 1, se obtuvo un modelo de la cantidad de arcos contenidos en el t-spanner, y este modelo se utilizó para obtener una estimación a priori de los arcos que se necesitan en un t-spanner. El modelo de la cantidad de arcos se presentará en el capítulo 6 *Resultados Experimentales*.

4.3. Algortimo de Inserción Masiva de Arcos

En la segunda versión se pretende disminuir el sobrecosto de procesamiento actualizando las estimaciones luego de insertar varios arcos (de acuerdo con la estimación indicada por el modelo de arcos obtenido de t-Spanner 1), con lo que se consigue amortizar los costos y a la larga, disminuirlos notablemente. Para verificar la condición de t-spanner, se calculan las distancias utilizado Dijkstra sobre el t-spanner en construcción.

Este algoritmo se desarrolla en tres etapas. En la primera se construye el esqueleto del t-spanner insertando $|V| - 1$ arcos (MSTs completos) en cada oportunidad, en la segunda se refina el t-spanner corrigiendo la mayoría de los arcos que están mal estimados, y en la última se insertan los arcos "difíciles".

Antes de describir en detalle el algoritmo se definen las siguientes heurísticas:

H_1 Determina la cantidad de arcos por nodo y se obtiene a partir del modelo de arcos de t-Spanner 1:

$$H_1 = |E_{t-\text{Spanner }1}(|V|, dim, t)|/|V|$$

A partir del valor de H_1 se definirán umbrales que se utilizarán al momento de decidir si insertar o no los arcos restantes (es decir, aquellos que aún estén mal t-estimados) del nodo que se esté revisando.

H_2 Se utiliza para dimensionar la cantidad de arcos que se almacenarán en la lista de arcos mal t-estimados, que se denominará *pendientes*, y entrega un criterio para determinar cuándo es necesario insertar un MST adicional. Se estableció que la capacidad de la lista de arcos pendientes debe ser igual a H_2:

$$H_2 = 1{,}2 \cdot |E|$$

donde $|E|$ se refiere al tamaño del t-spanner en construcción.

Para el ajuste de estas heurísticas se escogieron valores de modo que no insertaran demasiados arcos, ni que el costo en tiempo aumentara sin obtener t-spanners con una cantidad menor de arcos.

Las etapas de este algoritmo son:

1. Inserción de MST sucesivos. El primer MST sigue el mecanismo usual de Prim [Wei95], los siguientes MSTs se construyen utilizando Prim, pero considerando solamente el conjunto de arcos que aún no han sido insertados al t-spanner.

   Es importante señalar que debido a que la inserción de MSTs es incremental, es posible que en una iteración un arco esté mal t-estimado, y en iteraciones posteriores debido a la inserción de MSTs adicionales, la estimación del arco mejore lo suficiente como para que el arco esté bien t-estimado, aun sin formar parte de los MSTs insertados.

   En esta etapa se usa el valor de H_1 para decidir si la cantidad de arcos pendientes de un nodo es suficientemente pequeña como para justificar la inserción de dichos arcos, y H_2 para determinar el instante en que se tenga que insertar un MST adicional. Se utilizó como criterio que cuando el número de arcos pendientes de un nodo es menor que $H_1/2$ se insertan directamente. Por otro lado, cuando el tamaño de la lista global de pendientes supere el valor de H_2 se inserta otro MST. Note que la verificación de H_2 se hace previo a la revisión de un nodo, en cambio el criterio de $H_1/2$ se hace una vez determinado con cuando arcos pendientes queda el nodo.

   Esta etapa continúa hasta que se hayan revisado todos los nodos. Para para cada nodo se determina que arcos qué inciden en el nodo están mal t-estimados.

   La salida de esta etapa es la lista de arcos insertados al t-spanner (*t-Spanner*), y la lista global de arcos que aún no han sido bien t-estimados (*pendientes*). Ambas listas utilizan la estructura de una lista de adyacencia.

2. En la segunda etapa se procede a eliminar los arcos de la lista *pendientes*. Para esto, genera una lista en donde se guardan los nodos que tienen arcos pendientes y la cantidad de arcos pendientes (*nodosPendientes*), se ordena de mayor a menor de acuerdo a la cantidad de arcos pendientes y se revisan estos nodos en orden descendente verificando qué arcos han mejorado su t-estimación, o cuáles aún siguen mal t-estimados, insertando $H_1/4$ arcos de menor costo a cada nodo (dentro de los que están mal t-estimados). Esta revisión sigue hasta que se inserten $|V|$ arcos[1]. Luego de esto se vuelve a generar la lista de nodos con arcos pendientes y se repite el procedimiento.

   Esto entrega un mecanismo para revisar primero los nodos que necesitan mayor atención (aquellos que tienen más arcos pendientes), y que esto no signifique concentrar todos los esfuerzos en un mismo nodo.

   Dentro del procesamiento de los nodos se consideran dos casos especiales: el primero es que ya se han insertado más de $|V|$ arcos, en cuyo caso se regenera la lista *nodosPendientes* y repite el procedimiento (caso especial 1). El segundo es que la cantidad de arcos pendientes de un nodo sea tan pequeña que simplemente se insertan (caso especial 2).

   La condición de salida de esta etapa es que la cantidad de arcos contenidos en *pendientes* sea menor a $|V|/2$

[1]Note que es probable que luego de revisar por primera vez al nodo con más arcos pendientes, una vez insertados los $H_1/4$ arcos, aún puede tener arcos mal t-estimados.

3. A este nivel, en *pendientes* existe una pequeña cantidad de arcos que aún están mal t-estimados. En el tercera etapa se insertan los arcos que quedaron en *pendientes* luego de la etapa 2.

En la Figura 4.3 se muestra el algoritmo de inserción masiva de arcos.

4.4. Análisis del Algoritmo de Inserción Masiva de Arcos

Desde el punto de vista de la cantidad de las e.d., este algoritmo calcula en tiempo de construcción $O(n^2)$ e.d. por cada MST insertado, luego el costo es de $O(n \cdot m)$ e.d.

Desde el punto de vista del tiempo de procesamiento, el revisar las $O(n^2)$ aristas toma tiempo $O(n \cdot m \log m)$.

Desde el punto de vista de la memoria utilizada, requiere $O(n+m) = O(m)$, ya que necesita almacenar los elementos más la estructura del t-spanner. El costo de memoria $O(m)$ viene del hecho de que la lista *pendientes* nunca es mayor que $O(m)$, pues en cada iteración de la etapa 1, *pendientes* a lo más crece en n, y cuando supera a $1.2 \cdot m$ se extraen n arcos para agregar un nuevo MST, además, en las etapas siguientes se pretende extraer a todos los arcos de la lista *pendientes*.

Este algoritmo disminuye los costos de un modo notable y también disminuye el uso de memoria en tiempo de construcción, pero a cambio de esto tiene el inconveniente de que debido a que inserta conjuntos de arcos, podría insertar arcos de más, y que cada MST gasta $O(n^2)$ e.d., con lo que el proceso completo toma $O(n \cdot m)$ e.d, lo que eventualmente podría ser muy costoso; se podría bajar el costo de e.d. a $O(n^2)$ e.d. (utilizando una matriz con todas las distancias precalculadas) pero esto aumentaría el costo de memoria a $O(n^2)$. A continuación se muestra un algoritmo que sólo utiliza $n(n-1)/2$ e.d. en tiempo de construcción con memoria $O(n+m)$.

4.5. Algoritmo de Inserción Incremental de Nodos

En la tercera versión se pretende disminuir la cantidad de e.d., conservando la idea de amortizar costos en la actualización. Al igual que en el algoritmo t-Spanner 2, las distancias se calculan utilizando Dijkstra.

A diferencia de los dos algoritmos anteriores (que analizan el conjunto completo de objetos y distancias), en esta versión se prueba la alternativa de insertar un nodo a la vez, de modo que en cada iteración el t-spanner en crecimiento satisfaga la t-condición. Esta opción evalúa $n(n-1)/2$ veces la f.e.d., puesto que para el nodo j necesita calcular $j-1$ distancias a los $j-1$ nodos anteriores y la sumatoria $\sum_{j=1}^{n} j-1 = n(n-1)/2$.

En cambio, dado que no considera globalmente los nodos, forzosamente sus resultados son subóptimos en lo que se refiere al número de aristas insertadas al t-spanner.

Algoritmo 4.2 (t-Spanner 2) *:*

```
t-Spanner2 (Tolerancia t, Vértices 𝕌)

Inicializaciones:
  t-Spanner ← MST // estructura de arcos del t-spanner
                  // inicialmente contiene el primer MST
  pendientes ← // lista global de arcos mal t-estimados
  H1 ← |E_{t−Spanner 1}(|V|, dim, t)|/|V| // definición de H1

Etapa 1: Generación del esqueleto de t-Spanner y pendientes
  for (u ∈ 𝕌)
    if (|pendientes| > 1.2 ·|t-Spanner|) // utilizando H2
      t-Spanner ← t-Spanner ∪ MST // utilizando los arcos no insertados
    distancia ← Dijkstra(t-Spanner, u) // distancia(v) = d_{t-Spanner}(u, v)
    for (v ∈ 𝕌)
      if (distancia(v) ≤ t · d(u, v)) pendientes ← pendientes − {(u, v)}
      else pendientes ← pendientes ∪ {(u, v)}
    if (|pendientes(u)| ≤ H1/2)
      t-Spanner ← t-Spanner ∪ pendientes(u)
      pendientes ← pendientes − pendientes(u)

Etapa 2: Eliminación de arcos en pendientes
  while (|pendientes| > n/2)
    nodosPendientes ← nodos ordenados descendentemente según número
      de arcos pendientes
    for (u ∈ nodosPendientes) // seleccionado en orden decreciente
      if (ya se han insertado más n arcos) break // caso especial 1
      if (|pendientes(u)| < H1/4) // caso especial 2
        t-Spanner ← t-Spanner ∪ pendientes(u)
        pendientes ← pendientes − pendientes(u)
      else
        distancia ← Dijkstra(t-Spanner, u)
        for (v ∈ pendientes(u))
          if (distancia(v) ≤ t · d(u, v)) pendientes ← pendientes − {(u, v)}
        livianos ← los H1/4 arcos más livianos ∈ pendientes(u)
        t-Spanner ← t-Spanner ∪ livianos
        pendientes ← pendientes − livianos

Etapa 3: t-Spanner ← t-Spanner ∪ pendientes
```

Figura 4.3: Algoritmo de inserción masiva de arcos (t-Spanner 2).

Para cada nodo, el algoritmo realiza dos operaciones: primero lo conecta al t-spanner utilizando el arco más liviano (o más corto) y luego verifica que las distancias estén bien t-estimadas. De no ser así, inserta una colección de arcos y vuelve a verificar hasta que todas las distancias estén bien t-estimadas. Este proceso sigue hasta que se insertan los n nodos y termina.

En esta etapa también se utiliza la heurística H_1, con la diferencia que debido a que el tamaño del t-spanner aumenta, se recalcula H_1 incrementalmente en cada iteración. Con esto se fija que la cantidad de arcos que se insertan en la i-ésima iteración es $\delta = H_1/(5 \cdot i)$.

Es importante destacar el hecho de que el cálculo de distancias se realiza utilizando Dijkstra en un modo incremental y de propagación limitada. Esto se consigue utilizando una propiedad de Dijkstra, Dijkstra sólo actualiza si el nuevo costo es menor al que ya tiena calculado. Para esto, en el primer cálculo de distancias, Dijkstra recibe un arreglo con las distancias inicializadas con un valor levemente superior al máximo t-permitido: $t \cdot d(v_i, v_j) + \varepsilon$, $\varepsilon > 0$, $j \in [1, i-1]$ (ya que no interesan caminos con costo superior al t-permitido), pero para los siguientes reutiliza el arreglo de distancias obtenido de la iteración anterior, puesto una vez que se agregan los arcos o bien, el costo de camino baja o se conserva.

Para limitar la propagación de cálculos en Dijkstra (ver Algoritmo 2.2, Figura 2.6), se reemplaza la línea † por el nuevo valor de la inicialización para el i-ésimo nodo:

$$\boxed{T[i].\texttt{distancia} \leftarrow t \cdot d(inicio, v_i) + \varepsilon,\ \varepsilon > 0,\ i \in [1, inicio - 1]}$$

Para realizar los cálculos de Dijkstra (ver Algoritmo 2.2, Figura 2.6) de forma incremental, se reemplaza la línea † por el valor que previamente se había calculado para la distancia al i-ésimo nodo:

$$\boxed{T[i].\texttt{distancia} \leftarrow distancias(i)}$$

En la Figura 4.4 se muestra el algoritmo de inserción incremental de nodos.

4.6. Análisis del Algoritmo de Inserción Incremental de Nodos

Desde el punto de vista de la cantidad de las e.d., este algoritmo calcula en tiempo de construcción $n(n-1)/2$ e.d.

Desde el punto de vista del tiempo de procesamiento, el revisar las $O(n^2)$ aristas toma un tiempo $O(n \cdot m \log m)$, ya que utiliza $n \cdot O(1)$ cálculos de Dijkstra (en la versión incremental de propagación limitada).

Desde el punto de vista de la memoria utilizada, requiere $O(n+m)$, ya que necesita almacenar los elementos más la estructura del t-spanner.

Este algoritmo utiliza $n(n-1)/2$ e.d. en tiempo de construcción. Pero debido a que no realiza un análisis global forzosamente tiende a generar t-spanners de una calidad inferior

Algoritmo 4.3 (t-Spanner 3) *:*

```
t-Spanner3 (Tolerancia t, Vértices 𝕌)

Inicializaciones:
  t-Spanner ← ∅ // estructura de arcos del t-spanner

Ciclo de actualización:
  for (i ∈ [1,|V|])
    δ ← |E_{t−Spanner 1}(i, dim, t)|/(5 · i) // H_1 incremental
    k ← argmin_{j∈[1,i−1]}{(nodo_i, nodo_j)}
    t-Spanner ← t-Spanner ∪ {(nodo_i, nodo_k)}
    // definiendo el límite de propagación
    distancias ← {(nodo_j, t · d(nodo_i, nodo_j) + ε), j ∈ [1, i − 1]}
    while(nodo_i aún tiene arcos mal t-estimados)
      distancias ← Dijkstra(t-Spanner, nodo_i, distancias)// Dijkstra incremental
      pendientes_i ← {(nodo_i, nodo_j), j < i / distancias(nodo_j) > t · d(nodo_i, nodo_j)}
      livianos ← el conjunto de los δ arcos más livianos en pendientes_i
      t-Spanner ← t-Spanner ∪ livianos
```

Figura 4.4: Algoritmo de inserción incremental de nodos (t-Spanner 3).

(con una mayor cantidad de arcos). Esto se debe a que la condición de generar t-spanners incrementalmente restringe las posibilidades de solución, con lo que la solución obtenida puede ser subóptima.

A continuación se presentan dos algoritmos que pretenden entregar herramientas que permitan construir t-spanners con un costo de $n(n-1)/2$ e.d. en tiempo de construcción y con memoria $O(n+m)$ con la ventaja que se analizan globalmente los conjuntos de nodos y arcos.

4.7. Algoritmo Recursivo

Para la aplicación de interés, el algoritmo t-Spanner 3 es una buena alternativa de solución (pues utiliza una cantidad razonable de e.d. y su costo de construcción en tiempo también es razonable). Lamentablemente la metodología incremental no considera la cercanía (o lejanía) espacial de los objetos entre sí. Una forma de resolver esto es que el conjunto en que se construye incrementalmente el t-spanner esté compuesto por objetos más cercanos. Para ello la solución natural es dividir el conjunto en dos conjuntos de un diámetro menor, construir el t-spanner en las partes y luego fusionar el resultado. Siguiendo este principio, se llega a una solución que divide recursivamente el conjunto, construye sub-t-spanners en las partes y luego las mezcla de una forma conveniente.

Para la división inicial del conjunto se toman dos objetos lejanos entre sí, p_1 y p_2, que serán llamados *representantes* y luego se generan dos conjuntos: los que están más cerca de p_1

y los que están más cerca de p_2 (ver la Figura 4.5). Para las divisiones recursivas se toma el elemento más lejano al representante del grupo, y se separan según distancias hacia el representante y hacia el más lejano. El caso base se tiene cuando la cantidad de nodos es menor o igual que 2. De esta forma los conjuntos generados resultan de objetos cercanos entre sí.

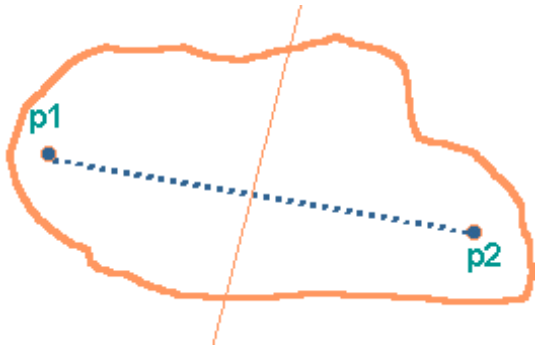


Figura 4.5: Mecanismo de división del conjunto de nodos (t-Spanner 4). Los elementos se separan de acuerdo a la cercanía a los representantes p_1 y p_2.

La metodología de mezclado también toma en cuenta la cercanía espacial de los objetos entre sí. Al momento de mezclar las subsoluciones se tienen dos subconjuntos de vértices V_1 y V_2. Sin perder generalidad se puede suponer que $|V_1| > |V_2|$ (de no ser así se intercambian los subconjuntos). Luego se toma el elemento del sub-t-spanner representado por p_2 ($stsp_2$) más cercano a p_1 (u_{stsp_2}), y se inserta al sub-t-spanner representado por p_1 ($stsp_1$) verificando que todas las distancias hacia los nodos de $stsp_1$ estén bien t-estimadas. Note que esto es equivalente a considerar que se utiliza t-Spanner 3, en donde se inserta el nodo u_{stsp_2} al t-spanner en crecimiento $stsp_1$. Luego se toma el segundo más cercano y se inserta. Este proceso de inserción se repite para todos los nodos de $stsp_2$ tomándolos en orden creciente de distancia contra p_1 (ver la Figura 4.6).

Figura 4.6: Mecanismo de mezclado del conjunto de nodos (t-Spanner 4). Los elementos de $stsp_2$ se insertan al sub-t-spanner mayor, $stsp_1$, de acuerdo a la cercanía al representante p_1.

Note que las aristas de $stsp_2$ no se eliminan, puesto que constituyen un t-spanner bien formado para el conjunto de nodos de $stsp_2$. Esto es beneficioso en la medida de que permite utilizar el trabajo de inserción de aristas ya realizado, y concentrar los esfuerzos en la operación de mezclado de los sub-t-spanners.

En este algoritmo también se utiliza una versión incremental y de propagación limitada de Dijkstra, pero esta vez sólo interesa limitar la propagación hacia los nodos de $stsp_1$ (puesto

que se sabe que hacia los nodos de $stsp_2$ ya se cumple la t-condición), luego el arreglo que recibe se incializa con: $t \cdot d(v_i, v_j) + \varepsilon$, $\varepsilon > 0$, $v_i \in V_2$ si $v_j \in stsp_1$; ∞ si $v_j \in stsp_2$. Para los siguientes cálculos de distancia reutiliza el arreglo de distancias obtenido de la iteración anterior.

La Figura 4.7 muestra el algoritmo recursivo y las funciones auxiliares `makeSubtSpanner(`p`,`V`)` que construye el sub-t-spanner representado por p con los objetos $v \in V$ y `mergeSubtSpanner(`$stsp_1$`,`$stsp_2$`)` que hace el mezclado de sub-t-spanners.

4.8. Análisis del Algoritmo Recursivo

Desde el punto de vista de la cantidad de las e.d., este algoritmo calcula en tiempo de construcción $n(n-1)/2 + O(n \log n)$ e.d., ya que necesita calcular las distancias entre todos los pares de objetos y cálculos de distancia adicionales para la división recursiva de conjuntos. Este análisis supone que los subconjuntos generados tienen similar cantidad de elementos, lo que es correcto en promedio suponiendo una distribución binomial.

Desde el punto de vista del tiempo de procesamiento el análisis es idéntico que para t-Spanner 3: el revisar las $O(n^2)$ aristas toma un tiempo $O(n \cdot m \log m) + O(n \log n)$.

Desde el punto de vista de la memoria utilizada, requiere $O(n+m)$, ya que necesita almacenar los elementos más la estructura del t-spanner.

Este algoritmo utiliza $O(n^2)$ e.d. en tiempo de construcción y debido a que realiza un análisis global de las distancias permite encontrar t-spanner de mejor calidad (es decir, que contienen menos arcos) que el algoritmo t-Spanner 3. Se espera que el tiempo de procesamiento sea mayor que el de t-Spanner 3.

4.9. Algoritmo Recursivo con Caso Base t-Spanner 1

La última versión estudiada de los algoritmos de construcción de t-spanners pretende analizar la posibilidad de considerar un caso base de la recursión que permita una mayor cantidad de elementos. Recuerde que el caso base del algoritmo t-Spanner 4 es dos elementos.

Conjuntamente se pretende analizar la posibilidad de utilizar al algoritmo t-Spanner 1 para resolver los subproblemas no divisibles conservando la técnica de mezclado del algoritmo t-Spanner 4.

La Figura 4.8 muestra el algoritmo recursivo con caso base t-Spanner 1.

4.10. Análisis del Algoritmo Recursivo con Caso Base t-Spanner 1

Este algoritmo se debiera comportar de modo similar a t-Spanner4, tanto para las e.d., el tiempo de procesamiento y el uso de memoria. Con respecto al parámetro *groupSize* no se

Algoritmo 4.4 (t-Spanner 4) *:*

```
t-Spanner4 (Tolerancia t, Vértices 𝕌)

t-Spanner ← 0 // estructura de arcos del t-spanner
(p1, p2) ← Selecciona dos objetos lejanos
(V1, V2) ← Divide 𝕌 según la distancia a (p1, p2)
stsp1 ← makeSubtSpanner(p1, V1)
stsp2 ← makeSubtSpanner(p2, V2)
t-Spanner ← mergeSubtSpanner(stsp1, stsp2)

makeSubtSpanner(representante p, Vértices V)
  if (|V| = 1) return t-spanner (nodos ← {p}, arcos ← ∅)
  else if (|V| = 2) return t-spanner (nodos ← V = {v1, v2}, arcos ← {(v1, v2)})
  else // |V| ≥ 3, dividir y mezclar
    plejano ← argmax_{v∈V} {d(p, v)}
    (V, Vlejano) ← Divide V según la distancia a (p, plejano)
    stsp_p ← makeSubtSpanner(p, V)
    stsp_lejano ← makeSubtSpanner(plejano, Vlejano)
    return mergeSubtSpanner(stsp_p, stsp_lejano)

mergeSubtSpanner(t-Spanner stsp1, t-Spanner stsp2)
  si (|nodos(stsp1)| ≤ |nodos(stsp2)|) intercambia stsp1 con stsp2
  nodos ← nodos(stsp1) ∪ nodos(stsp2)
  arcos ← arcos(stsp1) ∪ arcos(stsp2)
  δ ← |E_{t−Spanner1}(|nodos|, d, t)| / (i · 5) // H1 incremental
  p1 ← representante(stsp1)
  for (u ∈ nodos(stsp2) en orden creciente de d(u, p1))
    // definiendo el límite de propagación hacia stsp1
    for (v ∈ nodos(stsp1)) distancias(v) ← t · d(u, v) + ε
    for (v ∈ nodos(stsp2)) distancias(v) ← ∞
    while (u tiene arcos mal t-estimados hacia stsp1)
      distancias ← Dijkstra(arcos, u, distancias)// Dijkstra incremental
      pendientes_u ← {(u, v), v ∈ stsp1, distancias(v) > t · d(u, v)}
      livianos ← el conjunto de los δ arcos más livianos en pendientes_u
      arcos ← arcos ∪ livianos
  return t-Spanner (nodos ← nodos, arcos ← arcos)
```

Figura 4.7: Algoritmo recursivo (t-Spanner 4).

Algoritmo 4.5 (t-Spanner 5) *:*

```
t-Spanner5 (Tolerancia t, Vértices 𝕌)

t-Spanner ← 0 // estructura de arcos del t-spanner
groupSize ← número máximo de objetos del caso base de la recursión (≥ 3)
(p1, p2) ← Selecciona dos objetos lejanos
(V1, V2) ← Divide 𝕌 según la distancia a (p1, p2)
stsp1 ← makeSubtSpanner(p1, V1)
stsp2 ← makeSubtSpanner(p2, V2)
t-Spanner ← mergeSubtSpanner(stsp1, stsp2)

makeSubtSpanner(representante p, Vértices V)
  if (|V| < groupSize) return t-spanner construido con t-Spanner1
  else // |V| ≥ groupSize, dividir y mezclar
    plejano ← argmax_{v∈V} {d(p, v)}
    (V, Vlejano) ← Divide V según la distancia a (p, plejano)
    stsp_p ← makeSubtSpanner(p, V)
    stsp_lejano ← makeSubtSpanner(plejano, Vlejano)
    return mergeSubtSpanner(stsp_p, stsp_lejano)

mergeSubtSpanner(t-Spanner stsp1, t-Spanner stsp2)
  si (|nodos(stsp1)| ≤ |nodos(stsp2)|) intercambia stsp1 con stsp2
  nodos ← nodos(stsp1) ∪ nodos(stsp2)
  arcos ← arcos(stsp1) ∪ arcos(stsp2)
  δ ← |E_{t−Spanner1}(|nodos|, d, t)| / (i · 5) // H1 incremental
  p1 ← representante(stsp1)
  for (u ∈ nodos(stsp2) en orden creciente de d(u, p1))
    // definiendo el límite de propagación hacia stsp1
    for (v ∈ nodos(stsp1)) distancias(v) ← t · d(u, v) + ε
    for (v ∈ nodos(stsp2)) distancias(v) ← ∞
    while (u tiene arcos mal t-estimados hacia stsp1)
      distancias ← Dijkstra(arcos, u, distancias)// Dijkstra incremental
      pendientes_u ← {(u, v), v ∈ stsp1, distancias(v) > t · d(u, v)}
      livianos ← el conjunto de los δ arcos más livianos en pendientes_u
      arcos ← arcos ∪ livianos
  return t-Spanner (nodos ← nodos, arcos ← arcos)
```

Figura 4.8: Algoritmo recursivo con caso base t-Spanner 1 (t-Spanner 5).

hacen mayores comentarios puesto que si $groupSize \ll n$ su efecto es marginal.

Desde el punto de vista de la cantidad de las e.d., este algoritmo calcula en tiempo de construcción $n(n-1)/2 + O(n \log n)$ e.d., ya que necesita calcular las distancias entre todos los pares de objetos y cálculos de distancia adicionales para la división recursiva de conjuntos. Este análisis también supone que los subconjuntos generados tienen similar cantidad de elementos, lo que es correcto en promedio suponiendo una distribución binomial.

Desde el punto de vista del tiempo de procesamiento, al momento de revisar las $O(n^2)$ aristas toma un tiempo $O(n \cdot m \log m) + O(n \log n)$.

Desde el punto de vista de la memoria utilizada, requiere $O(n+m)$, ya que necesita almacenar los elementos más la estructura del t-spanner.

Este algoritmo utiliza $O(n^2)$ e.d. en tiempo de construcción y debido a que realiza un análisis global de las distancias permite encontrar t-spanner de mejor calidad (es decir, que contienen menos arcos) que el algoritmo t-Spanner 3. Se espera que el tiempo de procesamiento sea mayor que el de t-Spanner 3.

4.11. Análisis Comparativo de los Algoritmos de Construcción de t-Spanners

En el Cuadro 4.1 se muestran las complejidades obtenidas tanto para el algoritmo básico, como para los cinco algoritmos propuestos en esta tesis.

	Básico	Básico optimizado	Inserción masiva de arcos	Incremental	Recursivo	Recursivo caso base t-spanner 1
Tiempo CPU	$O(mn^2)$	$O(mk^2)$	$O(nm \log m)$	$O(nm \log m)$	$O(nm \log m)$	$O(nm \log m)$
Memoria	$O(n^2)$	$O(n^2)$	$O(m)$	$O(m)$	$O(m)$	$O(m)$
Evaluaciones de distancia	$O(n^2)$	$O(n^2)$	$O(nm)$	$O(n^2)$	$O(n^2)$	$O(n^2)$

Cuadro 4.1: Comparación de la complejidad de los algoritmos de construcción de t-Spanner. El valor k se refiere al número de nodos que tienen que ser revisados al actualizar distancias debido a la inserción de un nuevo arco.

Del Cuadro 4.1 se aprecia que para la aplicación de interés, uso de t-spanners para búsqueda en EM, los algoritmos apropiados son el incremental y los recursivos, puesto que son los que tienen la mejor complejidad de CPU, memoria lineal en el tamaño del t-spanner, y requieren $O(n^2)$ e.d. Posteriormente, considerando los resultados empíricos, se verifica que el algoritmo recursivo es el que presenta el mejor compromiso de tiempo de CPU y cantidad de arcos generados.

Capítulo 5

Uso de t-Spanners para Búsqueda en Espacios Métricos

En la sección 1.2 se esboza el objetivo principal de esta tesis, que consiste en: a partir de los objetos del espacio métrico, construir un t-spanner de modo de poder utilizarlo como la estructura de la base de datos que almacene los objetos. A continuación se muestran los algoritmos que otorgan la flexibilidad suficiente al t-spanner, de modo de poder utilizarlo como la base de datos para búsquedas en espacios métricos. Los siguientes mecanismos permiten:

- Insertar objetos al t-spanner.
- Eliminar objetos del t-spanner.
- Mantener la calidad de la estructura.
- Realizar búsquedas en proximidad sobre el t-spanner.

A continuación se describen cada uno de estos mecanismos.

5.1. Algoritmo de Inserción de Objetos al t-Spanner

El problema de insertar un elemento es equivalente al descrito en la sección 4.5. El invariante del algoritmo t-Spanner 3 consiste en que para el k-ésimo nodo, la estructura construída para los $k-1$ nodos anteriores es un t-spanner, luego para el k-ésimo nodo sólo se necesita verificar las distancias entre el elemento k y los $k-1$ anteriores.

En el caso de la inserción a un t-spanner, se tiene la garantía de que el t-spanner está bien construido (es decir, que la t-condición se cumple), luego lo que se necesita hacer es verificar que las distancias entre el nodo a insertar y los nodos del t-spanner cumplan la t-condición.

El la figura 5.1 se muestra el algoritmo de inserción de objetos al t-spanner.

Algoritmo 5.1 (Inserción de objetos al t-Spanner) *:*

```
Insert (t-Spanner t-Spanner, Vértice u)

n ← |nodos(t-Spanner)|
δ ← |E_{t−Spanner 1}(n + 1, dim, t)|/(5 · (n + 1)) // H_1 incremental
k ← argmin_{j∈[1,n]}{(u, nodo_j)}
t-Spanner ← t-Spanner ∪ {(u, nodo_k)}
// definiendo el límite de propagación
distancias ← {(nodo_j, t · d(u, nodo_j) + ε), j ∈ [1, n]}
while(u aún tiene arcos mal t-estimados)
  distancias ← Dijkstra(t-Spanner, u, distancias)// Dijkstra incremental
  pendientes ← {(u, nodo_j), j ≤ n / distancias(nodo_j) > t · d(u, nodo_j)}
  livianos ← el conjunto de los δ arcos más livianos en pendientes
  t-Spanner ← t-Spanner ∪ livianos
```

Figura 5.1: Algoritmo de inserción de objetos al t-Spanner.

Sigue un análisis del algoritmo de inserción de objetos al t-Spanners:

- Desde el punto de vista de la cantidad de las e.d., este algoritmo calcula en tiempo de inserción n e.d.
- Desde el punto de vista del tiempo de procesamiento, revisar las $O(n)$ aristas toma un tiempo $O(m \log m)$ (ver sección 4.6).
- Desde el punto de vista de la memoria utilizada, requiere $O(n + m)$, ya que necesita almacenar los elementos más la estructura del t-spanner.
- A diferencia del algoritmo t-Spanner 3, este algoritmo revisa globalmente las distancias hacia los n nodos del t-spanner, puesto que parte de un t-spanner compuesto por los V nodos y los E arcos.

5.2. Algoritmo de Borrado de Objetos del t-Spanner

La opción considerada para borrar elementos consiste en marcar el elemento como eliminado. Esta opción tiene la ventaja de que toma tiempo $O(1)$ para eliminar al objeto y no se modifica la estructura del t-spanner, pero tiene la deficiencia de que al momento de eliminar no libera la memoria ni del objeto borrado, ni de los arcos entre dicho objeto y el resto de los objetos de la base de datos. Esta opción es impensable en aplicaciones donde cada objeto tenga un tamaño muy grande, por ejemplo en el caso de espacios métricos de documentos o de imágenes.

De ser necesario borrar efectivamente al objeto de la base de datos, es necesario realizar una reconstrucción local del t-spanner en torno al elemento borrado de modo de conservar la condición de t-spanner. Esta reconstrucción consiste en conectar todos los vecinos del

nodo a eliminar (que no estén previamente conectados) entre sí, con arcos marcados como temporales. Note que no todos los arcos temporales son necesarios para conservar la condición de t-spanner. Esto se muestra en la Figura 5.2, en la que se ve el proceso de eliminación del elemento central (nodo 6) utilizando la opción de borrado efectivo del elemento. En la izquierda se muestra la situación inicial y en la derecha la situación final; los arcos dibujados con líneas segmentadas son temporales.

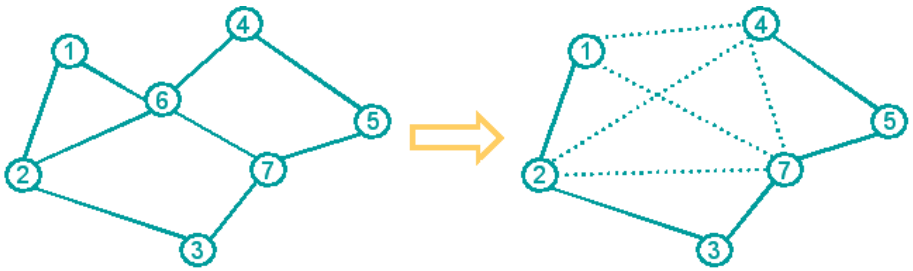


Figura 5.2: Opción de borrado efectivo de objetos al t-spanner.

Luego de borrados sucesivos la calidad del t-spanner se ve afectada, independientemente de la opción de borrado que se escoja. Si se escoge la opción de marcar al elemento como borrado, en algún momento se debe eliminar efectivamente al elemento y sus arcos de la base de datos; si se escoge la opción de borrar efectivamente al objeto, el t-spanner resultante puede contener más arcos de los necesarios. Esto motiva la necesidad de un algoritmo de reconstrucción que se aplique periódicamene para mantener la calidad de la estructura.

5.3. Algoritmo de Reconstrucción de t-Spanners

La reconstrucción de un t-spanner es equivalente a considerar que se está construyendo un t-spanner con una selección de arcos ya existente. Luego la semilla del t-spanner es $G' = G'(V, E_0)$ (distinto a $G' = G'(V, \emptyset)$, ver sección 2.2.5), donde E_0 es el conjunto de aristas luego de borrar las temporales o las incidentes a nodos borrados. Para conseguir el t-spanner a partir de esta selección de arcos, se insertan otros arcos hasta cumplir la t condición para todas las distancias entre pares de objetos.

Esta estrategia de reconstrucción permite, junto con reutilizar el trabajo previo, realizar una eliminación efectiva de los nodos marcados como borrados y los arcos de los nodos borrados o de los arcos temporales. Luego, antes de reconstruir el t-spanner, se tiene la posibilidad de destruir efectivamente los objetos marcados como borrados, y la lista de arcos que tienen estos objetos; o de eliminar los arcos temporales.

Con esto el algoritmo pasa por dos etapas:

- Eliminación de nodos y arcos marcados como borrados o de los arcos temporales.
- Construcción del t-spanner a partir del conjunto de nodos y arcos que quedaron luego de la eliminación.

Para la segunda etapa se utiliza el algoritmo t-Spanner 4, con la salvedad que en vez de iniciar con una lista vacía de arcos, inicia con la lista resultante luego de la eliminación.

El la figura 5.3 se muestra el algoritmo de reconstrucción de t-spanners.

Algoritmo 5.2 (Reconstrucción de t-Spanners) *:*

```
Reconstrucción(t-Spanner t-Spanner)

Inicializaciones:
  vertices ← nodos(t-Spanner)
  arcos ← arcos(t-Spanner)

Eliminación de los datos temporales dependiendo de la opción de borrado:

  (a) Eliminación de nodos y arcos marcados como borrados:
      for (nodo ∈ vertices)
        if (nodo está marcado como borrado)
          vertices ← vertices − {nodo}
          arcos ← arcos − {(nodo, u) ∈ arcos / u ∈ vertices}

  (b) Eliminación de arcos temporales:
      for (i ∈ [1, |V|])
        for (j ∈ vecinos(i))
          if (arco (i, j) está marcado como temporal)
            arcos ← arcos − {(i, j)}

Ejecuta t-Spanner4, con semilla t-Spanner = (nodos, arcos)
```

Figura 5.3: Algoritmo de reconstrucción de t-Spanner.

5.4. AESA Simulado sobre el t-Spanner

Para resolver una consulta por rango, el algoritmo AESA se basa en la eliminación sucesiva de objetos de la base de datos. De una iteración a la siguiente sólo se conservan los elementos que cumplen la condición 2.1 (sección 2.1.4).

En principio, la diferencia entre AESA tradicional y AESA simulado sobre el t-spanner, es que al descartar elementos es necesario considerar el factor de estimación t. Esto significa que el borde exterior del cascarón de exclusión de AESA se "extiende", con lo cual se pierde en capacidad de discriminación.

La extensión del cascarón de exclusión obedece a la propiedad de aproximación de las distancias (ver la ecuación 2.3, sección 2.2.4). Esto obliga a relajar la condición de exclusión, considerando la condición de exclusión 5.1, extendida para el caso de los t-spanners. Luego los objetos que no cumplan esta condición se eliminan del conjunto de solución de la consulta.

Ecuación 5.1 (Condición de exclusión extendida para t-Spanners) *:*

$$\text{Sea } p \in \mathbb{U} \text{ y } d_{pq} = d(p,q)$$

$$d(u,q) \leq r \quad \Rightarrow \quad \begin{cases} d_{t-\text{Spanner}}(p,u) \geq d_{pq} - r \\ d_{t-\text{Spanner}}(p,u) \leq t \cdot (d_{pq} + r) \end{cases}$$

Adicionalmente, es necesario modificar el criterio de selección de pivotes, puesto que ahora debe considerar que sólo se tiene una t-estimación de las distancias. Para compensar el efecto de t, se define α_t y se reescribe el criterio para la selección pivotes; esto se muestra en la ecuación 5.2.

Ecuación 5.2 (Criterio de selección de pivotes en AESA simulado) *:*

$$\text{Sea } \alpha_t = \frac{2/t+1}{3} \quad \Rightarrow \quad sumLB'(u) = \sum_{i=0}^{k-1} \left| d(p_i,q) - d_{t-\text{Spanner}}(p_i,u) \cdot \alpha_t \right|$$

$$\text{Luego el pivote } p_k \leftarrow \text{argmin}_{u \,\in\, \texttt{candidatos}} \Big\{ sumLB'(u) \Big\}$$

Una vez considerado el efecto de t, se puede reescribir AESA simulado sobre el t-spanner. Se excluirán aquellos objetos que no cumplan la condición 5.1, y la selección del nuevo pivote se utilizará el criterio de minimizar $sumLB'$. Para obtener la distancia entre el pivote y el resto de los elementos se necesita calcular la estimación de la distancia sobre el t-spanner. Esto se resuelve con Dijkstra, pero dado que sólo interesan aquellos objetos dentro del cascarón extendido, los cálculos se propagan sólo hasta sobrepasar $t \cdot (d(pivote,q) + r)$. En la Figura 5.4 se muestra un ejemplo de una consulta con AESA simulado sobre el t-Spanner.

En la Figura 5.4 (a) se muestra la estructura de la base de datos. En (b) se aprecia en azul los elementos dentro del cascarón estricto de AESA y en verde los elementos que están en el cascarón t-extendido; el conjunto de elementos formado por ambos grupos clasifica para la siguiente iteración. En la siguiente iteración no se realiza ninguna distinción entre los elementos que están en el cascarón estricto o en el extendido. En (c) se muestra la segunda eliminación; note que aun cuando ya se han descartado algunos elementos en la búsquda, se consideran todos los arcos del t-spanner, puesto que son necesarios al momento de realizar el cálculo de los caminos mínimos, lo cual se utiliza para estimar las distancias entre los objetos. En (d) se muestra la siguiente iteración de la búsqueda. Note que, en general, en todas las iteraciones quedan elementos en el cascarón extendido, y esto sucede puesto que no se tiene información suficiente para poder eliminarlos. El algoritmo finaliza cuando se encuentran todos los elementos que satisfacen la consulta por rango, o se han descartado todos los elementos.

En la Figura 5.5 se muestra el algoritmo AESA simulado sobre el t-Spanner para resolver la consulta (q,r) (que es la consulta tipo $c1$, ver sección 2.1.1).

Sólo se considera probar consultas del tipo $c1$, puesto que las otras consultas en espacios métricos se pueden resolver a partir de esta consulta.

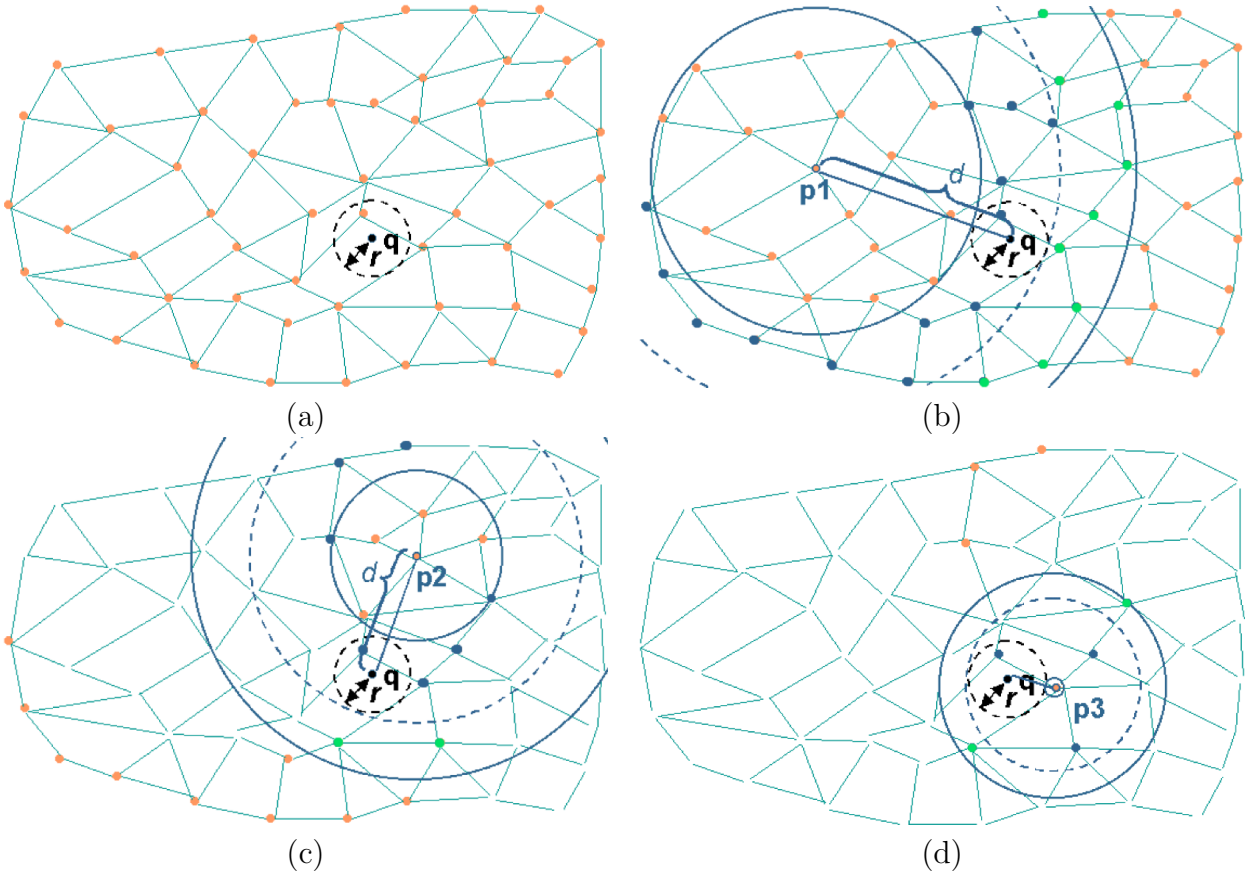


Figura 5.4: Ejemplo de AESA simulado sobre el t-Spanner.

Algoritmo 5.3 (AESA simulado sobre el t-Spanner) *:*

```
t-AESA (Query q, Radio r, t-Spanner t-Spanner)

Inicializaciones:
  vertices ← nodos(t-Spanner)
  arcos ← arcos(t-Spanner)
  candidatos ← vertices
  encontrados ← ∅
  α_t ← (2/t + 1)/3
  for (c ∈ candidatos) sumLB'(c) ← 0

Ciclo de eliminación de candidatos:
  while (candidatos ≠ ∅)
    p ← argmin_{c∈candidatos}{sumLB'(c)}
    candidatos ← candidatos − {p}
    distPQ ← d(p, q)
    if (distPQ ≤ r) encontrados ← encontrados ∪ {p}
    distancias ← Dijkstra(t-Spanner, p, t(distPQ + r) + ε)
    for (c ∈ candidatos)
      if (distancias(c) ∉ [distPQ − r, t(distPQ + r)])
        candidatos ← candidatos − {c}
      else
        sumLB'(c) ← sumLB'(c) + |distPQ − distancias(c) · α_t|
```

Figura 5.5: Algoritmo AESA simulado sobre el t-Spanner.

Capítulo 6

Resultados Experimentales

6.1. Espacio Métrico $\mathbb{R}^D$ con Distancia Euclidiana

La primera serie de experimentos se realizó sobre un conjunto sintético de puntos escogidos al azar en un espacio D-dimensional. No se utilizó el hecho de que el espacio tiene coordenadas, sino que se empleó como si los puntos fueran objetos abstractos de un espacio métrico desconocido. Esta opción permite medir el rendimiento de los algoritmos en función de la dimensión del espacio.

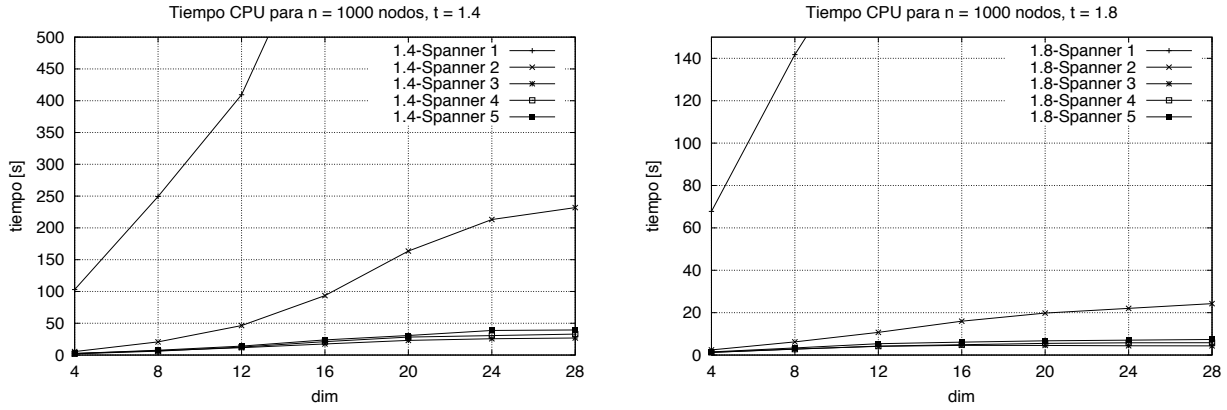
6.1.1. Construcción

Analizando los resultados del Cuadro 4.1, es de esperar que el algoritmo básico optimizado tome el mayor tiempo de ejecución. En las primeras pruebas se pretende hacer una comparación de los cinco algoritmos de construcción, verificar cuán costoso es el algoritmo básico optimizado, y verificar la calidad de los t-spanners que genera. En la Figura 6.1 se muestra el tiempo de CPU necesario por cada algoritmo, y en la Figura 6.2, la cantidad de arcos generados. Ambos experimentos se realizaron con $n = 1000$ nodos.

Como se ve, el algoritmo básico optimizado tiene un costo en tiempo de CPU altísimo (ver Figura 6.1), pero tiene la ventaja de producir los t-spanners de mejor calidad (ver Figura 6.2), es decir, con la menor cantidad de arcos para todo el rango de dimensiones revisado. Debido a que este algoritmo genera t-spanners de buena calidad, se utiliza para generar un modelo de arcos que permite predecir la cantidad de arcos necesarios en la construcción; este modelo de arcos lo utilizan los restantes algoritmos de producción de t-spanners. En adelante, sólo se mostrarán resultados con los otros algoritmos de construcción.

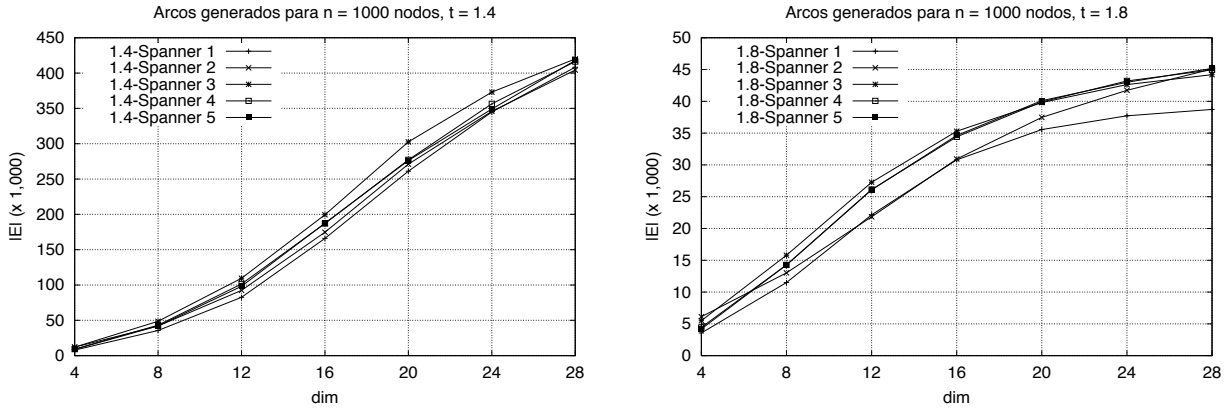
El segundo lugar en calidad de t-spanners depende de la dimensión considerada; para dimensiones bajas lo tiene el algoritmo recursivo con caso base t-spanner 1 (ver Figura 6.6), y para dimensiones altas, lo tiene el algoritmo de inserción masiva de arcos (ver Figura 6.4). Sin embargo, para dimensiones altas la diferencia en la calidad de los t-spanners se hace cada vez menos notoria.

Es interesante destacar que, para valores de $t > 1.65$, se tienen buenos resultados tanto para



(a) (b)

Figura 6.1: Comparación incial de tiempos para los cinco algoritmos. La cantidad de elementos es $n = 1000$. En (a) t-Spanner1 alcanza 2500 segundos en $dim = 28$. En (b) t-Spanner1 alcanza 250 segundos en $dim = 28$.



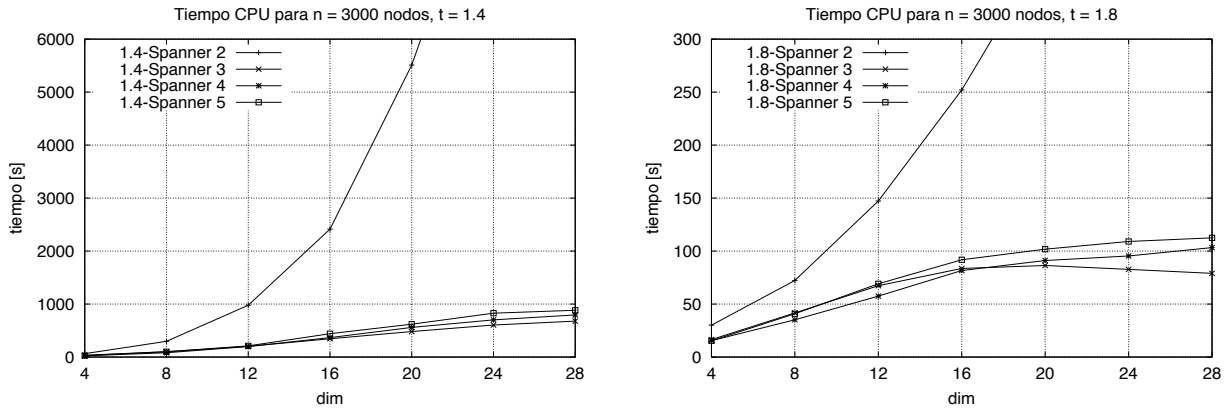
(a) (b)

Figura 6.2: Comparación incial de arcos generados para los cinco algoritmos. La cantidad de elementos es $n = 1000$.

tiempo de procesamiento como para arcos generados (ver Figuras 6.7 y 6.8). En particular para $t = 1.8$, para todo el rango de dimensiones se necesitó menos de un 10 % de los arcos del grafo completo para construir el t-spanner (ver Figuras 6.2 (b) y 6.4 (b)).

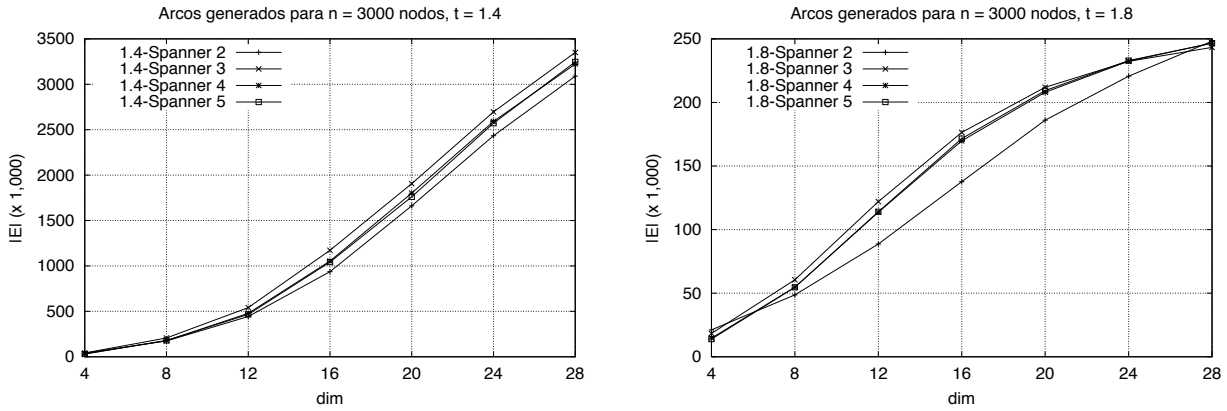
El algoritmo de construcción más rápido, para todo el rango de dimensiones, es el algoritmo incremental (ver Figura 6.3), pero es el que produce t-spanners con la peor calidad (ver Figura 6.4); esto último era esperable, pues la metodología incremental analiza localmente la inserción de arcos. Con los algoritmos recursivos, se consigue corregir en gran medida la calidad de la metodología incremental, consiguiendo los beneficios de utilizar un análisis global al insertar los arcos, llegando en algunos casos a ser competitiva con el algoritmo básico optimizado, con la ventaja de que los tiempos obtenidos son mucho mejores (con mejoras medidas del orden de 100 veces).

Para dimensiones bajas, salvo para el algoritmo básico optimizado, se observa que los algoritmos toman un tiempo cuadrático (ver Figura 6.5), en cambio, el uso de arcos es



(a) (b)

Figura 6.3: Tiempo de CPU para $n = 3000$ variando la dimensión. En (a) t-Spanner2 alcanza 14000 segundos en $dim = 28$. En (b) t-Spanner2 alcanza 600 segundos en $dim = 28$.



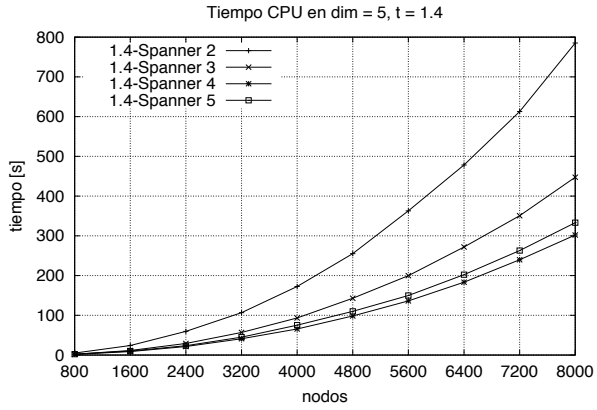
(a) (b)

Figura 6.4: Arcos generados para $n = 3000$ variando la dimensión.

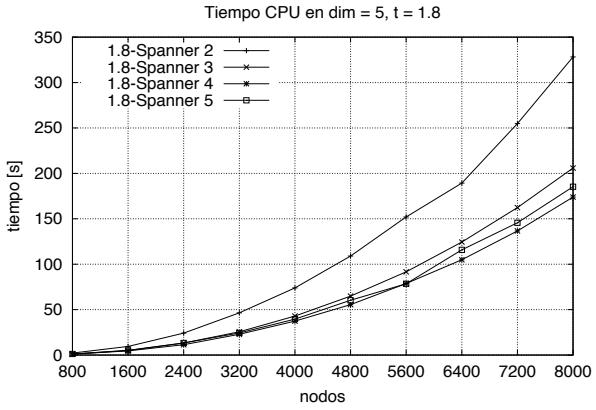
levemente superior al lineal (ver Figura 6.6).

El parámetro que tiene mayor relevancia tanto para el costo en tiempo de CPU como el de memoria, es la cantidad de nodos, lo cual no sorprende, pues es el parámetro que admite mayor variabilidad. Lo que sí es interesante es que la acción conjunta de valores de t pequeños ($t < 1.5$) con dimensiones altas ($dim > 16$) provoca una explosión en los costos de tiempo de CPU y cantidad de arcos generados (ver Figuras 6.7 y 6.8).

Considerando los resultados de los experimentos de construcción de t-spanners, se seleccionó el algoritmo recursivo como el mejor algoritmo para la construcción de la base de datos métrica, puesto que presenta las mejores características de uso de recursos (tiempo de CPU, uso de memoria y número de evaluaciones de distancia) bajo el rango de parámetros estudiados, vale decir, la cantidad de nodos, la dimensionalidad del espacio y el valor de t. Para los próximos experimentos, tanto los de búsqueda en espacios métricos vectoriales como los de construcción y búsqueda en espacios métricos de strings y documentos, se utilizará el algoritmo recursivo para construir la estructura de la base de datos métrica.

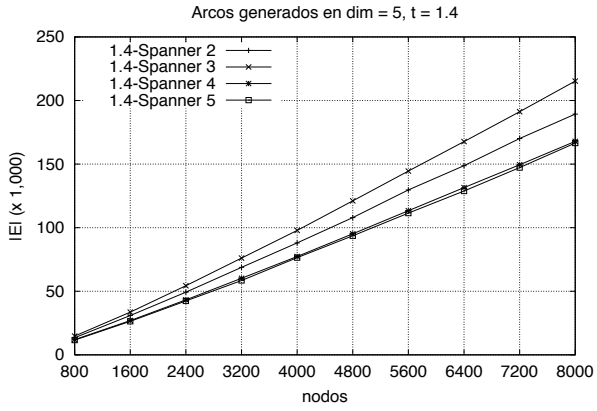


(a)

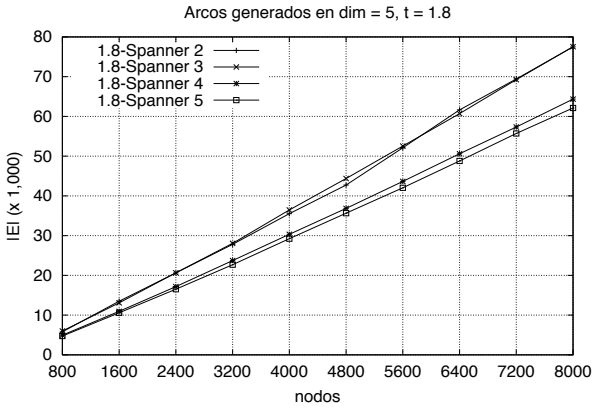


(b)

Figura 6.5: Tiempo de CPU para $dim = 5$ variando la cantidad de nodos.



(a)



(b)

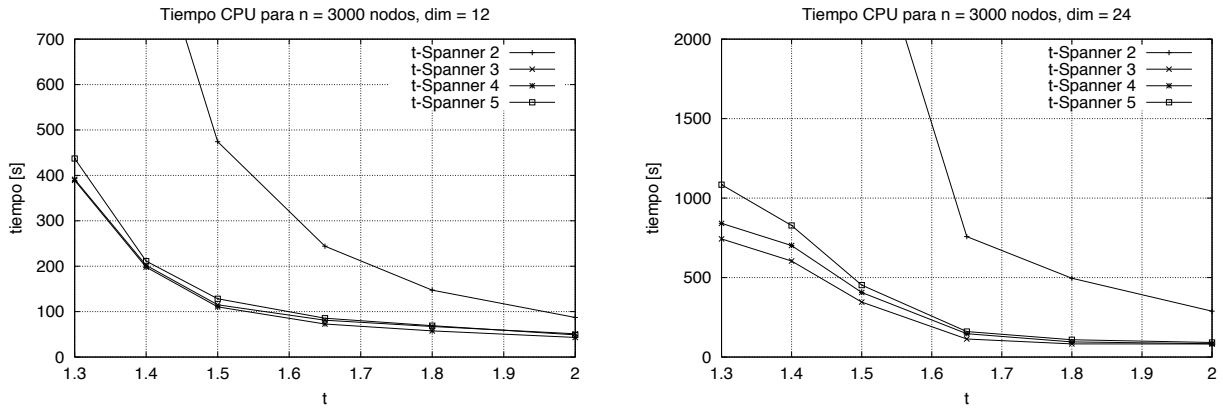
Figura 6.6: Arcos generados para $dim = 5$ variando la cantidad de nodos.

6.1.2. Reconstrucción

En las Figuras 6.9 (a) y (b) se muestran los resultados de los experimentos de reconstrucción.

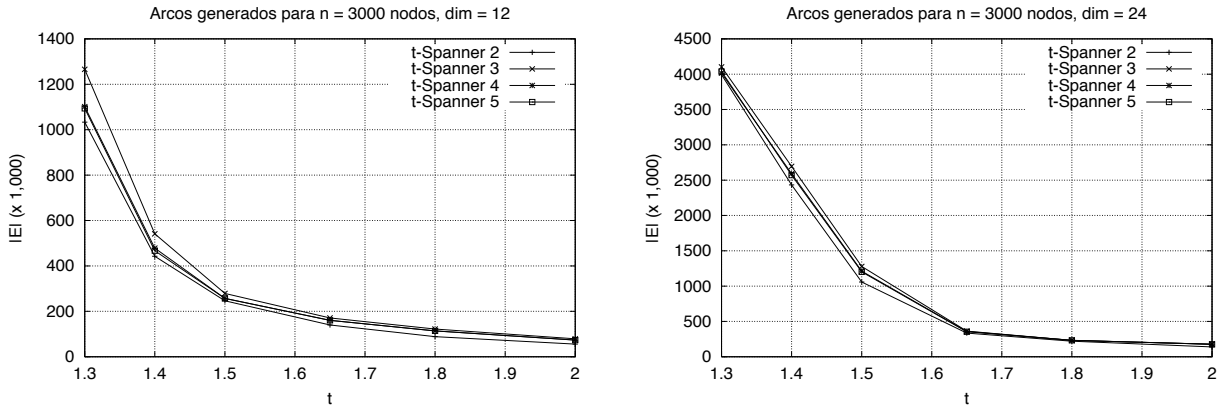
En el gráfico de la Figura 6.9 (a) se muestra el cuociente entre el tiempo de reconstrucción y el tiempo de construcción de un t-spanner con la misma cantidad de nodos. En el gráfico de la Figura 6.9 (b) se muestra el cuociente entre los arcos generados en la reconstrucción y en la construcción para los mismos t-spanners anteriores. Para los experimentos de reconstrucción se considera un conjunto inicial de 2000 nodos con componentes aleatorias dentro de un espacio D-dimensional, y luego se insertan elementos hasta llegar a la cantidad de $maxnodos$ elementos, con $maxnodos \in [2800, 6000]$. Luego de esto, se eliminan $maxnodos - 2000$ nodos al azar para volver al tamaño original de 2000 nodos. Note que es posible que al reconstruir el t-spanner existan nodos que pertenecían al t-spanner original y otros que no.

Considerando el tiempo de procesamiento (ver Figura 6.9 (a)), se observan buenos resultados para valores de t grandes ($t > 1.65$), en especial cuando el t es grande, y la dimensión es pequeña o mediana ($dim \leq 12$).



(a) (b)

Figura 6.7: Tiempo de CPU para $n = 3000$ variando t. En (a) t-Spanner2 alcanza 2800 segundos en $t = 1.3$. En (b) t-Spanner2 alcanza 18600 segundos en $t = 1.3$.



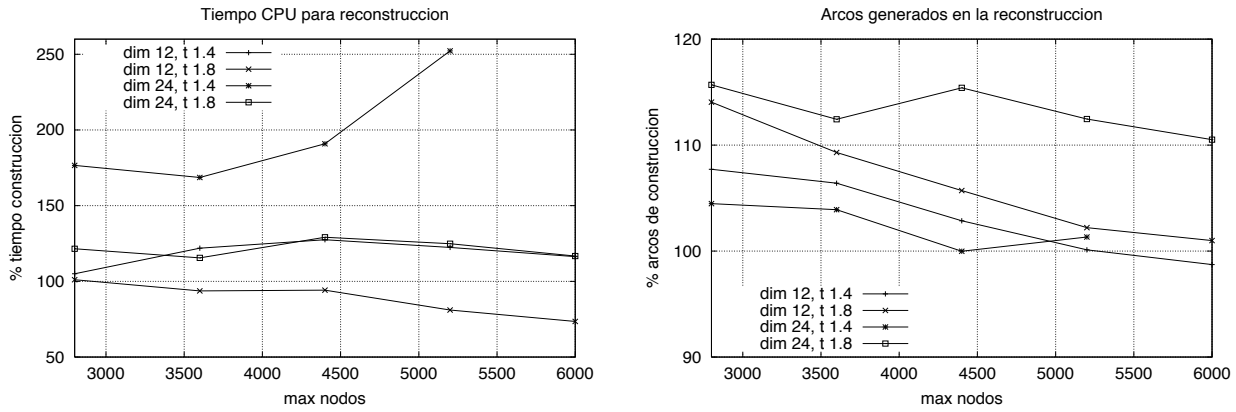
(a) (b)

Figura 6.8: Arcos generados para $n = 3000$ variando t.

Considerando la cantidad de arcos generados (ver Figura 6.9 (b)), se observan buenos resultados con dimensiones bajas y valores de t pequeños.

Consolidando ambos resultados se obtiene la conclusión de que para dimensiones bajas es conveniente aplicar el algoritmo de reconstrucción, en cambio, para dimensiones altas es mejor regenerar la estructura completamente. Para entender este hecho es necesario recordar que cuando se construye por primera vez el t-spanner, al utilizar el algoritmo recursivo, los cálculos de Dijkstra sólo se propagan dentro de los sub-t-spanners, y que cuando se mezclan dos sub-t-spanners, los cálculos sólo se propagan entre las partes que se están mezclando. Cuando se utiliza el algoritmo recursivo para reconstruir el t-spanner, los cálculos de distancia se pueden propagar fuera de los sub-t-spanners que se están mezclando[1], luego las operaciones de mezclado eventualmente pueden involucrar a todos los nodos. Por otro lado, se tiene que a mayor dimensión los t-spanners generados son más densos, es decir, poseen más arcos.

[1]Recuerde que se reconstruye sobre una selección inicial de arcos, y esta puede formar un grafo conexo compuesto por todos los nodos.



(a) (b)

Figura 6.9: Reconstrucción de t-spanners en espacios vectoriales.

Esto conduce al hecho de que al eliminar los datos temporales en dimensiones bajas, la lista inicial de arcos sea más pequeña, luego no genera un sobrecosto considerable al momento de mezclar, en cambio en dimensiones altas, la lista inicial de arcos contiene muchos arcos, luego genera un sobrecosto considerable, pues propaga cálculos fuera de los sub-t-spanners que se están mezclando.

6.1.3. Búsqueda

En la Figura 6.10 se muestran los resultados de los experimentos de recuperación de objetos pertenecientes a espacios métricos vectoriales ($\mathbb{R}^D$, con distancia euclidiana).

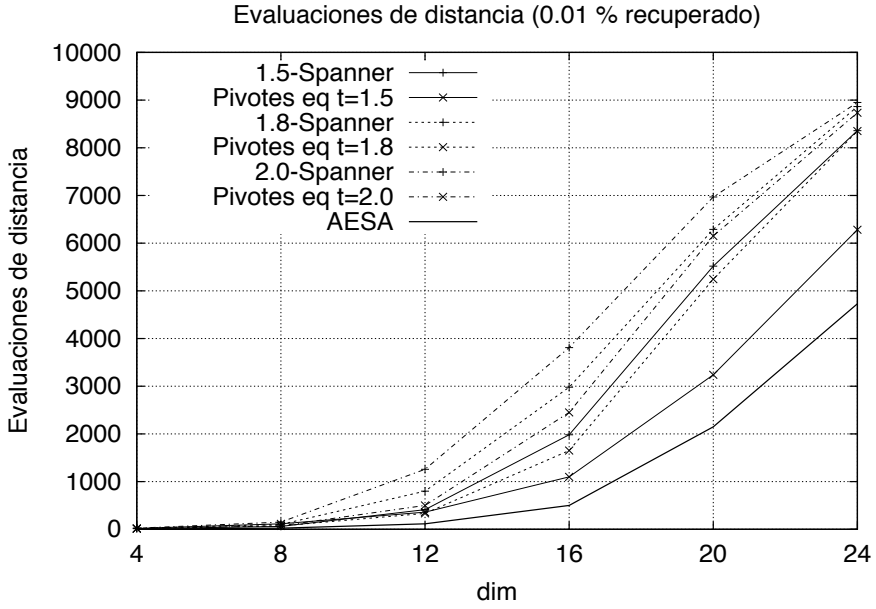


Figura 6.10: Consultas por rango en espacios vectoriales.

La estructura del t-spanner se construyó sobre la base de un conjunto de $n = 10000$ nodos, distribuidos uniformemente en el espacio, el parámetro $D = dim$ toma valores en el intervalo [4,28] y $t \in [1.4, 2.0]$.

En la Figura 6.10, se muestra la comparación del rendimiento del algoritmo AESA simulado sobre el t-spanner para tres valores de $t = 1.5$, 1.8 y 2.0. Se observa que, a medida que el valor de t desciende, la cantidad de evaluaciones de distancia requeridas se hace menor. La línea gruesa representa el comportamiento del algoritmo AESA para las distintas dimensiones, y se observa que AESA requiere muchas menos evaluaciones de distancia, con la desventaja que requiere de la matriz completa de distancias. Adicionalmente se muestra el comportamiento del algoritmo básico basado en pivotes, que utiliza la misma memoria que el t-spanner para almacenar la estructura del índice de búsquedas, para cada valor de t.

Se observa que, para este espacio métrico, el algoritmo de pivotes equivalente tiene un mejor desempeño para cada valor de t; esto se puede explicar considerando que el factor t hace perder capacidad de discriminación. La pérdida en la capacidad de discriminación opera negativamente a medida que la dimensión del espacio aumenta, puesto que a mayor dimensión, el histograma de distancias se concentra haciendo más parecidas las distancias entre los objetos. Una de las ventajas de AESA es que, a medida que el algoritmo itera, escoge pivotes con mayor promisoriedad (ver ecuación 2.2, sección 2.1.5), lo que significa que la distancia entre el pivote y el objeto de consulta tiende a disminuir, lo que debiera excluir rápidamente a los objetos que están fuera del cascarón de exclusión. Una deficiencia del algoritmo AESA simulado sobre el t-spanner es que, para distancias muy pequeñas, es muy posible que el t-spanner conserve estos arcos "livianos", valiendo en este caso un factor de aproximación local $t' \leq t$ que se desaprovecha. En el límite (que ocurre cuando se utiliza el arco verdadero y no una aproximación sobre el t-spanner), se tiene que $t' = 1$. Pero dado que el t-spanner sólo garantiza la t-aproximación de las distancias (ver ecuación 2.3, sección 2.2.4), se debe considerar el cascarón extendido, con lo cual estas últimas iteraciones eliminan muy pocos candidatos.

En las secciones 6.3 y 6.4, se verá que esta situación desfavorable se revierte en espacios de la vida real.

6.2. Modelos Empíricos para Tiempo de Procesamiento y Memoria

Analizando los resultados de las pruebas de construcción de t-spanners en espacios vectoriales bajo la distancia euclidiana, se obtuvieron modelos empíricos para tiempo de CPU y uso de memoria. Note que en términos de evaluaciones de distancia, los algoritmos de construcción presentados (salvo el de inserción masiva) tienen un costo de aproximadamente $n(n-1)/2$ e.d., luego no tiene sentido obtener los modelos empíricos para la cantidad de evaluaciones de distancia.

Dentro de los trabajos relacionados al tema de t-spanners, en [ADDJ90, ADD$^+$93] se demuestra que la cantidad de arcos producidos por el algoritmo básico sigue la forma $n^{1+O(\frac{1}{t-1})}$, donde la dimensión se esconde en los términos del exponente, por lo cual se

intentarán aproximaciones de esta forma. Por otro lado se consideran modelos de ajuste de la forma $k \cdot dim^{a+b\frac{\ln t}{\ln dim}} \cdot n^c$, que fueron los mejores entre todas las familias generales de aproximacin consideradas. De estos dos modelos, se mostrará aquel que tenga mejor correlación.

En las Ecuaciones 6.1 a 6.5 se muestran los modelos de tiempo de CPU y uso de memoria de los cinco algoritmos de producción de t-spanners. Estos modelos tienen validez para $dim \leq 20$ y $t > 1.4$, ya que como se dijo en la sección 6.1.1, la acción conjunta de dim altas y t pequeños tiene el efecto de hacer cuadrático el uso de arcos.

Ecuación 6.1 (Tiempo de CPU y Memoria para t-Spanner 1) *:*

$$tpo_{t-\text{Spanner1}} = 1.3 \cdot 10^{-7}\ dim^{\frac{5.0}{t^{3.5}} + 3.1\frac{\ln t}{\ln dim}}\ n^{3.1\, t^{0.095}}$$

$$arcos_{t-\text{Spanner1}} = n^{1+\frac{1}{5.6dim^{-0.44}t + 0.031dim - 1.5}}$$

Ecuación 6.2 (Tiempo de CPU y Memoria para t-Spanner 2) *:*

$$tpo_{t-\text{Spanner2}} = 1.2 \cdot 10^{-6}\ dim^{\frac{5.3}{t^{2.0}} + 3.1\frac{\ln t}{\ln dim}}\ n^{2.2}$$

$$arcos_{t-\text{Spanner2}} = n^{1+\frac{1}{4.6dim^{-0.36}t + 0.017dim - 1.3}}$$

Ecuación 6.3 (Tiempo de CPU y Memoria para t-Spanner 3) *:*

$$tpo_{t-\text{Spanner3}} = 2.7 \cdot 10^{-6}\ dim^{\frac{2.6}{t^{1.6}} + 0.74\frac{\ln t}{\ln dim}}\ n^{\frac{2.3}{t^{0.020}}}$$

$$arcos_{t-\text{Spanner3}} = n^{1+\frac{1}{3.5dim^{-0.31}t + 0.0081dim - 0.83}}$$

Ecuación 6.4 (Tiempo de CPU y Memoria para t-Spanner 4) *:*

$$tpo_{t-\text{Spanner4}} = 9.3 \cdot 10^{-7}\ dim^{\frac{4.5}{t^{2.5}} + 3.2\frac{\ln t}{\ln dim}}\ n^{2.2\, t^{0.027}}$$

$$arcos_{t-\text{Spanner5}} = n^{1+\frac{1}{3.9dim^{-0.35}t + 0.010dim - 0.86}}$$

Ecuación 6.5 (Tiempo de CPU y Memoria para t-Spanner 5) *:*

$$tpo_{t-\text{Spanner5}} = 2.7 \cdot 10^{-6}\ dim^{\frac{3.4}{t^{1.9}} + 2.2\frac{\ln t}{\ln dim}}\ n^{2.2}$$

$$arcos_{t-\text{Spanner5}} = n^{1+\frac{1}{4.1dim^{-0.36}t + 0.012dim - 0.93}}$$

De los modelos de tiempo de CPU se aprecia que salvo el algoritmo básico optimizado, que tiene un costo cúbico en n, los otros algoritmos tienen costo levemente supercuadrático, alrededor de $O(n^{2.2})$. También se aprecia que la acción conjunta de valores pequeños de t con valores grandes de dim aumentan los costos en tiempo, y que a medida que t decrece, la cantidad de arcos generados aumenta.

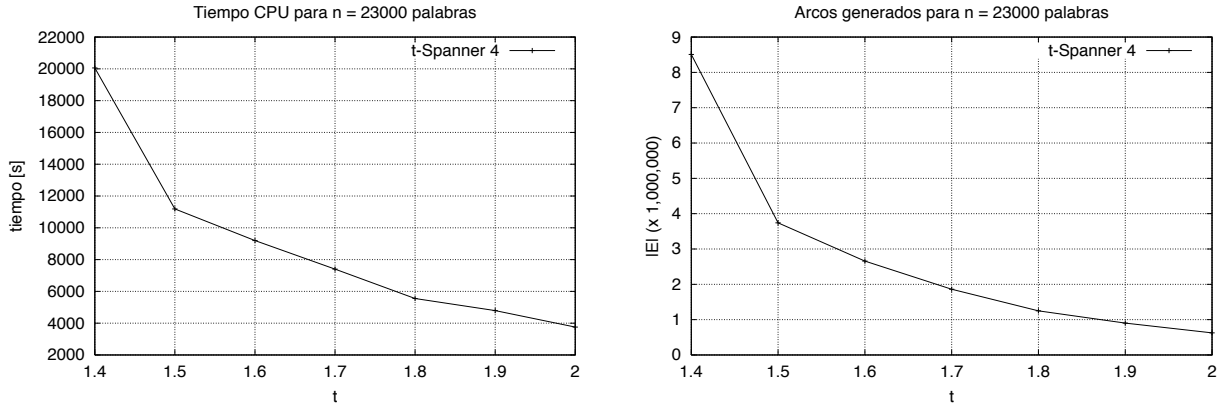
6.3. Espacio Métrico de los Strings con Distancia de Edición

Una de las características del espacio métrico de los strings es que no posee coordenadas, luego se ajusta mejor que los vectores a la noción de espacio métrico abstracto. Se determinó experimentalmente que la dimensión del espacio de strings es $dim = 8$, lo que corresponde a un espacio de dimensionalidad media [CN01].

En este espacio métrico se realizaron pruebas de búsqueda variando el radio de tolerancia, para valores de $t \in [1.4, 2.0]$. Se consideró un conjunto de aproximadamente 23000 palabras de un diccionario inglés.

6.3.1. Construcción

Como se mencionó en la sección 6.1.1, se utilizó el algoritmo recursivo para la construcción de los t-spanners. Los resultados de la construcción, tanto en tiempo de CPU como en arcos generados, se ven en las Figuras 6.11 (a) y (b) respectivamente; en estos experimentos el parámetro t tomó valores en el rango [1.4, 2.0].



(a) (b)

Figura 6.11: Construcción de t-Spanners en espacio de strings variando t, $n = 23000$. (a) Tiempo de CPU. (b) Arcos generados.

AESA, considerando los 23000 nodos, necesita para sus cálculos una matriz con aproximadamente 265 millones de distancias, en cambio para el valor de t más exigente ($t = 1.4$) sólo se utilizaron 8.5 millones de arcos, aproximadamente un 3.21 % de la cantidad total de arcos del grafo completo.

6.3.2. Búsqueda

En las Figuras 6.12, 6.13 y 6.14, se muestran los resultados de los experimentos de recuperación de objetos pertenecientes a espacios métricos de strings considerando la distancia de edición, para distintos radios de recuperación $r = 1$, 2 y 3. Nuevamente se

considera la comparación entre el algoritmo basado en t-spanners, AESA y el algoritmo basado en pivotes con un uso de memoria equivalente.

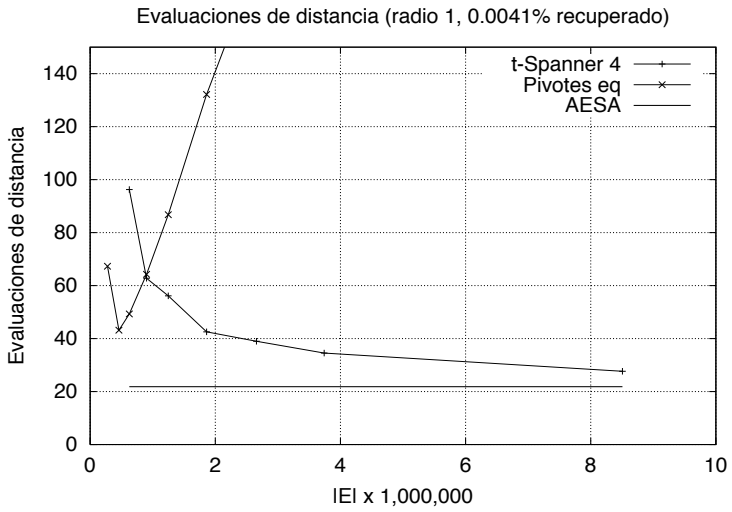


Figura 6.12: Consultas por rango en espacios de strings, radio de búsqueda $r = 1$. El algoritmo basado en pivotes alcanza 540 e.d. para 8.5 millones de arcos ($t = 1.4$).

Para el radio 1 (Figura 6.12), se aprecia que para $t = 2.0$ se comporta mejor el algoritmo basado en pivotes, en cambio, a medida que el valor de t disminuye (lo que se traduce en que la estructura utiliza más memoria), se observa que el t-spanner responde de mejor forma, llegando a ser muy competitivo para 8.5 millones de arcos ($t = 1.4$), donde realiza sólo un 27 % más de evaluaciones de distancia (e.d.) que AESA con un uso de memoria de un 3.21 %. Es interesante notar que la curva que representa el uso de e.d. del algoritmo basado en pivotes, presenta un punto mínimo, que corresponde al número óptimo de pivotes, y luego la cantidad de e.d. aumenta sostenidamente, sin ser capaz de aprovechar el hecho de que dispone de más memoria para el índice de búsquedas no pudiendo resolver la consulta con menos e.d.

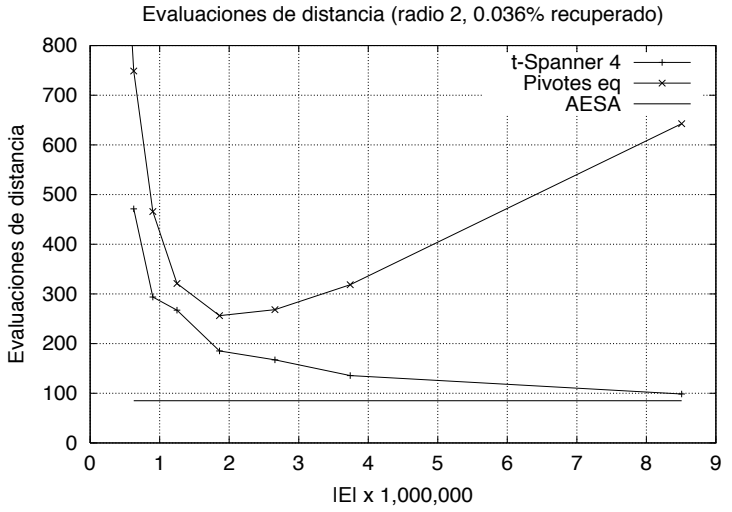


Figura 6.13: Consultas por rango en espacios de strings, radio de búsqueda $r = 2$.

En el caso de radio 2 (Figura 6.13), se nota el mejor desempeño del t-spanner con sólo un 16 % de e.d. adicionales para $t = 1.4$. En este caso también se aprecia que el algoritmo basado en pivotes, una vez alcanzado el número óptimo de pivotes, aumenta los costos de búsqueda sostenidamente, en cambio el t-spanner responde mejor a medida que t disminuye.

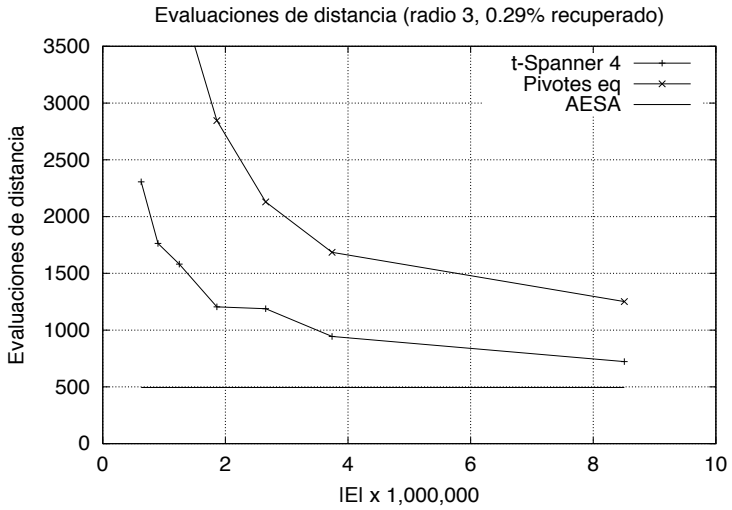


Figura 6.14: Consultas por rango en espacios de strings, radio de búsqueda $r = 3$. El algoritmo basado en pivotes alcanza 6100 e.d. para 0.6 millones de arcos ($t = 2.0$).

En el caso de radio 3 (Figura 6.14), se nota un mejor desempeño para todo el rango del algoritmo basado en t-spanners por sobre el del algortimo equivalente de pivotes. Comparado con AESA, se tiene sólo un 46 % de e.d. adicionales para $t = 1.4$. También se aprecia que no se alcanza el número óptimo de pivotes.

Para entender este resultado favorable, se debe contemplar el hecho de que la distancia de edición es una función discreta. Esto se traduce en que al t-aproximar las distancias, en la práctica, las distancias aproximadas se truncan al entero inferior, lo que se puede interpretar como que el t efectivo de la estructura es "menor" al t nominal, luego la pérdida de discriminación para distancias cortas no es tan notoria como en el caso del espacio métrico vectorial, con lo que se recupera el poder de discriminación de AESA.

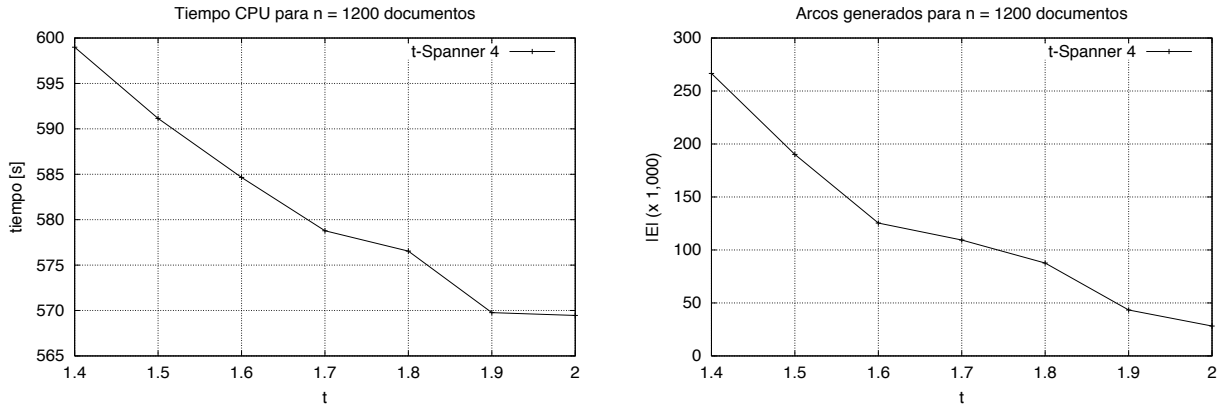
6.4. Espacio Métrico de los Documentos con Distancia Coseno

El espacio métrico de los documentos obedece a la noción de un espacio métrico general, en el sentido de que el cálculo de la e.d. es muy costoso, y además, no se puede acceder a la geometría espacial de los objetos, sino que sólo a la distancia medida entre ellos. Se determinó experimentalmente que la dimensión del espacio de documentos es $dim = 30$, lo que corresponde a un espacio de dimensionalidad alta, lo cual se considera intratable [CN01].

En este espacio métrico, se realizaron pruebas de búsqueda variando el radio de tolerancia, para valores de $t \in [1.4, 2.0]$. Se consideró un conjunto de aproximadamente 1200 documentos.

6.4.1. Construcción

Nuevamente se utilizó el algoritmo recursivo para la construcción de los t-spanners. Los resultados de la construcción tanto en tiempo de CPU como en arcos generados se ven en las Figuras 6.15 (a) y (b) respectivamente; en estos experimentos el parámetro t tomó valores en el rango [1.4, 2.0].



(a) (b)

Figura 6.15: Construcción de t-Spanner en espacio de documentos variando t, $n = 1200$. (a) Tiempo de CPU. (b) Arcos generados.

AESA, considerando los 1200 nodos, necesita para sus cálculos una matriz con aproximadamente 720000 distancias, en cambio para el valor de t menos exigente ($t = 2.0$) sólo se utilizaron 28000 arcos, aproximadamente un 3.83 % de la cantidad total de arcos del grafo completo. Se debe recordar que, según lo mostrado en el gráfico 6.10, en dimensiones altas el método basado en t-spanners no obtiene mayor beneficio con valores de t más pequeños.

6.4.2. Búsqueda

En las Figuras 6.16 y 6.17 se muestran los experimentos de recuperación de objetos pertenecientes a espacios métricos de documentos considerando la distancia coseno, para radios de recuperación escogidos para recuperar 1 ó 10 documentos en promedio por cada consulta ($r = 0.1325, 0.167$ respectivamente). Nuevamente se considera la comparación entre el algoritmo basado en t-spanners, AESA y el algoritmo basado en pivotes con un uso de memoria equivalente.

Para el caso de recuperar 1 documento (Figura 6.16), se aprecia que para todo el rango de valores explorados, el algoritmo basado en t-spanners se comporta mejor que el algoritmo basado en pivotes. El algoritmo basado en t-spanner es extremadamente competitivo frente a AESA, realizando sólo un 9 % más de evaluaciones de distancia que AESA con un uso de memoria de un 3.84 % para el caso $t = 2.0$. Para valores menores de t la cantidad de evaluaciones de distancia adicionales que realiza el algoritmo basado en t-spanner es menor, llegando a realizar sólo un 2.3 % de evaluaciones adicionales con $t = 1.4$, pero con el inconveniente de que requiere más memoria para construir el índice de búsquedas.

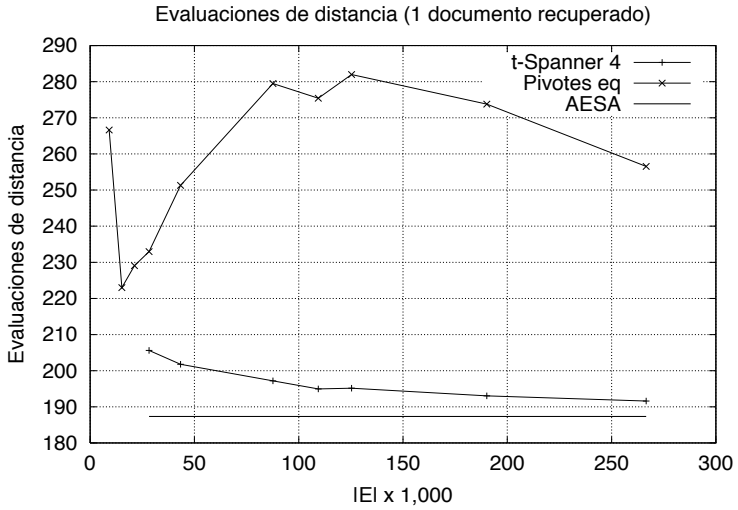


Figura 6.16: Consultas por rango en espacios de documentos, en promedio se recupera 1 documento por consulta, radio de búsqueda $r = 0.1325$.

Para el caso de recuperar 10 documentos (Figura 6.17), también se aprecia que para todo el rango de valores explorados, el algoritmo basado en t-spanners se comporta mejor que el algoritmo basado en pivotes. En este caso AESA simulado realiza sólo un 8 % más de evaluaciones de distancia que AESA con un uso de memoria de un 3.84 % para el caso $t = 2.0$. Para valores menores de t la cantidad de evaluaciones de distancia adicionales que realiza AESA simulado es menor, llegando a realizar sólo un 2.2 % de evaluaciones adicionales con $t = 1.4$, pero con el inconveniente de que requiere más memoria para construir el índice de búsquedas.

Se podría explicar este resultado considerando que los espacios métricos reales, a diferencia de los espacios métricos sintéticos generados según una distribución uniforme de los datos, presentan regiones en donde naturalmente se concentran los objetos del espacio y que el t-spanner obtiene beneficios de la existencia de los clusters del espacio. En cambio, como la selección de los pivotes es aleatoria, no se puede asegurar que dicha selección obtenga provecho de estas zonas con una concentración mayor de objetos. En la sección 6.5 se explica detalladamente esta conjetura y se demuestra experimentalmente.

6.5. Espacio Métrico $\mathbb{R}^D$ Gaussiano con Distancia Euclidiana

Esta serie de experimentos se realizó sobre un conjunto sintético de puntos escogidos según una distribución gaussiana en un espacio de 20 dimensiones. No se utilizó el hecho de que el espacio tiene coordenadas, sino que se empleó como si los puntos fueran objetos abstractos de un espacio métrico desconocido.

En esta sección se pretende demostrar empíricamente la conjetura de que el t-spanner obtiene mayores beneficios de la existencia de clusters en el espacio que estructuras alternativas.

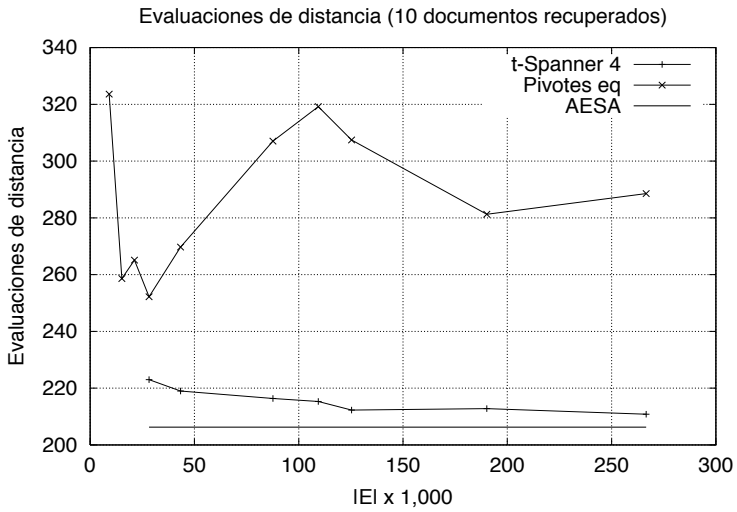


Figura 6.17: Consultas por rango en espacios de documentos, en promedio se recuperan 10 documentos por consulta, radio de búsqueda $r = 0.167$.

Los espacios métricos reales efectivamente presentan regiones donde se concentran los objetos del espacio. Tales regiones se conocen como "clusters" (en español racimos o grupos) y estas regiones se caracterizan por el hecho de que los objetos que las forman están muy cercanos. Con el espacio vectorial gaussiano, se pretende simular un espacio métrico real. Para esto se consideran dos cantidades de clusters: 32 y 256, y por otro lado, tres valores de desviación estándar: $\sigma = 0.1$, 0.3 y 0.5; mayor detalle de los espacios vectoriales gaussianos utilizados se muestra en la sección 3.4.1.

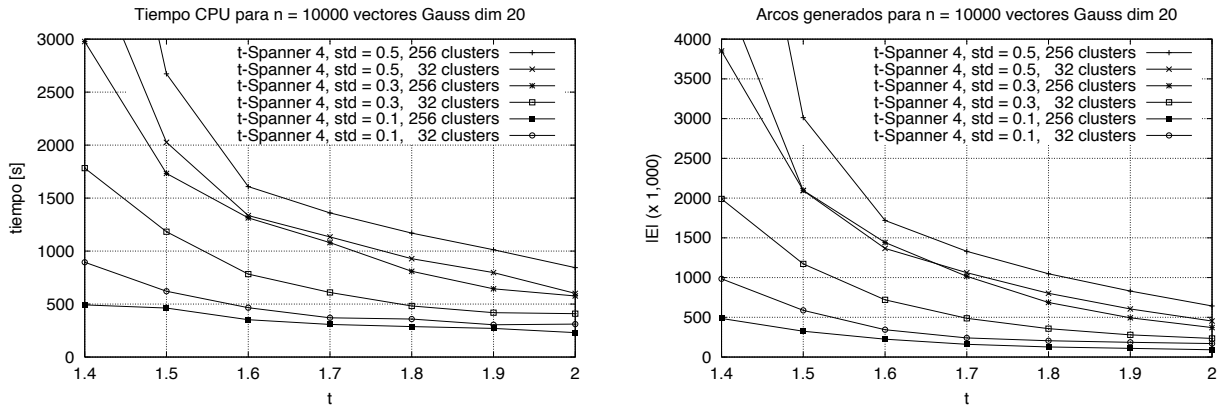
6.5.1. Construcción

En las Figuras 6.18 (a) y (b), se muestran los resultados de construcción de t-spanners. Se observa que a medida que σ se hace más pequeño, tanto la cantidad de arcos necesarios como el tiempo de CPU requerido para la construcción del t-spanner disminuyen.

Esto se explica considerando que para construir el t-spanner es necesario inspeccionar las distancias medidas entre todos los pares de objetos, y como el algoritmo recursivo detecta los clusters por construcción (pues se basa en la mezcla de conjuntos de diamétro pequeño), el t-spanner generado se adapta naturalmente a la existencia de los clusters. Esto se debe a que el algoritmo recursivo puede hacer buenas aproximaciones de las distancias de un cluster a otro utilizando pocos arcos, tal como se ve en la Figura 6.19, donde para conectar los dos clusters, el t-spanner del ejemplo sólo requiere de un arco.

6.5.2. Búsqueda

Como se aprecia en los gráficos de las Figuras 6.20 y 6.21, los resultados son favorables para el método basado en t-spanners en la medida de que los clusters son más concentrados.



(a) (b)

Figura 6.18: Construcción de t-spanners en espacio vectorial gaussiano variando t, $n = 10000$. (a) Tiempo de CPU. (b) Arcos generados. En (a) 1.4-spanner 4 para 32 clusters alcanza 3900 segundos y para 256 clusters alcanza 6200 segundos, ambos con $\sigma = 0.5$. En (b) 1.4-spanner 4 para 32 clusters alcanza 4.7 millones de arcos y para 256 clusters alcanza 8.2 millones de arcos, ambos con $\sigma = 0.5$.

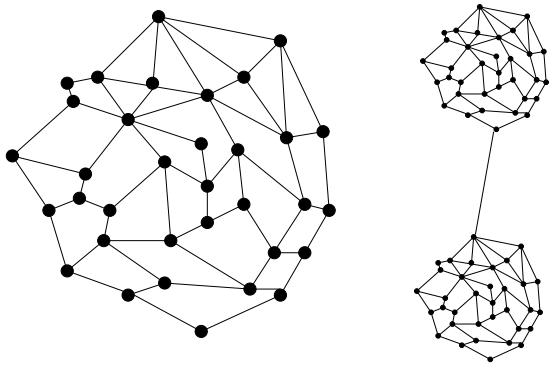
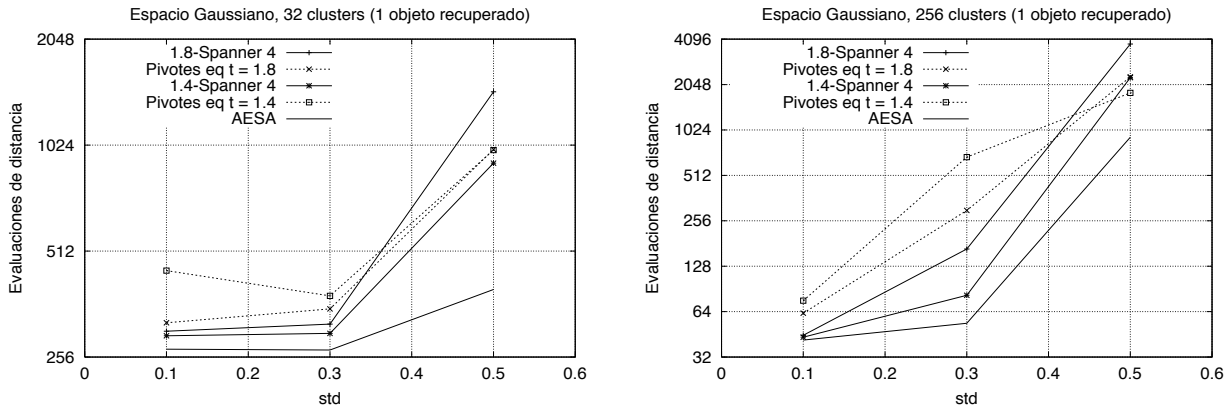


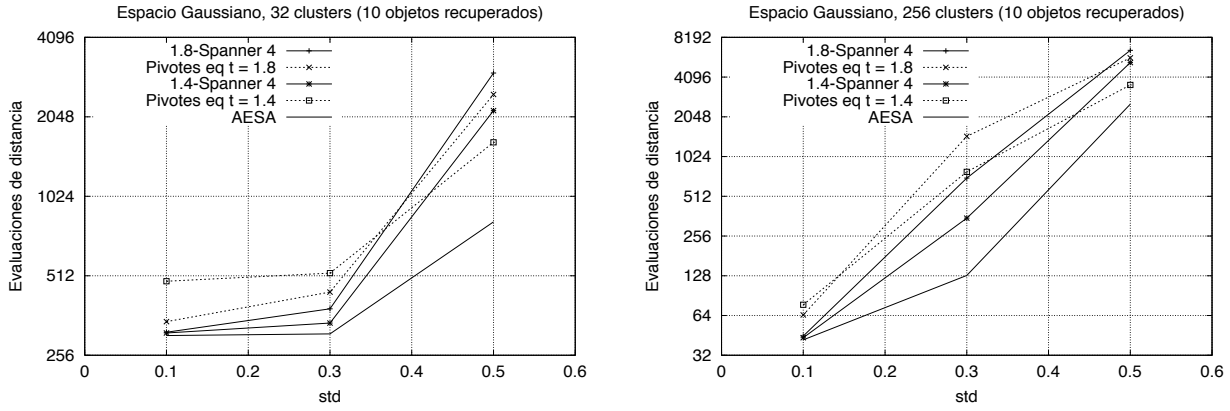
Figura 6.19: Ejemplo de t-spanner sobre clusters. A la izquierda, un t-spanner sobre un conjunto de nodos distribuidos uniformente, a la derecha, el t-spanner sobre un conjunto de nodos compuesto por dos clusters. Note que para conectar los clusters sólo requiere de un arco.

Para el caso de $\sigma = 0.1$, en donde se tienen clusters muy pequeños, los resultados son mejores que con pivotes, y es extremadamente competitivo frente a AESA, requiriendo sólo de un 0.34 % de memoria de AESA (con un 2-spanner), y pagando un exceso de un 12 % de e.d. sobre AESA para recuperar 1 objeto, y tan sólo un 3 % de e.d. sobre AESA para recuperar 10 objetos. Para $\sigma = 0.3$, los resultados con t-spanners son mejores que con pivotes, y bastante competitivos contra AESA, en donde utilizando un 4 % de la memoria de AESA (con un 1.4-spanner) y un exceso de un 12 % de e.d. sobre AESA para recuperar 1 objeto, y un 10 % de e.d. sobre AESA para recuperar 10 objetos. Esta situación se revierte para $\sigma = 0.5$, en donde se muestra que los pivotes tienen una mejor respuesta que t-spanners. Es necesario indicar que para el espacio considerado (distribución gaussiana en el intervalo [-1, 1]), con $\sigma = 0.5$, se tiene una distribución pseudo uniforme de los objetos, puesto que los clusters son lo suficientemente extendidos como para traslaparse.



(a) (b)

Figura 6.20: Consultas por rango en espacios vectoriales gaussianos, en promedio se recupera 1 objeto por consulta.



(a) (b)

Figura 6.21: Consultas por rango en espacios vectoriales gaussianos, en promedio se recuperan 10 objetos por consulta.

Los resultados favorables para $\sigma = 0.1$ y 0.3 se explican considerando que, como se dijo en la sección 6.5.1, el t-spanner se adapta a la presencia de clusters, en cambio, debido a que la selección de los pivotes es aleatoria, no se puede asegurar que dicha selección obtenga provecho de estas zonas con una concentración mayor de objetos. En cambio, cuando la distribución de los datos es uniforme o pseudo uniforme ($\sigma = 0.5$), el t-spanner pierde capacidad de discriminación de objetos al realizar las búsquedas.

Esto demuestra experimentalmente la conjetura de que los t-spanners aprovechan los clusters que naturalmente se forman en un espacio métrico real. A medida de que los clusters son más pequeños mejora el comportamiento del t-spanner, y cuando los clusters son lo suficientemente grandes como para traslaparse entre sí, lo que se acerca a un espacio con distribución uniforme, esta ventaja se pierde. Es sabido que todos los algoritmos de búsqueda mejoran cuando hay clusters, pero los t-spanners mejoran más que, por ejemplo, los pivotes.

Capítulo 7

Conclusiones

En este trabajo se abordó el problema de buscar objetos en un *Espacio métrico* (EM) desde la perspectiva de utilizar un *t-Spanner* para representar la estructura de la base de datos (BD) del EM. Se busca reducir la cantidad de evaluaciones de distancia (e.d.). En una primera etapa, se diseñaron cinco algoritmos de construcción de t-spanners. En una segunda etapa, se diseñaron mecanismos de actualización de la estructura y un algoritmo de búsqueda que utiliza el t-spanner como estructura de soporte.

Una de las mejores técnicas para buscar objetos en un espacio métrico es AESA (Approximating Eliminating Search Algorithm), que es un algoritmo que presenta un buen desempeño incluso en dimensiones altas, que es un caso difícil para la mayoría de los algoritmos de búsqueda en EM conocidos actualmente. El problema de este algoritmo es que la estructura donde realiza las búsquedas corresponde a una matriz en donde están precalculadas las $n(n-1)/2$ distancias entre pares de objetos, con lo que el costo en memoria del algoritmo llega a ser $O(n^2)$, lo que lo hace impracticable para valores de $n > 10000$, considerando la capacidad del hardware disponible actualmente.

Por otro lado, el t-spanner es un concepto conocido en la teoría de grafos. Dado un grafo $G(V, A)$, un t-spanner es un grafo $G'(V, E), E \subseteq A$, construido a partir del grafo G original, considerando una versión relajada de la condición de encontrar los caminos más cortos entre pares de nodos. La principal propiedad de un t-spanner es que garantiza que el costo del camino entre cualquier par de vértices del grafo G' es a lo sumo t veces la distancia del camino mínimo obtenido en el grafo original G, lo que se conoce como *condición de t-spanner*.

La idea central de esta tesis fue unir estos dos conceptos: Para conseguir la reducción en el uso de memoria de AESA, se considera la alternativa de utilizar *t-Spanners* para representar la estructura de distancias entre objetos del espacio métrico. De este modo, se puede adaptar el funcionamiento de AESA a la estructura del t-spanner, permitiendo su utilización en aplicaciones reales.

A continuación se resumen los principales logros de la tesis y se analiza su impacto.

Construcción de t-spanners

Se proponen cinco algoritmos de construcción de t-spanners, y condiciones que permiten que el uso de memoria sea subcuadrático en la cantidad de elementos de la BD y con tiempo de CPU muy similar al que utiliza AESA para la construcción de la estructura de búsquedas.

El algoritmo básico optimizado (t-spanner 1) es el que produce t-spanners de mejor calidad, en el sentido de que poseen la menor cantidad de arcos, requiriendo de sólo $n(n-1)/2$ e.d., pero el tiempo de CPU que utiliza para el resto de los cálculos es cúbico en n, lo que le resta utilidad práctica. El algoritmo de inserción masiva de arcos (t-spanner 2), que es una evolución del anterior, tiene la virtud de que consigue tiempos de CPU levemente supercuadráticos (aproximadamente $O(n^{2.2})$), aumentando levemente la cantidad de arcos insertados; el problema es que requiere demasiadas e.d. (aproximadamente $O(|V| \cdot |E|)$ e.d.).

El algoritmo de inserción incremental de nodos (t-spanner 3) requiere de sólo $n(n-1)/2$ e.d., y el tiempo de CPU también es levemente supercuadrático. El inconveniente de este algoritmo es que genera t-spanners con una calidad inferior. El problema de la calidad se resuelve con el algoritmo recursivo (t-spanner 4), aumentando levemente la cantidad de e.d. a $n(n-1)/2 + O(n \log n)$ con costo en tiempo de CPU supercuadrático, pero con una mejor calidad en el t-spanner generado. Por último se desarrolló un algoritmo recursivo utilizando al algoritmo t-spanner 1 en el caso base, presentando una calidad en el t-spanner generado similar a t-spanner 4, pero con un tiempo de CPU más alto que t-spanner 4.

Luego de este análisis, que considera el compromiso tiempo de CPU versus arcos generados, se ha escogido la metodología de recursiva de construcción de t-spanners (t-spanner 4) como la mejor metodología de construcción.

Actualización de la estructura

Se proponen mecanismos de inserción y borrado de elementos en la BD. La inserción y borrado son eficientes pero degradan lentamente la calidad de la estructura. Adicionalmente se propone un algoritmo de reconstrucción del t-spanner luego de inserciones y borrados sucesivos. La combinación de estos mecanismos permite mantener actualizada la estructura de la BD preservando su calidad.

Recuperación de objetos

Se propone un algoritmo de solución de consultas por rango basado en el t-spanner como estructura de datos de soporte de la BD métrica.

La metodología de búsqueda en espacios métricos basada en t-spanners puede sacarle partido al hecho de que se disponga de mayor memoria para almacenar el índice, ya que a menor valor de t, se necesita mayor memoria para almacenar la estructura del índice, con la ventaja de que las aproximaciones mejoran. Esto permite disminuir el uso de evaluaciones de distancia al momento de realizar las búsquedas. En cambio, la metodología basada en pivotes no posee esta característica; una vez alcanzado el número óptimo de pivotes, considerar más pivotes no disminuye el uso de evaluaciones de distancia al momento de la búsqueda.

Comparación de t-spanners contra otras metodologías

Se muestran comparaciones de rendimiento para la recuperación de objetos de la BD métrica, mostrando que el algoritmo de búsqueda AESA simulado sobre el t-spanner es una excelente alternativa en comparación con AESA y el algoritmo clásico basado en pivotes. En espacios métricos del mundo real, se obtienen resultados altamente competitivos con respecto a AESA y mejores que los algoritmos basados en pivotes. En la recuperación de documentos, el algoritmo basado en t-spanners es extremadamente competitivo frente a AESA, realizando sólo un 9 % más de evaluaciones de distancia que AESA con un uso de memoria de un 3.84 % para el caso $t = 2.0$. Similar resultado se tiene para el espacio de los strings. Se proporciona evidencia emprica de que este buen comportamiento se mantiene en cualquier espacio que presente clusters.

Ventajas del método basado en t-spanners

En la fase de construcción del t-spanner, se aprovecha la distribución espacial de los objetos en el espacio métrico considerado. Los espacios métricos del mundo real presentan regiones donde se concentran los objetos, y la metodología recursiva naturalmente se adapta a la distancia relativa entre los objetos.

Las conclusiones más fuertes que se obtienen de este trabajo son:

- Los t-spanners presentan un muy buen comportamiento en espacios métricos compuestos por datos obtenidos del mundo real. En particular, permiten un excelente compromiso entre tiempo y memoria como alternativa a AESA.
- Bajo la presencia de clusters en el EM, la estructura de búsqueda, el t-spanner, se adapta naturalmente a la distribución espacial de los datos, permitiendo de esta manera obtener mejores resultados que otras conocidas actualmente, tanto en la fase de construcción como en la de búsqueda (en particular, mejor que con algoritmos basados en pivotes).
- Uno de los problemas que tiene el método basado en t-spanners es que requiere de un mayor esfuerzo de cálculo que otras metodologías para eliminar candidatos al momento de hacer las búsquedas. Sin embargo, cuando el costo de cálculo de la evaluación de distancia es efectivamente elevado, como es el caso del espacio de los documentos con distancia coseno, el costo de la e.d. absorbe el esfuerzo de cálculo extra debido a la estructura del t-spanner, siendo éste un caso en donde realmente se notan las virtudes de esta metodología.

Trabajos futuros

Este trabajo de tesis se puede extender tanto en el tema de la construcción de t-spanners métricos como en la búsqueda.

Una extensión directa de los algoritmos de construcción consiste en considerar un valor de t que dependa de la distancia entre los nodos, de modo de que, a menor distancia entre los nodos, menor sea el valor de t. Con esto se debiera conseguir por un lado t-spanners con una

selección reducida de arcos, y por otro factores de aproximación local cercanos a 1.0, lo que a la larga permitirá mantener un uso subcuadrático de memoria, mejorando la capacidad de discriminación a medida que los elementos seleccionados como pivotes se acercan a la consulta.

Una característica de los t-spanners es que los arcos conectan nodos cercanos entre ellos. Se podrían obtener mejoras para los algoritmos de búsqueda explorando la posibilidad de utilizar el t-spanner como una estructura de navegación sobre el espacio métrico. De esta manera se podría considerar la búsqueda como la combinación de un mecanismo de acercamiento hacia la consulta, y luego la eliminación de los candidatos que estén fuera del cascarón de exclusión. Esto tiene sentido, puesto que mientras más cerca de la consulta está el pivote utilizado para excluir elmentos, mayor será la cantidad de objetos que elimine, luego la posibilidad de acercar el pivote a la consulta aumentará su potencial de exclusión de candidatos. Esto posiblemente permita un buen compromiso entre número de evaluaciones de distancia y tiempo extra de CPU.

Publicaciones generadas durante este trabajo de tesis

A partir de las investigaciones desarrolladas en este trabajo de tesis se escribieron dos publicaciones; se pueden ver ambas en los anexos.

Bibliografía

[ADD+93] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete Computational Geometry*, 9:81–100, 1993.

[ADDJ90] I. Althöfer, G. Das, D. Dobkin, and D. Joseph. Generating sparse spanners for weighted graphs. In *Proc. 2nd Scandinavian Workshop on Algorithm Theory (SWAT'90)*, LNCS 447, pages 26–37, 1990.

[Bar98] Y. Bartal. On approximating arbitrary metrics by tree metrics. In *Proc. 30th Symposium on the Theory of Computing (STOC'98)*, pages 161–168, 1998.

[BBK02] C. Böhm, S. Berchtold, and D. Keim. Searching in high-dimensional spaces — index structures for improving the performance of multimedia databases. *to appear in ACM Computing Surveys*, 2002.

[Ben75] J. Bentley. Multidimensional binary search trees used for associative searching. *Comm. of the ACM*, 18(9):509–517, 1975.

[Ben79] J. Bentley. Multidimensional binary search trees in database applications. *IEEE Trans. on Software Engineering*, 5(4):333–340, 1979.

[BK73] W. Burkhard and R. Keller. Some approaches to best-match file searching. *Comm. of the ACM*, 16(4):230–236, 1973.

[BKK96] S. Berchtold, D. Keim, and H. Kriegel. The x-tree: an index structure for high-dimensional data. In *22nd Conference on Very Large Databases (VLDB'96)*, pages 28–39, 1996.

[BO97] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *ACM SIGMOD International Conference on Management of Datas*, pages 357–368. Sigmod Record 26(2), 1997.

[Bri95] S. Brin. Near neighbor search in large metric spaces. In *Proc. Very Large Databases (VLDB'95)*, pages 574–584. Morgan Kaufmann, 1995.

[Bus00] B. Bustos. *Investigación de técnicas de selección de pivotes para algoritmos de búsqueda en espacios métricos*. Universidad de Chile, 2000. Memoria para optar al título de Ingeniero Civil en Computación.

[BWY80] J. Bentley, B. Weide, and A. Yao. Optimal expected-time algotithms for closet point problems. *ACM Trans. on Mathematical Software*, 6(4):563–580, 1980.

[BYCMW94] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proc. Combinatorial Pattern Matching (CPM'94)*, number 807 in LNCS, pages 198–212. Springer-Verlag, 1994.

[BYRN99] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[CCG+98] M. Charikar, C. Chekuri, A. Goel, S. Guha, and S. Plotkin. Approximating a finite metric by a small number of tree metrics. In *Proc. 39th Symp. on Foundations of Computer Science (FOCS'98)*, pages 379–388, 1998.

[CDNS95] B. Chandra, G. Das, G. Narasimhan, and J. Soares. New sparseness results on graph spanners. In *Internat. J. Comput. Geom. Appl.*, volume 5, pages 125–144, 1995.

[CMN99] E. Chávez, J. Marroquín, and G. Navarro. Overcoming the curse of dimensionality. In *European Workshop on Content-Based Multimedia Indexing (CBMI'99)*, pages 57–64, 1999.

[CN01] E. Chávez and G. Navarro. Towards measuring the searching complexity of metric spaces. In *Proc. Mexican Computing Meeting*, volume II, pages 969–978, Aguascalientes, México, 2001. Sociedad Mexicana de Ciencias de la Computación.

[CNBYM01] E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquín. Proximity searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.

[Coh98] E. Cohen. Fast algorithms for constructing t- spanners and paths with stretch t. In *SIAM Journal on Computing*, volume 28, pages 210–236, 1998.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: an efficient access method for similarity search in metric spaces. In *23rd Conference on Very Large Databases (VLDB'97)*, pages 426–435, 1997.

[DH87] F. Dehne and H. Noltemeier. Voronoi trees and clustering problems. In *Information Systems*, volume 12(2), pages 171–175, 1987.

[Epp99] D. Eppstein. Spanning trees and spanners. In *Handbook of Computational Geometry*, pages 425–461. Elsevier, 1999.

[GG98] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[GLN00] J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Improved greedy algorithms for constructing sparse geometric spanners. In *Proc. 7th Scand. Workshop Algorithm Theory*, volume 1851 of Lecture Notes Comput. Sci., Berlin, pages 314–327, 2000.

[Gut84] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[Har95] D. Harman. Overview of the Third Text REtrieval Conference. In *Proc. Third Text REtrieval Conference (TREC-3)*, pages 1–19, 1995. NIST Special Publication 500-207.

[JD88] A.K. Jain and R.C. Dubes. *Algorithms For Clustering Data*. Prentice-Hall, Englewood Cliffs, 1988.

[Kei88] J.M. Keil. Approximating the complete Euclidean graph. In *Proc. 1st Scandinavian Workshop in Algorithm Theory (SWAT'88)*, LNCS 318, pages 208–213, 1988.

[KM83] I. Kalantari and G. McDonald. A data structure and an algorithm for the nearest point problem. *IEEE Transactions on Software Engineering*, 9(5), 1983.

[KP94] G. Kortsarz and D. Peleg. Generating sparse 2-spanners. *Journal of Algorithms*, 17(2):222–236, September 1994.

[MOV94] L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (aesa) with linear preprocessing-time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.

[Nav99] G. Navarro. Searching in metric spaces by spatial approximation. In *Proceedings of the 6th International Symposium on String Processing and Information Retrieval (SPIRE'99)*, pages 141–148. IEEE CS Press, 1999.

[NP02] G. Navarro and R. Paredes. Practical construction of metric t-spanners. Technical Report TR/DCC-2002-4, Dept. of Computer Science, Univ. of Chile, July 2002.

[NPC02] G. Navarro, R. Paredes, and E. Chávez. t-Spanners as a data structure for metric space searching. In *Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE 2002)*, LNCS. Springer, 2002. To appear.

[NR02] G. Navarro and M. Raffinot. *Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, 2002.

[PS89] D. Peleg and A. Schaffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

[PU89] D. Peleg and J. Ullman. An optimal synchronizer for the hypercube. *SIAM J. on Computing*, 18:740–747, 1989.

[RS91] J. Ruppert and R. Seidel. Approximating the d-dimensional complete Euclidean graph. In *3rd Canadian Conference on Computational Geometry*, pages 207–210, 1991.

[Sam84] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, 1984.

[SW90] D. Shasha and T. Wang. New techniques for best-match retrieval. *ACM Trans. on Information Systems*, 8(2):140–158, 1990.

[Uhl91] J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40:175–179, 1991.

[VR86] E. Vidal Ruiz. An algorithm for finding nearest neighbors in (approximately) constant time. *Pattern Recognition Letters (PRL'96)*, 4:145–157, 1986.

[Wei95] M. A. Weiss. *Estructuras de datos y algoritmos.* Addison-Wesley Iberoamericana, 1995.

[WJ96] D. White and R. Jain. Algorithms and strategies for similarity retrieval. Technical Report VCL-96-101, Visual Computing Laboratory, University of California, La Jolla, California, July 1996.

[Yia93] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA'93)*, pages 311–321. SIAM Press, 1993.

[Yia98] P. Yianilos. Excluded middle vantage point forests for nearest neighbor search. Technical report, NEC Research Institute, Princeton, NJ, July 1998.

Apéndice A

Publicaciones Generadas

A continuación se presentan las publicaciones desarrolladas durante el presente trabajo de tesis.

La primera de ellas, *Practical Construction of Metric t-Spanners* [NP02], es un reporte técnico relativo al tema de construcción de t-spanners, que corresponde al capítulo 4, algoritmos para construcción de t-spanners. El reporte está publicado en `ftp://ftp.dcc.uchile.cl/pub/users/gnavarro/buildsp.ps.gz`, bajo el número TR/DCC-2002-4. Está en fase de correcciones para luego ser enviado a alguna conferencia internacional.

La segunda, *t-Spanners as a Data Structure for Metric Space Searching*, [NPC02], es un artículo aceptado en la 9th International Symposium on String Processing and Information Retrieval (SPIRE 2002), próxima a publicarse en las actas de dicha conferencia en septiembre de 2002 y tiene que ver con el capítulo 5, uso de t-spanners para búsqueda en espacios métricos. Las actas de esta conferencia aparecerán en la serie *Lecture Notes in Computer Science*, publicada por Springer.

Se planea enviar los trabajos a revistas internacionales cuando sea el momento.