UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"Ubicación óptima de círculos intersectando rectas"

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRA EN CIENCIAS (COMPUTACIÓN)

P R E S E N T A:

MAYRA YADIRA CORVERA ESPINOZA

DIRECTOR DE TESIS: Dr. Jorge Urrutia Galicia

México, D.F. **2009.**

A mi familia con amor.

Agradecimientos

Gracias Jorge Urrutia por haber confiado en mí de la manera que lo has hecho desde que empecé a trabajar contigo y hasta la fecha, es muy valioso el conocimiento que he obtenido siendo tu alumna, no sólo en Geometría Computacional, si no también en la forma de vivir cada día y de ver el mundo. Te admiro.

Agradezco a mi familia el apoyo brindado. Mamá yo sé que hubieras querido estar conmigo todo este tiempo.

Gracias a mis compañeros por el apoyo y compañía este tiempo, en especial a Kno y a Joel, por ayudarme cuando lo necesité y por tenerme paciencia cuando incluso ni yo misma me la tenía.

Gracias a José Miguel Díaz-Bañez por su asesoría con los problemas de este trabajo de tesis. Gracias a Sergio Rajsbaum, a Francisco Hernández, a Juan José Montellanos y a José Galaviz por aceptar ser mis sinodales.

Índice general

Índice de figuras

Prefacio

La Geometría Computacional tiene como objetivo estudiar el diseño y optimización de algoritmos geométricos de diferentes tipos. Al hablar de cómo surgió, resulta indispensable mencionar a Euclides, quien es el escritor de la obra matemática *Los Elementos*. Este trabajo fue consultado por todo aquel estudiante que quería aprender geometría. Se sabe que gran parte del contenido de *Los Elementos* fue desarrollado por Euclides, otra parte, se supone, está basada en libros anteriores y aportaciones de Eudoxio. De cualquier forma *Los Elementos* es el primer trabajo que presenta contenido de geometría de una manera lógica, por lo que sirvió como base para el razonamiento lógico y sustentó lo que hoy conocemos como geometría. En *Los Elementos* se puede encontrar la llamada construcción de Euclides, que consiste de un algoritmo y su prueba. Esta construcción satisface todos los requerimientos de un algoritmo: es un conjunto ordenado de operaciones, bien definido y finito; que permite dar la solución a un problema. Hay quienes dicen que así surge la Geometría Computacional.

A diferencia de la geometría, el análisis de algoritmos estuvo en pausa. Al tener las construcciones de Euclides, los geómetras se dedicaron, en parte, a buscar la manera de refinarlas para que ejecutaran un

número pequeño de operaciones. Sin embargo, fue hasta 1902 cuando surgió el término *complejidad*, aunque no fue a nivel computacional. Émile Lemoine creo un sistema llamado *Géométrographie*, el objetivo de este sistema de construcciones era proporcionar un proceso para simplificar las construcciones existentes. En la descripción de su sistema, Lemoine enlistó las siguientes operaciones:

1. Colocar un compás en un punto dado.
2. Colocar un compás en una línea dada.
3. Trazar un círculo con el compás colocado en el punto o línea mencionados anteriormente.
4. Colocar una regla en una línea dada.
5. Extender la línea dada con la regla.

En este trabajo él llamó *simplicidad* al número total de operaciones ejecutadas por una construcción, después él mismo reconoció que era mejor utilizar el término *medida de complejidad*, aunque en ese momento no existía relación entre el número de operaciones realizadas y la cantidad de datos disponibles. La complejidad de un algoritmo se refiere, comúnmente, al número de pasos y espacio en memoria requeridos para dar solución a un problema de manera algorítmica.

Existen otros parámetros que pueden ser estudiados, por ejemplo, el número de procesadores requeridos para implementar un algoritmo en paralelo.

El análisis de algoritmos forma parte de la teoría de complejidad computacional, y ayuda a determinar, de manera teórica, la cantidad de recursos como espacio y tiempo, que requiere un algoritmo. La complejidad computacional se mide respecto al número de pasos que toma resolver un problema a partir del tamaño de la entrada (datos requeridos para resolver el problema), por ejemplo si se toma un problema con entrada de longitud n que puede resolverse en n^2 pasos, se dice que su complejidad en tiempo de ejecución es de n^2, el número exacto de pasos depende de varios factores como la máquina en la cual se implementa la solución y el lenguaje de programación seleccionado. La notación O se utiliza para generalizar la noción de costo independientemente de la máquina o el lenguaje utilizado, así cuando un problema tiene costo en tiempo $O(n^2)$ en una máquina y lenguaje dados, su costo será el mismo en otra máquina o utilizando otro lenguaje. Existen 3 cotas utilizadas comúnmente, éstas son dadas respecto al tamaño de la entrada $\cdot$: inferior, de notación $\Omega(\cdot)$, superior, de notación $O(\cdot)$ y justa, de notación $\Theta(\cdot)$. Usualmente la cota del análisis de complejidad para un algoritmo se refiere a la cantidad de operaciones que el algoritmo requiere en el peor de los casos, es de-

cir, considerando el peor de los escenarios. Por lo que generalmente se utiliza la cota O.

Varias áreas de aplicación tienen problemas que requieren soluciones eficientes; algunos son de naturaleza geométrica como el problema del agente viajero [4] y el árbol generador de peso mínimo [32]. En 1978, Michael Shamos de la Universidad de Yale introdujo el término Geometría Computacional en su tesis doctoral. La Geometría Computacional ha tenido aplicaciones en diversos campos como: gráficos por computadora, robótica, redes y sistemas de información geográfica. Está relacionada con Topología, Geometría Combinatoria, Teoría de Gráficas, Matemáticas Discretas y Matemáticas Aplicadas, entre otras.

La característica fundamental de la Geometría Computacional es que permite analizar las propiedades de objetos geométricos de cierto problema y retomar las que son útiles para resolver el problema computacionalmente, es decir, de forma algorítmica.

1

Introducción

1.1 Origen del problema

Los problemas estudiados en este trabajo de tesis tienen relación con dos problemas muy conocidos en Geometría Computacional: el problema de *apuñalar* y el problema del k-centro.

En el problema de *apuñalar* se desea encontrar un objeto, que es llamado *puñal*, que intersecte a todos los objetos geométricos de una colección finita (que se denota como C) en una dimensión d del espacio euclidiano. Muchos investigadores han estudiado variantes de este

problema donde C es una colección de n puntos, segmentos de rectas, rectas, rayos o hiperesferas. Y el objeto que se desea encontrar es un segmento de recta, una recta, un hiperplano o un disco. Los resultados sobre estas variantes pueden consultarse en el trabajo de M. E. Houle *et al*, [21].

El *problema del 1-centro* es un clásico en esta área, este problema consiste en encontrar el radio de un círculo que intersecte a una colección de n puntos en el plano tal que este radio sea el menor de todos. Este problema tuvo su origen en la vida real, por ejemplo, se desea ubicar un conjunto de centros de salud de tal manera que el máximo tiempo de respuesta a una emergencia se minimice. Para un radio dado, Megiddo probó cómo resolverlo en tiempo lineal, [29], subsecuentemente Dyer dio un algoritmo con la misma complejidad, [12, 14]. En la siguiente versión de este problema, también se considera a C como una colección de puntos en el plano pero se buscan dos círculos, por esto se conoce como el *problema del 2-centro*. Asimismo, la generalización de los problemas anteriores es el famoso *problema del k-centro*. En este problema se buscan k discos, tales que el radio más grande es minimizado, cuya unión cubre C, y se resuelve en $O(n^{O(p)})$; sin embargo, cuando k es parte de la entrada el problema es **NP**-Completo, si el lector lo desea puede consultar este resultado en [30, 31].

La versión clásica del *problema del 2-centro* ha sido muy estudiada, este problema fue resuelto en 1994 por Z. Drezner, [11], con un tiempo de ejecución de $O(n^3)$. En el mismo año, Pankaj K. Agarwal y Micha Sharir, [1], mejoraron este resultado, ellos presentan una solucion de tiempo $O(n^2 \log^3 n)$. Asimismo, en 1994 Jerzy W. Jaromczyk y Miroslaw Kowaluk, [22], presentan una solución de tiempo $O(n^2 \log n)$. El mejor resultado hasta ahora fue presentado por Micha Sharir, [34], en 1996, con un tiempo de ejecución de $O(n \log^9 n)$.

En los problemas estudiados en este trabajo, C es una colección de rectas y el objeto que intersecta a C es representado como la unión de dos círculos tal que el radio del círculo más grande es minimizado. Hasta ahora no se conoce un algoritmo que resuelva este problema; sin embargo, existen algunas variantes que han sido resueltas por varios investigadores.

En el capítulo 4 de este trabajo se presenta un algoritmo para resolver una variante del *problema del 2-centro*. Este algoritmo se ejecuta en tiempo $O(n^2 \log^4 n)$, mejorando así, a otro algoritmo propuesto en este mismo trabajo. Como en otras soluciones, una de las partes más importantes del algoritmo es el procedimiento que resuelve el problema de decisión: dado un radio r, determinar si C es cubierto por 2 discos con radio r. Este procedimiento es combinado con la técnica de gráficas expansoras para obtener el algoritmo completo,

esta técnica es explicada en el capítulo 3 de este trabajo.

1.2 Localización de Servicios

La Localización de Servicios es una rama de la Geometría Computacional que tiene como objetivo determinar la mejor ubicación de uno o varios servicios, de tal manera que se satisfaga a un conjunto de clientes (puntos demandantes) y una serie de restricciones. Generalmente estas restricciones están relacionadas con la distancia que hay entre los servicios y sus usuarios.

Por ejemplo, ¿qué pasaría si todos los hospitales de una ciudad estuvieran en el mismo lugar? Algunas personas, esencialmente las que viven cerca de ese lugar, tendrían una ventaja ante las demás, a las personas vecinas de los hospitales no les costaría el mismo tiempo ni esfuerzo desplazarse hasta un hospital en caso de una emergencia; sin embargo, el resto de las personas tendrían la desventaja de correr el riesgo de no llegar a tiempo al hospital. Esta situación provoca una reflexión acerca de la distribución y ubicación de servicios.

Ahora supóngase que el gobernador de una ciudad desea mejorar el complejo vial agregando la opción de cambio de direcciones con glorietas. El gobernador solicita que cada vía tenga al menos un retorno disponible, es decir, cada vía debe pasar por al menos una

glorieta. Supóngase además que, por disposición federal, el tamaño de las glorietas sólo puede ser chico, mediano o grande. La persona encargada de estimar el presupuesto observó que construir una glorieta grande es más fácil, pero también mucho más costoso que construir dos chicas o dos medianas. Esto porque al construir dos glorietas, las vías pueden pasar por una o por la otra. Entonces es necesario determinar el tamaño óptimo entre los disponibles, es decir, se quiere saber si es mejor construir dos glorietas medianas o dos chicas. Incluso podría ser, que la mejor opción sea construir una glorieta chica y una mediana, por lo tanto, la pregunta que se realizaría en este caso sería ¿cuál es el tamaño *óptimo* que debe tener cada una de las dos glorietas?

El problema anterior es modelado de la siguiente manera: el área de la ciudad es representada por el plano, cada vía vehicular se representa como una recta y cada glorieta como un círculo. Es necesario determinar la ubicación y el radio de las glorietas, de manera que cada vía pase por al menos una glorieta, es decir, puede darse el caso de que en una vía haya acceso a las dos glorietas, pero no se permite que haya vía sin acceso a alguna glorieta. Este problema pertenece a la Localización de Servicios.

Otro ejemplo surge cuando una persona cambia de trabajo. Generalmente, una persona al pasar por esta situación busca el transporte

que lo lleve a su trabajo y lo tome cerca de su casa. Algo similar pasa cuando una persona cambia de casa, al ir a un vecindario nuevo es necesario que ubique las oficinas de los servicios básicos, como energía eléctrica y agua potable, para ejercer sus contratos y trámites. Para estos últimos ejemplos la restricción es minimizar la distancia del servicio al cliente. Incluso al diseñar una casa, una escuela, un edificio, etc., hay que considerar la Localización de Servicios.

Muchos problemas de Localización de Servicios, desde el punto de vista geométrico, han tenido su origen en áreas de aplicación de Geometría Computacional como son: la robótica, el reconocimiento de patrones, los sistemas de información geográfica, etc. Es gracias a esto que actualmente la Geometría Computacional y la Localización de Servicios intercambian problemas y técnicas para su resolución. La versión clásica del problema de Localización de Servicios consiste en ubicar uno o varios servicios que satisfagan a un conjunto de clientes, tal que la ubicación sea óptima con respecto a ciertas restricciones.

De todos los artículos relacionados con la Localización de Servicios, la mayoría se enfoca a estudiar los problemas cuando los servicios son “demandados” por los clientes, tales como: escuelas, hospitales, centros comerciales, estaciones de bomberos y policías; sin embargo, existen también problemas de Localización de Servicios

cuando éstos no son deseados, por ejemplo plantas tratadoras de agua, plantas nucleares, etc. Por esto es necesario el desarrollo de algoritmos eficientes para determinar la ubicación óptima de servicios, teniendo en cuenta los requerimientos de los clientes y las limitantes del espacio.

1.3 Organización de la tesis

Este trabajo se desglosa en 5 capítulos. El capítulo 2 contiene definiciones útiles para el desarrollo de este trabajo. En el capítulo 3 se describen las técnicas utilizadas en los algoritmos de las soluciones obtenidas, técnicas de Optimización Geométrica y de división del plano. En el capítulo 4 se explican detalladamente los algoritmos obtenidos para dar solución a los problemas estudiados en este trabajo, y se incluye el análisis de complejidad en tiempo para cada algoritmo. Finalmente en el capítulo 5 se presentan problemas para trabajo a futuro y conclusiones de los resultados presentados en esta tesis.

2

Conceptos preliminares

En este capítulo se presentan definiciones que facilitarán la comprensión de los algoritmos correspondientes a las soluciones obtenidas para los problemas estudiados en este trabajo.

2.1 Definiciones

Nótese que para las definiciones dadas a continuación, se asume posición general, es decir, se considera que 3 rectas no se intersectan en un mismo punto, además durante este trabajo, todo se realiza sobre

el plano.

Definición 1. *Sea S una colección de objetos geométricos, el radio mínimo $r*$ de intersección de S, se define como el radio mínimo r tal que $S \bigcap D(r) \neq \emptyset$, donde $D(r)$ es el círculo con radio r que intersecta a todos los objetos.*

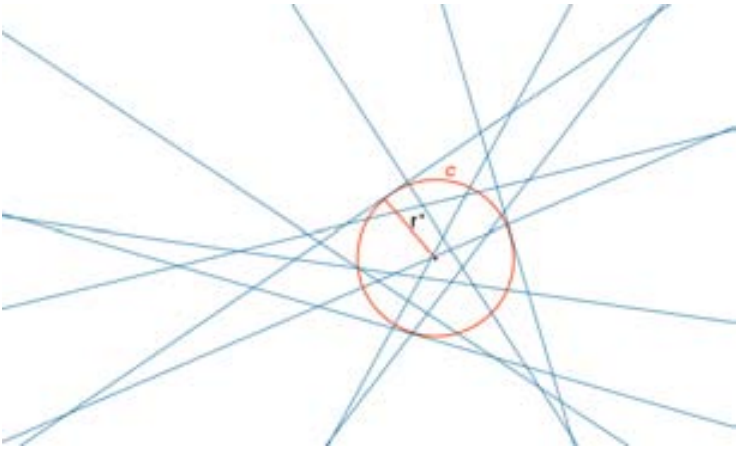

Figura 2.1: r^* es el radio mínimo tal que c intersecta a todas las rectas.

Un ejemplo es el de la Figura 2.1, el radio mínimo r^* intersecta a todas las rectas y es mínimo puesto que si se reduce, como se mues-

tra en la Figura 2.2, deja de intersectar a alguna de las rectas de la colección.

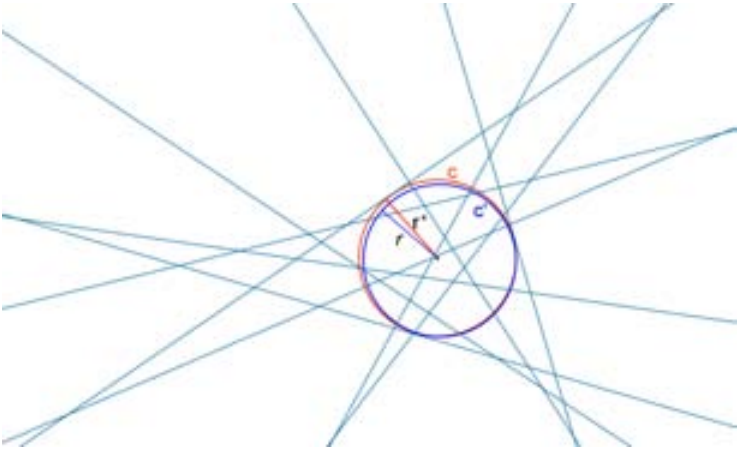

Figura 2.2: Como r^* es mínimo, al reducirse pierde una de las rectas.

Definición 2. *Sea L una colección de rectas en el plano en posición general y sea l una recta* $\in$ *L. Una banda B*(*l*) *es la región del plano acotada por dos rectas paralelas, tal que cada una se encuentra a distancia r de l.*

Nótese que el ancho de $B(l)$ es $2r$ y que un disco d de radio r tiene la propiedad de intersectar a l si y sólo si el centro de d está contenido

en $B(l)$ (ver Figura 2.3).

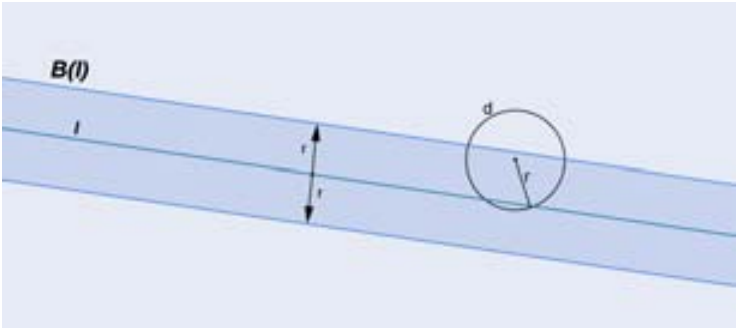

Figura 2.3: El círculo d intersecta a l si y sólo si su centro está contenido en la banda inducida por l que es $B(l)$.

Definición 3. *Una cara es una intersección de bandas, no vacía. Ésta puede ser representada por un subconjunto s de una colección L de rectas (ver Figura 2.4).*

Dada una cara c_a de ancho $2r$, se dice que una cara candidata a ser complementaria $\overline{c_a}$ es una región del plano con la propiedad de que si se coloca un círculo de radio r con centro en cualquier punto contenido en esa región, este círculo intersecta a todas las rectas que definen la cara.

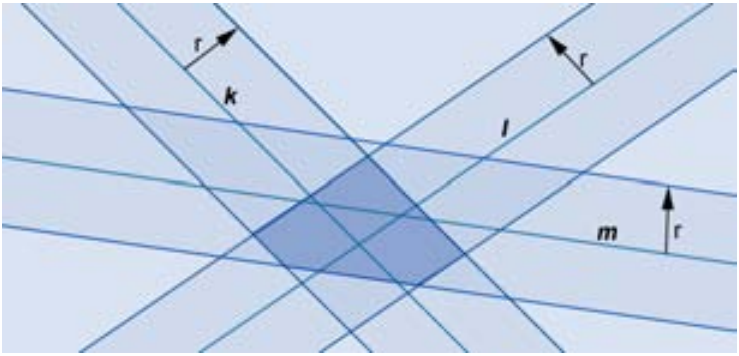

Figura 2.4: Cara representada como k,l,m.

Definición 4. *Una cara es complementaria de otra si y sólo si la unión de los subconjuntos que las representan contienen a todas las rectas de la colección L.*

Un ejemplo es el de la Figura 2.5, la cara $c(l_1, l_2)$ tiene dos caras candidatas a ser complementarias: $c(l_3)$ y $c(l_1, l_3)$, la $c(l_3)$ es la cara complementaria para $c(l_1, l_2)$.

2.2 Cortes

Para los resultados obtenidos en este trabajo de tesis se utilizan algunas herramientas de Geometría Computacional, una de ellas es el

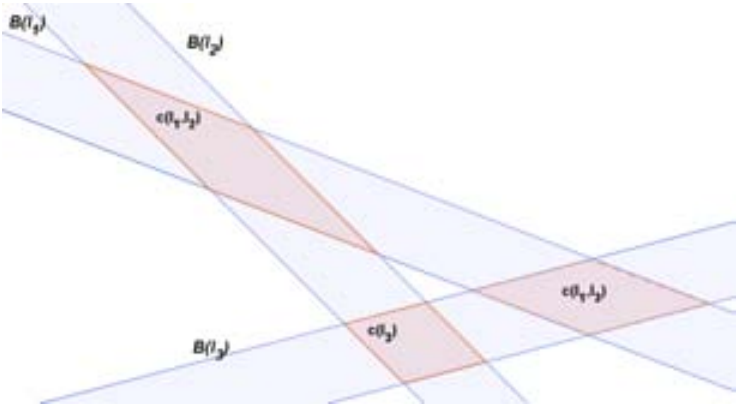

Figura 2.5: .

árbol de cortes, el cual es explicado en esta sección.

La idea de los árboles de cortes es muy similar a la de los árboles de partición, es decir, el plano es dividido en regiones (a veces triangulares) disjuntas. Sea L una colección de n rectas en el plano y sea r un parámetro de $1 \leq r \leq n$. Se dice que una recta cruza a un triángulo si intersecta el interior de éste. Un $(1/r)$-corte de L es un conjunto $\Xi(L) := t_1, t_2, \ldots, t_m$ de triángulos, posiblemente no acotados, con interiores disjuntos, que juntos cubren el plano y con la propiedad de que ninguno de los triángulos de la partición es cruzado por más de (n/r) rectas de L. El tamaño del $\Xi(L)$-corte es el número de triángulos

que lo integran. El corte de la Figura 2.6 tiene 6 rectas y 10 regiones triangulares, cada triángulo es cruzado por a lo más $(n/r) = (6/3) = 3$ rectas. Si el lector desea profundizar en el tema de cortes puede consultar el capítulo 16 de Mark de Berg *et al*, 2008, [10].

Figura 2.6: Un $(1/2)$-corte de tamaño 10 para un conjunto de 6 líneas.

Teorema 1. *Para cualquier conjunto L de n líneas en el plano, y cualquier parámetro r con $1 \leq r \leq n$, siempre existe un $(1/r)$-corte de tamaño $O(r^2)$. Además, tal corte (incluyendo el subconjunto de*

líneas de L que cruza cada triángulo) es construido en $O(nr)$ tiempo de ejecución.

La prueba de este teorema se puede estudiar ampliamente en los artículos de B. Chazelle, 1993 y J. Matoušek, 1992, [25, 8].

La estructura de datos basada en cortes se llama árbol de cortes. Para un conjunto L de n rectas se construye de la siguiente manera:

- Si la cardinalidad de L es 1, el árbol de cortes tiene una hoja única que almacena L. El conjunto L es el *subconjunto canónico* de la hoja.

- De otra forma, la estructura es un árbol T. Existe una correspondencia de uno a uno entre los hijos de la raíz del árbol y los triángulos de un $(1/r)$-corte $\Xi(L)$ para el conjunto L, donde r es una constante suficientemente larga, que después se explicará cómo elegir. El triángulo de un corte que corresponde a un hijo v se denota por $t(v)$. El subconjunto de rectas en L que caen abajo de $t(v)$ se llama *subconjunto canónico inferior* de v; el cual se denota como $L^-(v)$. El subconjunto de rectas en L que cae arriba de $t(v)$ es llamado *subconjunto canónico superior* de v y se denota como $L^+(v)$. El subconjunto de rectas que cruzan $t(v)$ se llama *subconjunto cruce* de $t(v)$. El hijo v es la raíz de un árbol de particiones, recursivamente definido en su

subconjunto de cruce; este subárbol se denota como T_v.

- En cada hijo v se guarda un triángulo $t(v)$. Además se guarda la información de los subconjuntos canónicos inferior $L^-(v)$ y superior $L^+(v)$; para contar la cantidad de rectas debajo de un punto de consulta, sólo es necesario guardar la cardinalidad del conjunto $L^-(v)$, pero para otra aplicación puede ser necesario guardar otra información.

La intención de explicar el cómo se construyen los árboles de cortes, es porque esta estructura de datos es utilizada para los algoritmos obtenidos en este trabajo. En el siguiente capítulo se analiza la herramienta esencial, la cual es utilizada en el resultado principal de esta tesis; la técnica de gráficas expansoras que pertenece a la Optimización Geométrica

3

Optimización Geométrica

3.1 Introducción

Una parte sustancial de la Geometría Computacional trata con el diseño de algoritmos eficientes, a veces óptimos, para problemas dados. La Optimización Geométrica estudia problemas relacionados con objetos geométricos, de tal forma que se cumplan ciertos criterios y restricciones. Sus técnicas pueden ser aplicadas en algoritmos de Geometría Computacional con el objetivo de mejorar el tiempo de ejecución.

Generalmente un problema de Optimización Geométrica involucra un número constante de variables y un número grande de restricciones que son inducidas por una colección dada de objetos geométricos. En estos casos, se espera que se desarrollen algoritmos simples y rápidos explotando la naturaleza geométrica del problema. Durante los últimos 20 años se ha realizado mucho trabajo en problemas de Optimización Geométrica, ha habido valiosos intentos por desarrollar técnicas que puedan ser aplicadas a problemas de este tipo y es así que han surgido varias ideas elegantes y sofisticadas.

Algunas de las técnicas utilizadas comúnmente en problemas de Optimización Geométrica son: búsqueda paramétrica, que puede consultarse en el trabajo de Megiddo, 1979 y 1983, [27, 26]; gráficas expansoras, las cuales pueden consultarse en los trabajos de Katz y Sharir, 1993, [24]; Katz, 1995, [23]; Ajtai y Megiddo, 1996, [2] y Sharir, 1997, [34]; cortes geométricos, de los que se habla en los trabajos de Agarwal *et al*, 1993, [5] y Chazelle, 1994, [7]; y búsqueda en matrices ordenadas, que se explica detalladamente en los trabajos de Frederickson, 1991, [16]; Frederickson y Johnson, 1982, 1983, 1984, [17, 18, 19] y Glozman *et al*, 1998, [20]; entre otras. En este capítulo se explican las técnicas que sirvieron como herramientas para la resolución del problema y sus variantes planteadas en este trabajo de tesis.

3.2 Técnicas de Optimización Geométrica

En 1979, Megiddo, [27, 26], propuso una ingeniosa técnica de Optimización Geométrica llamada búsqueda paramétrica, existen versiones preliminares para la búsqueda paramétrica, [15], pero la versión completa es la de Megiddo. Esta técnica fue motivada por problemas de optimización paramétrica en optimización combinatoria aunque no recibió mucha atención por la comunidad de Geometría Computacional hasta finales de los 80's; sin embargo, recientemente ha sido aplicada a varios problemas de optimización obteniendo soluciones eficientes, por lo que es muy popular en la Optimización Geométrica.

Concurrentemente al desarrollo de la búsqueda paramétrica, Megiddo, [28, 29] creó otra ingeniosa técnica para resolver problemas de programación lineal y de optimización. Esta técnica, ahora conocida como *reducir-y-buscar*, fue definida y extendida tiempo después por Dyer, [12], Clarkson, [13], y otros. Esta técnica puede ser vista como la versión optimizada de la búsqueda paramétrica, en la cual ciertas propiedades del problema, permiten mejorar la eficiencia del algoritmo.

La idea general de la búsqueda paramétrica es suponer que se tiene un problema de decisión $P(\mu)$ que es monótono en μ, esto significa

que si $P(\mu_0)$ se cumple, entonces $P(\mu)$ se cumple para toda $\mu < \mu_0$. Además $P(\mu)$ recibe como entrada n objetos de datos y un parámetro real μ. Se desea encontrar el valor óptimo (mínimo o máximo) μ^* del parámetro μ tal que $P(\mu)$ satisface ciertas propiedades. Por ejemplo, tomando el enfoque de los problemas estudiados en este trabajo, se desea encontrar el valor mínimo μ^* tal que $P(\mu)$ satisface que la unión de los dos círculos con radio μ^* intersecta a todos las rectas de una colección. Supóngase, también, que se tiene un algoritmo secuencial y eficiente A_s para resolver $P(\mu)$ dado cualquier valor μ, y que por el resultado el algoritmo A_s determina si el valor μ es igual, menor que o mayor que el deseado μ^*. Se asume además, que el flujo de la ejecución de A_s depende de comparaciones, cada una de las cuales es resuelta probando el signo de un polinomio de grado menor en μ y los n objetos de datos de entrada.

Ahora, la técnica de Megiddo ejecuta el algoritmo A_s genéricamente sobre un intervalo I que contiene el valor de μ^*, sin especificar el valor de μ, esto con la intención de simular su ejecución con el valor desconocido de μ^*. Cada vez que se realiza una comparación, se calculan las raíces; $r_1, r_2, \ldots$; del polinomio asociado y se ejecuta A_s con cada raíz, determinando así, la ubicación de μ^* entre las raíces y con esto se obtiene el signo del polinomio con μ^*, lo que determina la salida de la comparación para μ^*. Si una de las r_i es igual a μ^*,

el algoritmo se detiene porque el valor de μ^* ha sido encontrado. De otra manera, se obtiene un intervalo menor al anterior que se sabe que contiene el valor de μ^*. De esta manera el intervalo generado es cada vez más restringido, así se obtiene una secuencia de intervalos progresivamente menores, donde cada uno contiene el valor de μ^*. Esto se hace hasta encontrar el valor de μ^* en una de las comparaciones o hasta obtener un intervalo final con una cantidad lineal de posibles valores para μ^*.

Si A_s se ejecuta en tiempo T_s y realiza C_s comparaciones, el costo del procedimiento, en general, es $O(C_s T_s)$, lo que es usualmente cuadrático. Megiddo propuso implementar el algoritmo genérico con un algoritmo paralelo A_p, esto bajo el modelo de cómputo de Valiant que puede consultarse en [35]. Entonces si A_p utiliza P procesadores y se ejecuta en T_p pasos paralelos, cada paso paralelo involucra, a lo más, P comparaciones independientes; es decir, no es necesario conocer el resultado de cada comparación para ejecutar otras comparaciones en la misma ejecución. Entonces es posible calcular las $O(P)$ raíces de todos los polinomios asociados con estas comparaciones y hacer una búsqueda binaria para localizar μ^* en estos valores, utilizando A_s en cada paso de la búsqueda binaria. El costo por la simulación de un paso paralelo de A_p es $O(P + T_s \log P)$, no es necesario ordenar las $O(P)$ raíces porque se utiliza la búsqueda de mediana repetida

que tiene un costo de $O(P)$, esto da un tiempo total de ejecución de $O(PT_p + T_pT_s \log P)$.

La búsqueda paramétrica confía, crucialmente, en la disponibilidad de la versión paralela del algoritmo para el problema de decisión, y esto no siempre se tiene, depende de la geometría del problema a estudiar. Incluso a veces, aunque sí es posible paralelizar el algoritmo, no se logra una mejora en el tiempo de ejecución del algoritmo que soluciona el problema, sin utilizar la búsqueda paramétrica, es por eso que como alternativa a la búsqueda paramétrica, surgieron otras técnicas de Optimización Geométrica, tal es el caso de las matrices ordenadas y las gráficas expansoras. Además estas técnicas tienen algunas ventajas sobre la búsqueda paramétrica como que no requieren paralelización de algoritmo para el problema de decisión y que usualmente obtienen resultados más eficientes a los obtenidos con búsqueda paramátrica.

La técnica de matrices ordenadas sigue la idea general de la búsqueda paramétrica pero asume que el conjunto de todos los valores potenciales que puede tener μ^* es presentado en una matriz ordenada mxn y que hay un algoritmo de decisión A para $P(\mu)$, el cual decide en tiempo T si μ es igual, menor o mayor que el valor deseado μ^*.

La técnica de gráficas expansoras es la que se utiliza en el resultado principal de este trabajo, por esto, en la siguiente sección se

explican a detalle su funcionamiento y sus ventajas sobre otras técnicas.

3.3 Gráficas expansoras

Las gráficas expansoras fueron definidas inicialmente por Bassalygo y Pinkster, su existencia fue probada por Pinkster al principio de los 70s. La propiedad de una gráfica expansora parece importante en los contextos matemáticos, computacionales y físicos. Por ello no es de sorprender que las gráficas expansoras sean útiles en el diseño de redes de comunicación. Lo que es menos obvio es que las gráficas expansoras tienen utilidad en otras áreas como en la teoría de corrección de errores y la teoría pseudoaleatoria.

En Combinatoria, las gráficas expansoras están fuertemente conectadas; para desconectar una gran parte de la gráfica, es necesario eliminar muchas aristas. De igual manera usando la noción geométrica de Isoperimetría, cada conjunto de vértices tiene, relativamente, una vecindad grande. Desde el punto de vista probabilístico, se considera el camino aleatorio de una gráfica, en el cual se tiene un token en un vértice, que se mueve en cada paso a otro vértice vecino al azar, elegido de manera uniforme e independiente. Las expansoras son gráficas para las cuales este proceso converge a su distribución

límite.

Algebraicamente, consideramos el operador de Laplace en la gráfica y su expectro. Desde esta perspectiva, en las gráficas expansoras, el primer valor propio positivo (de su operador de Laplace) está acotado a un valor muy lejano de cero.

La técnica utilizada en el algoritmo obtenido, está basada en gráficas expansoras, las cuales son construidas sobre ciertos subconjuntos de los objetos de entrada. Puesto que las gráficas expansoras pueden ser construidas de una forma explícitamente determinística, [3], la técnica utlilizada en este trabajo es determinística. Incluso, esta técnica es conceptualmente más simple, a diferencia de la búsqueda paramétrica, no requiere paralelización y tiene una clara interpretación geométrica. Además, evita realizar algunas de las comparaciones genéricas que son ejecutadas en aplicaciones estándares de la búsqueda paramétrica. Su análisis confía en una propiedad simple y muy conocida de gráficas expansoras, la cual establece, que para cada par suficientemente grande de conjuntos de vértices, la gráfica expansora contiene una cantidad suficientemente grande de aristas entre ellos.

Ahora se dan las definiciones y propiedades necesarias para aplicar la técnica de gráficas expansoras.

Definición 5. *Una gráfica* $G = (V, E)$ *es una* (n, d, c)*-expansora si*

tiene n vértices, su grado es d y para todo conjunto de vértices $W \subset V$ de cardinalidad $|W| \leq n/2$, $|N(W)| \geq c|W|$, donde $N(W)$ es el conjunto de vértices en $V \backslash W$ que están conectados a W por una arista de G.

La siguiente propiedad es probada en [3], Capítulo 9, Corolario 2.2.

Lema 1. *Si G es una gráfica d-regular con n vértices y λ es el segundo valor propio más grande en la matriz de adyacencia de G, entonces G es una (n, d, c)-expansora con $c = (d - \lambda)/2d$. Así, si λ es mucho menor que d (que es el mayor valor propio de la matriz de adyacencia de G) entonces G es una buena expansora.*

Lubotzky, Phillips y Sarnak [20] (independientemente Margulis [21]) han dado una descripción explícita de una gráfica d-regular G con n vértices para la cual $\lambda \leq 2\sqrt{d-1}$, para cualquier $d = p + 1$ y $n = q + 1$, donde p y q son primos congruentes a 1 módulo 4. Estas gráficas en realidad tienen una propiedad más estricta, todos sus valores propios (excepto d) tienen valor absoluto menor a $2\sqrt{d-1}$. Estas gráficas son referidas como *LPS*-expansoras. De la descripción en [20] se sigue que siempre y cuando d sea una constante, una *LPS*-expansora de grado d con n vértices es construida en $O(n)$ tiempo de ejecución.

El siguiente lema trata de la propiedad principal de las *LPS*-

expansoras que se requiere en este trabajo de tesis, y es probada en [3], Capítulo 9, Corolario 2.5.

Lema 2. *Sea $G = (V, E)$ una gráfica d-regular con n vértices. Asuma que el valor absoluto de todos sus valores propios, con excepción del más grande es a lo más λ. Entonces, para cada dos conjuntos de vértices, A y B, de cardinalidad a y b, respectivamente, se tiene $|e(A, B) - abd/n| \leq \lambda\sqrt{ab}$, donde $e(A, B)$ es el número de aristas de G conectando un vértice de A con un vértice de B.*

Corolario 1. *Si G es una LPS-expansora de grado d con n vértices, y A y B son dos conjuntos de vértices con cardinalidades a y b, respectivamente, tal que $ab \geq 9n^2/d$, entonces $e(A, B) \geq 3n$.*

Demostración. El lema anterior, y el hecho de que $\lambda < 2\sqrt{d}$, implican que $e(A, B) \geq abd/n - 2\sqrt{abd} \geq 3n$, como es fácilmente comprobable. □

Los párrafos anteriores contienen los conceptos principales de las gráficas expansoras, técnica que es utilizada en uno de los algoritmos que se explican minuciosamente en el siguiente capítulo.

4

Resultados

En este capítulo se explican los problemas que se plantearon y resolvieron en este trabajo de tesis. Asimismo, se explican a detalle los algoritmos obtenidos para resolver estos problemas. Una parte importante de este trabajo es la complejidad de los algoritmos obtenidos, por lo que para cada algoritmo se incluye su análisis de complejidad.

4.1 El problema

Supóngase que se tiene un conjunto de trenes y que cada uno de ellos tiene una ruta, así como dos antenas que transmiten datos a los trenes. Se desea que eventualmente cada tren reciba la señal de una antena y que la potencia sea mínima puesto que un incremento de potencia implica un incremento al costo de cada antena. Por lo tanto, es necesario preguntarse ¿dónde deben colocarse las antenas? ¿Cuál debe ser el radio mínimo de potencia que cada una requiere, para que todos los trenes reciban señal? Ahora de manera independiente, supóngase que se conocen las trayectorias por las cuales pasará un conjunto de satélites, además se tienen dos estaciones receptoras que recolectan información de los satélites, es necesario que la potencia de las estaciones sea mínima para reducir sus costos.

Estos dos problemas pueden ser modelados de la misma manera en términos de Geometría Computacional. Se tiene un arreglo de rectas en el plano en posición general y se desean encontrar dos círculos del mismo radio, tal que su unión intersecta a todas las rectas y cuyo radio es minimizado. Las rectas representan las rutas de los trenes o satélites y los círculos representan las antenas o estaciones receptoras.

A partir de la idea anterior surgió el problema planteado para este trabajo de tesis, el reto era dar un algoritmo óptimo que lo resolviera.

El problema de este trabajo queda enunciado de la siguiente manera:

Problema: *Dada una colección L de n rectas en el plano en posición general, determinar los dos círculos, cuya unión intersecta todas las rectas en L y el radio del círculo más grande es minimizado (de ahora en adelante problema de* minimax*).*

Inicialmente se obtuvo un algoritmo trivial que se ejecuta en tiempo $O(n^4)$, después de analizar otras ideas y técnicas que pudiera ser de utilidad, se obtuvo un algoritmo que toma tiempo $O(n^2 \log^3 n)$ de ejecución.

Para el problema de encontrar sólo un círculo con radio mínimo que intersecta a una colección de rectas en el plano, existe una solución que utiliza programación lineal. En ella se determina el tamaño del radio y la ubicación del centro del círculo, este resultado se puede consultar en el trabajo de Binay K. Bhattacharya *et al*, [6]. A partir de esta referencia se analizaron las propiedades geométricas de la colección de rectas y los círculos deseados. Se observó que el círculo con el radio buscado está definido por 2 ó 3 puntos. Si el círculo de menor radio es definido por tres puntos, entonces el círculo de menor radio está inscrito en el triángulo que forman las tres rectas que contienen a los puntos (ver Figura 4.1). La aseveración anterior queda asentada

con el siguiente lema.

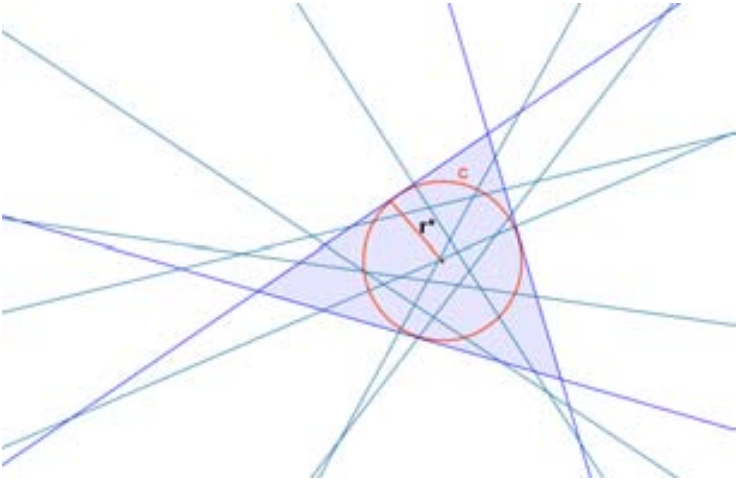

Figura 4.1: El círculo de menor radio está inscrito en un triángulo.

Lema 3. *Sea C_i el círculo de mayor radio inscrito en un triángulo T_i, de los generados por las intersecciones de una colección de rectas. Entonces C_i es el círculo de menor radio que intersecta a todas las rectas en L.*

Demostración. Supóngase que existe un círculo C_j de radio menor al de C_i que intersecta a todas las rectas. Sabemos que C_i está inscrito en

T_i. Sean k, l y m las rectas que definen a T_i. Dado que C_i está inscrito en T_i, C_i intersecta a las rectas k, l y m. Ahora, como el radio de C_j es menor al de C_i, no es posible que C_j esté inscrito en T_i, por lo que no intersecta, al menos, una de las rectas k, l o m.

Como C_j no intersecta a todas las rectas, existe al menos una recta t que no es intersectada por C_j, esta recta t, junto con dos de las rectas de T_i, definen un triángulo cuyo círculo inscrito $C_{i'}$ tiene un radio mayor al de C_j. Además $C_{i'}$ sí intersecta a t, por lo que C_j no es el círculo de radio mínimo que intersecta a todas las rectas (ver Figura 4.2). □

Con este lema, se concluye entonces, que el radio mínimo que se busca en el problema enunciado anteriormente, es el de un círculo inscrito en uno de los triángulos generados por las intersecciones de las rectas en L, de lo contrario, alguna de las rectas en L no sería intersectada.

Por lo anterior, el algoritmo trivial que resuelve el problema de *minimax*, comienza generando todos los círculos, que se encuentran inscritos en los triángulos generados por las intersecciones de las rectas, almacena estos radios y sobre ellos ejecuta una búsqueda exhaustiva de la siguiente manera:
Para cada triángulo generado:

1. Eliminar las rectas que el círculo, inscrito en este triángulo, intersecta.

2. Calcular la solución para el resto de rectas. Considerando que el resto de las rectas, es la configuración para la que se desea obtener el radio mínimo de intersección, esto se hace usando el

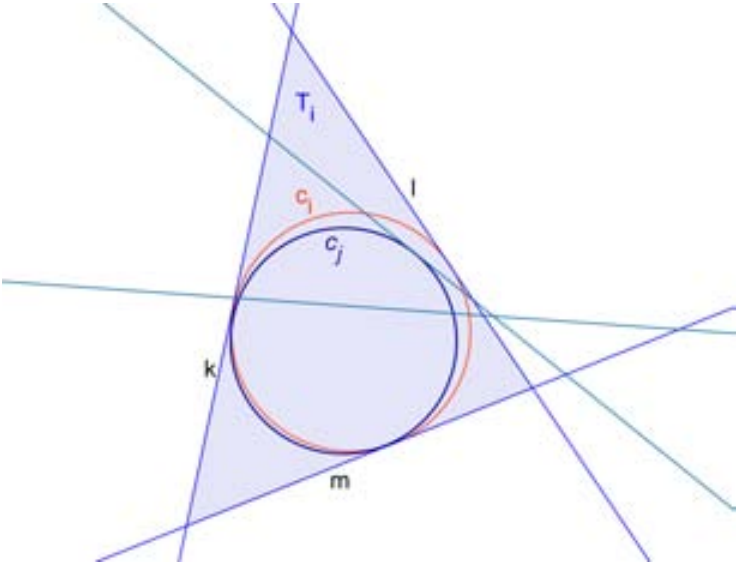

Figura 4.2: C_i es el círculo de menor radio que intersecta a todas las rectas.

algoritmo de *Bhattacharya* que se ejecuta en tiempo lineal; esta metodología se explica ampliamente en [6].

3. Comparar en cada iteración la solución obtenida, con la anterior y almacenar la menor.

Dado que el algoritmo calcula todos los radios que hay en la colección, es posible encontrar la pareja de círculos tal que su unión intersecta a todas las rectas de la colección. Teniendo esta solución, es posible tomar el menor de los radios y hacerlo crecer hasta que sea igual al de su radio en la pareja de círculos, con esto se resuelve el problema cuando se busca que el radio para los dos círculos sea el mismo.

Este algoritmo tiene una complejidad de $O(n^4)$. El paso 1 del algoritmo toma $O(n^3)$ puesto que existen $O(n^3)$ tripletas en la colección de rectas, esto genera $O(n^3)$ de círculos y por lo tanto de radios.

Por otro lado, la búsqueda secuencial sobre los $O(n^3)$ radios implica tomar cada uno de estos radios y calcular otro radio, que se realiza en tiempo lineal, el costo de esto es $O(n^4)$, que es el tiempo total de ejecución.

Debido a que este algoritmo realiza una búsqueda exhaustiva, su tiempo de ejecución es muy alto; por este motivo, se buscó un algoritmo más rápido, esto se hizo reemplazando la búsqueda exhaustiva

por una búsqueda binaria. Se observó que el resolver el siguiente problema de decisión, es útil para guiar la búsqueda binaria, al ejecutar éste tipo de búsqueda, se obtiene un mejor tiempo de ejecución para el algoritmo. En la siguiente sección se explica a detalle cómo resolver el problema de decisión y más adelante se explica el algoritmo que utilizada la solución.

4.2 Problema de decisión

Problema de decisión: *Dada una colección L de n rectas en el plano en posición general y dado un radio r, determinar si la unión de dos círculos con radio r, intersecta a todas las rectas en L.*

El algoritmo inicial que resuelve este problema tiene un tiempo de ejecución de $O(n^4)$ mientras que el algoritmo final se ejecuta en $O(n \log^2 n)$, la primera versión del algoritmo que resuelve el problema de decisión utiliza la definición 2 de banda, dada en el capítulo 2 de este trabajo. Asimismo, hace uso del siguiente lema.

Lema 4. *Un círculo de radio r intersecta a una recta l si su centro está contenido en la banda B(l), generada a partir de l.*

Demostración. Sea $B(l)$ la banda definida por la recta l con ancho $2r$ y sea c un círculo de radio r. Supóngase que el centro de c está contenido en $B(l)$ y no intersecta a l, si el centro de c es tangente a la frontera de $B(l)$, la distancia existente es exactamente r. Como el centro de c está contenido en $B(l)$, no puede estar más allá de la frontera de $B(l)$. Por lo tanto, si el centro de c está contenido en $B(l)$, el círculo intersecta a l. □

En el procedimiento del algoritmo se utilizan las definiciones 3 y 4 dadas en el capítulo 2, correspondientes a cara, cara candidata y cara complementaria, respectivamente. El lema anterior representa la propiedad principal del algoritmo, por lo que el algoritmo comienza generando bandas para las rectas de L de la siguiente manera:

1. Por cada recta l_i de L, dibujar una paralela a distancia r de l_i en cada uno de los semiplanos definidos por l_i. La región acotada que se genera al colocar estas rectas es la banda generada a partir de l_i y se denota como $B(l_i)$ y la colección de bandas generadas a partir de L se denota como $B(L)$.

2. Por cada cara de las generadas, determinar si existe una cara complementaria, explorando todas las caras.

Existen $O(n^2)$ caras y $O(n^2)$ caras complementarias. Por cada una de las $O(n^2)$ caras se busca su cara complementaria en las $O(n^2)$ caras

complementarias, por lo que el tiempo total de ejecución de este algoritmo es de $O(n^4)$.

La idea de generar bandas a partir de la colección de rectas es muy útil, por lo que también es utilizada en la segunda versión del algoritmo para resolver el problema de decisión, la cual considera las bandas generadas a partir de la colección de rectas, en el paso 1 del procedimiento del algoritmo anterior se explica cómo generarlas.

Inicialmente se toma una de las $O(n^2)$ caras existentes y se ignoran las bandas que definen tal cara. Del conjunto de bandas restantes se toman las rectas que generan estas bandas y se aplica el algoritmo de *Bhattacharya*, [6]; como se mencionó antes, este algoritmo determina en tiempo lineal el radio mínimo del círculo que intersecta a una colección de rectas. Si el radio resultante de aplicar el algoritmo de Bhattacharya es menor o igual a r, existe un círculo de radio r que intersecta a este subconjunto de rectas, entonces el algoritmo de decisión responde que dos círculos con radio r sí intersectan a la colección de rectas inicial. Una vez que este procedimiento se repite para todas las $O(n^2)$ caras, es posible determinar si la unión de dos círculos con radio r intersecta o no a la colección de rectas inicial.

Es claro que este algoritmo toma $O(n^3)$ en tiempo de ejecución puesto que por cada una de las $O(n^2)$ caras se aplica el algoritmo de *Bhattacharya* que se ejecuta en tiempo lineal.

Por otro lado, una banda puede verse como la intersección de dos semiplanos (ver Figura 4.3). Considerando esta representación es cla-

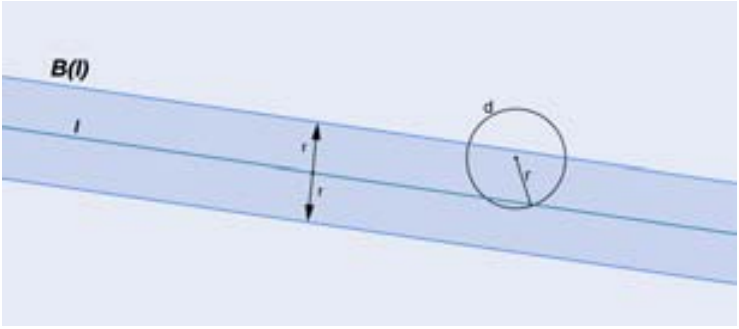

Figura 4.3: Una banda es representada como la intersección de dos semiplanos.

ro que si la intersección de *todos* los semiplanos que definen a la colección de bandas no es vacía, significa que existe un círculo con radio r que intersecta a todas las rectas de la colección.

Existe una estructura de datos que mantiene *dinámicamente* la intersección de una colección de semiplanos. Esta estructura de datos se construye en $O(n \log^2 n)$, la operación de agregar un semiplano toma $O(\log^2 n)$ y la de eliminar $O(\log^3 n)$, los detalles que se refieren a esta estructura de datos se pueden consultar en el trabajo de Overmars y Leeumen, 1981, [33].

Observe que el problema de determinar si un solo círculo con radio r intersecta a todas las rectas de la colección, se resuelve utilizando la estructura de datos mencionada, con esto el problema es resuelto en $O(n \log^2 n)$. Al extender esta idea, se obtiene el algoritmo final para resolver el problema de decisión para dos círculos.

4.3 Algoritmo para el problema de decisión

Este algoritmo también utiliza la idea de generar bandas, la cual fue explicada anteriormente. En una de las bandas generadas por la colección de rectas se coloca un círculo con radio r tal que su circunferencia es tangente a la frontera de la banda y todas las intersecciones con otras bandas quedan a la derecha. El círculo es deslizado sobre la frontera de la banda hasta que toque o deje de tocar a otras bandas. Se ignoran las bandas que el círculo toca y se pregunta por la intersección de las bandas restantes, representándolas como intersección de semiplanos, esto con ayuda de la estructura de datos que mantiene dinámicamente la intersección de semiplanos.

Este proceso se repite con el mismo r para cada banda en la colección. Entonces como se mencionó antes, si: $\bigcap_{i=1}^{n} B(l_i) \neq \emptyset$ significa

que existe un círculo de radio r que intersecta a todas las bandas y por lo tanto a las rectas. El algoritmo final, procede como sigue:

Por cada banda $B(l_i) \in B(L)$:

1. Asignar una orientación arbitraria a $B(l_i)$.

2. Colocar un círculo c de radio r que sea tangente a la frontera de $B(l_i)$, tal que c intersecta sólo a $B(l_i)$ y todas las bandas que intersecta $B(l_i)$ están a su derecha. Sea $B(S) \subset B(L)$ el conjunto de bandas $B(s_i)$ que no son intersectadas por c, claramente en este punto $B(S) = B(L) \setminus B(l_i)$ (ver Figura 4.4).

3. Tomar el círculo c y deslizarlo a la derecha sobre la banda $B(l_i)$, cuidando que sea tangente a la frontera de $B(l_i)$, cuando c sea tangente a una banda $B(l_j) \in B(L)$ se deja de mover el círculo. Si $B(l_j) \notin c$ se hace $B(S) = B(S) \cup B(l_j)$; de otra manera $B(S) = B(S) \setminus B(l_j)$.

4. Verificar $\bigcap_{i=1}^{n} B(s_i)$, si no es vacía entonces existen dos círculos con radio r cuya unión intersecta todas las rectas en L, si es vacía se continua deslizando el círculo c sobre $B(l_i)$. Utilizando la estructura de datos mencionada para mantener $\bigcap_{i=1}^{n} B(s_i)$ de manera eficiente.

Una vez que el círculo ha sido posicionado y deslizado sobre las n bandas, es posible saber si existen dos círculos de radio r, cuya unión intersecta todas las rectas en L.

Mientras el círculo c está siendo deslizado sobre $B(l_i)$, la banda $B(l_j) \in B(L)$, $B(l_i) \neq B(l_j)$ se agrega o elimina de $B(S)$ exactamente una vez. Entonces por cada banda $B(l_i)$ se realiza un número lineal de actualizaciones. Por lo anterior al ejecutar este algoritmo, es posible

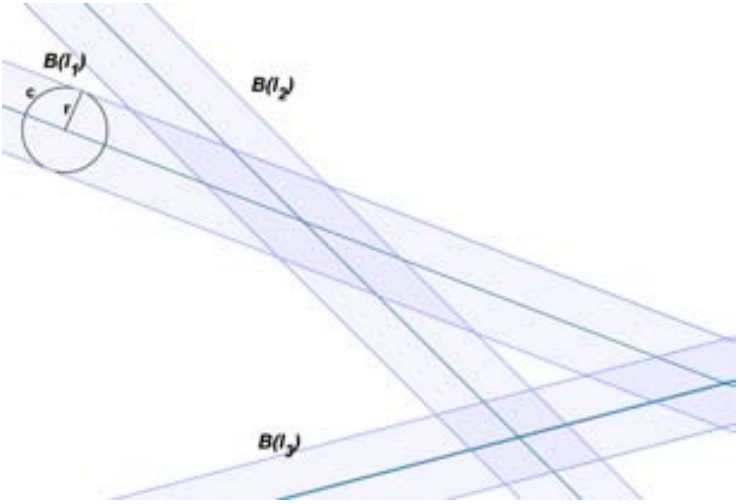

Figura 4.4: El círculo c es deslizado sobre la banda $B(l_1)$.

decidir si hay dos círculos de radio r cuya unión intersecta L. Con este algoritmo se mejora el tiempo de ejecución para resolver el problema de decisión, el análisis de complejidad se explica con el siguiente teorema.

Teorema 2. *El problema de decisión que determina si la unión de dos círculos, con radio r, intersecta a una colección L de n rectas, se puede resolver en tiempo* $O(n^2 \log^3 n)$.

Demostración. Como se mencionó antes, la estructura de datos que mantiene la intersección de semiplanos dinámicamente se construye en $O(n \log^2 n)$, mientras que la operación de agregar un semiplano toma $O(\log^2 n)$ y la de eliminar un semiplano $O(\log^3 n)$.

El círculo c realiza $O(n^2)$ paradas puesto que hace una parada en cada una de las $O(n^2)$ intersecciones y en el peor de los casos es posible que en cada parada elimine una banda que en la estructura de datos es eliminar dos semiplanos, esto toma $O(2 \log^3 n)$, por lo que el tiempo total de ejecución total es de $O(n^2 \log^3 n)$. □

El tiempo de ejecución probado en el teorema anterior es el mejor obtenido en este trabajo para el problema de decisión. El algoritmo siguiente utiliza esta solución en su procedimiento.

4.4 Algoritmo que resuelve el problema de *minimax*

Este algoritmo resuelve el problema de *minimax* y utiliza una técnica de Optimización Geométrica, la cual está basada en gráficas expansoras, esta técnica se explicó en el capítulo 3 de este trabajo. La idea de la solución es utilizar las gráficas expansoras para reducir el espacio de búsqueda en el que se encuentra el valor del radio deseado r^*, y realizar una búsqueda binaria sobre este espacio, obteniendo así el valor de r^*. La búsqueda binaria para determinar r^*, es guiada por el algoritmo para el problema de decisión que se explicó anteriormente. Entonces, la idea del algoritmo es realizar un tipo de búsqueda binaria sobre r, utilizando $A(r)$ como procedimiento discriminatorio, hasta que r^* sea encontrado.

En el capítulo 3 se mencionó que una propiedad conocida de las gráficas expansoras es que para cada par de conjuntos suficientemente grandes de vértices, la gráfica expansora contiene suficientes aristas entre ellos. Esta técnica tiene una clara interpretación geométrica y no requiere paralelización, además en comparación con otras técnicas de Optimización Geométrica, evita realizar comparaciones genéricas.

La solución se ejecuta en $O(\log n)$ etapas. Cada una de estas eta-

pas produce un intervalo cada vez más pequeño que contiene a r^* y con la propiedad adicional de que este j-ésimo intervalo I_j contiene a lo más $O(n^2)$ valores posibles de r^*.

La j-ésima etapa empieza por generar una representación compacta del conjunto de todos los círculos definidos por las tripletas de recta en L cuyo radio está en el intervalo I_{j-1}. Para obtener esta representación se ejecuta una *batched range searching*, técnica que es explicada más adelante; esta búsqueda genera una colección de gráficas bipartitas que son *reemplazadas* por gráficas expansoras. Las aristas de las gráficas expansoras son el espacio donde se realiza una búsqueda binaria utilizando el algoritmo de decisión para guiar tal búsqueda y con esto se obtiene un nuevo intervalo.

Después de $O(\log n)$ etapas, es posible determinar r^* ejecutando un número constante de comparaciones puesto que el último intervalo generado contiene pocos valores posibles de r^*.

El algoritmo procede en etapas; la j-ésima etapa produce un intervalo $I_j = (\alpha_j, \beta_j)$, se sabe que r^* está contenido en I_j, además se sabe que r^* no es más grande que el radio de la solución de un solo círculo, por esto el intervalo inicial es $I_0 = (0, R)$ donde R es el radio mínimo del círculo que intersecta todas las rectas en L.

La j-ésima etapa procede como sigue:

1. Por cada par de rectas $m, n \in L$ se trazan los bisectores. Por cada bisector (existen 4 de ellos), después se dibujan dos círculos $C_{m,n}$, $D_{m,n}$ de radio α_j y β_j respectivamente, tal que los centros están en el bisector y las fronteras de los círculos son tangentes a las rectas m y n (ver Figuras 4.5 y 4.6).

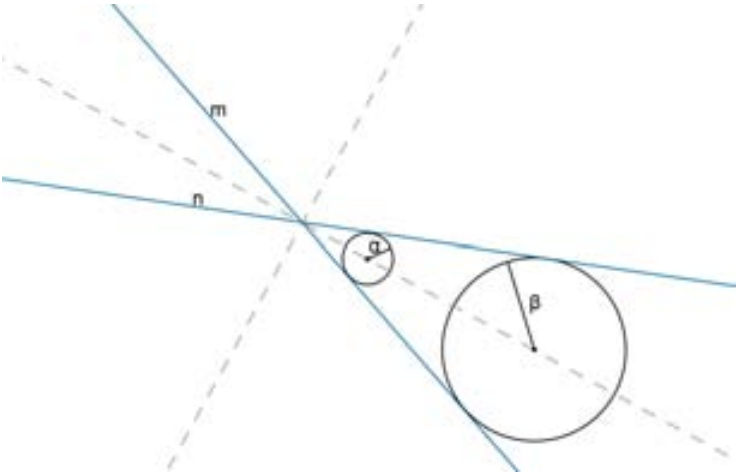

Figura 4.5: Círculos de radio α y β sobre bisector.

Observe que el círculo definido por las rectas que son tangentes a $C_{m,n}$, $D_{m,n}$ y una recta que corta a tales círculos, definen un círculo cuyo radio queda fuera del intervalo I_j. Al iniciar la ejecución del

algoritmo, es probable que los círculos α y β no se intersecten (ver Figura 4.7); pero conforme se genera un nuevo intervalo, la intersección entre los círculos se incrementa (ver Figura 4.8). Por esto las rectas de interés son las que cortan sólo a un círculo, ya sea $C_{m,n}$ o $D_{m,n}$, no a los dos al mismo tiempo. Se denota como $R_{m,n}$ a la diferencia simétrica de $C_{m,n}$ y $D_{m,n}$. Se distinguen dos casos por cada bisector, según si l intersecta $C_{m,n} - D_{m,n}$ o $D_{m,n} - C_{m,n}$. A continuación se muestra cómo obtener la colección de todos los pares de la forma $(R_{m,n}, l)$ donde l intersecta $C_{m,n} - D_{m,n}$, el otro tipo se obtiene de manera simétrica.

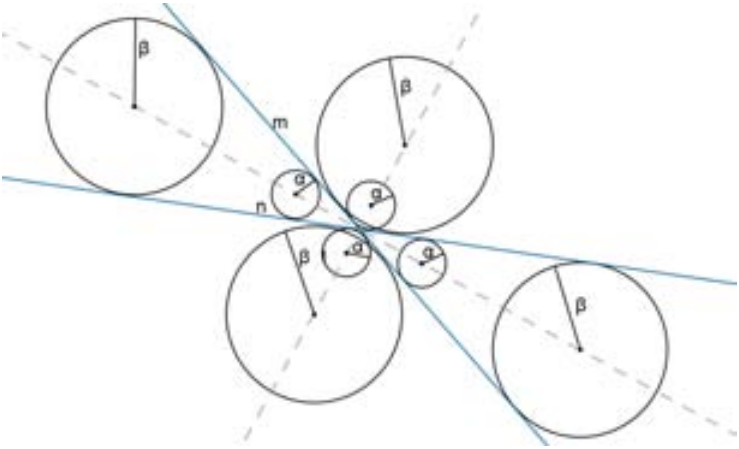

Figura 4.6: Círculos de radio α y β sobre bisectores de rectas m y n.

Se denota como C (respectivamente D) al conjunto de círculos $C_{m,n}$ (respectivamente $D_{m,n}$), y sea A el conjunto de bandas de ancho α generado por L. Se construye un $(1/n)$ corte para el conjunto A, se mantiene el árbol T que se obtiene en la construcción, donde los nodos del k-ésimo nivel del árbol representan las celdas del corte en el k-ésimo nivel en la jerarquía. Sea $\overline{C}$ el conjunto de centros de los círculos C. Se ejecutan $\binom{n}{2}$ consultas de localización de puntos en el corte final con los puntos de $\overline{C}$, en modo por lotes. Cada consulta es

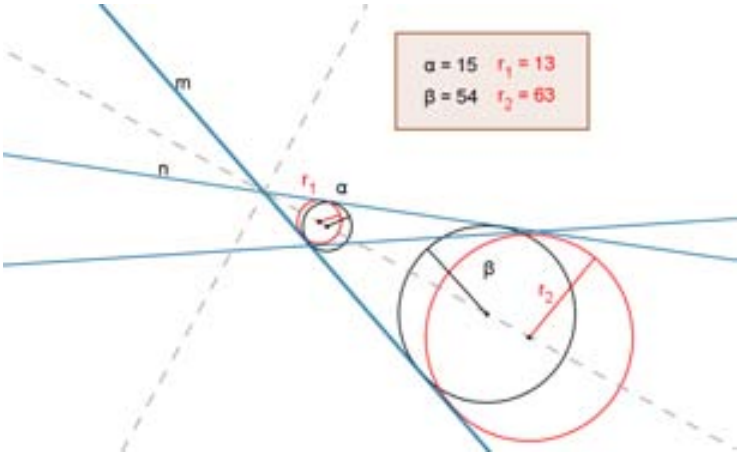

Figura 4.7: Los radios r_1 y r_2 quedan fuera del intervalo de búsqueda para $r*$.

ejecutada buscando con el punto de consulta a través de un camino apropiado de T, esto toma $O(\log n)$, por lo que todas las consultas toman $O(n^2 \log n)$. En el nodo v de T, excluyendo la raíz, se obtiene un par $(A_v, \overline{C_v})$ de conjuntos donde $A_v \subseteq A$. A_v es el conjunto de bandas cuyas fronteras cruzan la celda asociada con el padre de v y contienen totalmente la celda asociada con v, y $\overline{C_v} \subseteq \overline{C}$ es el conjunto de puntos de consulta pasando por v, esto es caen dentro de la celda asociada con v.

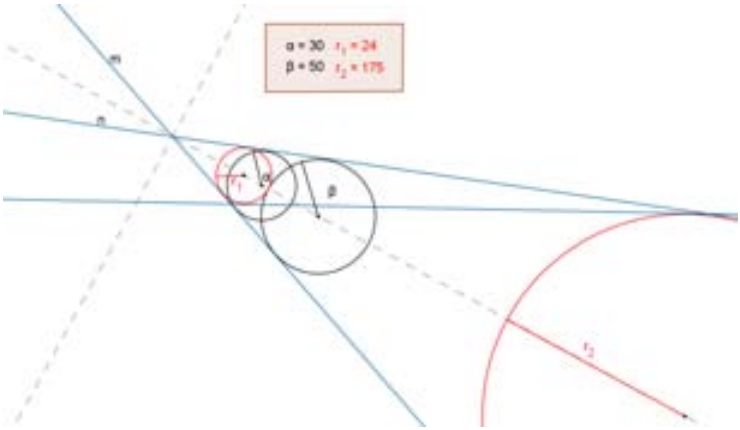

Figura 4.8: Los radios r_1 y r_2 quedan fuera del intervalo de búsqueda para $r*$.

Ahora se describe cómo ejecutar eficientemente la parte de *batched range searching*. En cada nodo v se tiene el siguiente subproblema:

Se denota como D_v el subconjunto de D que corresponde al subconjunto $\overline{C}_v$, esto es, el conjunto de círculos $D_{m,n}$, donde los centros de los círculos correspondientes $C_{m,n}$ pertenecen a $\overline{C}_v$. Se denota como $L_v \subseteq L$ el conjunto de rectas del conjunto de bandas A_v.

Se quiere reportar, en una forma compacta, todos los pares $(D_{m,n}, l)$, donde $D_{m,n} \in D_v$ y $l \in L_v$, tal que l no intersecta $D_{m,n}$. El siguiente algoritmo es aplicado en cada uno de estos subproblemas.

Sea S un conjunto de n bandas en el plano. Se preprocesa S para que dado un punto de consulta p, el conjunto de bandas de S que contienen p pueda ser encontrado rápidamente. El siguiente teorema de P. Awargal es aplicado, éste requiere pequeñas modificaciones puesto que en lugar se círculos se tienen bandas y se desea obtener las rectas que no intersectan $R_{m,n}$.

Teorema 3. *Sea C un conjunto de n bandas en el plano. Es posible construir, en* $O(n^2 \log n)$*, una estructura de datos que usa espacio de* $O(n^2 \log n)$*, tal que el conjunto de bandas que contienen a un punto de consulta en su interior pueda ser reportado en* $O(\log n)$*, como una colección de* $O(\log n)$ *conjuntos canónicos de parejas disjuntas.*

La estructura de datos es construida como se explica a continuación:

Para el arreglo $A(S)$ de las bandas en S, como el que se muestra en la Figura 4.9, se construye la gráfica dual G de $A(S)$; los nodos de G son las caras de $A(S)$ y sus aristas conectan pares de caras adyacentes a lo largo de una arista de $A(S)$ (ver Figura 4.10). Se calcula el árbol generador de G como en la Figura 4.11 y se duplica cada arista del árbol para obtener un camino euleriano π de G y se enumeran los nodos del camino euleriano para identificar segmentos, esto puede verse en la Figura 4.12. El número total de las aristas es $O(n^2)$ y el camino euleriano puede visitar una cara de $A(S)$ cualquier número de veces. Ahora se toma una banda $s \in S$, y se marcan todas las aristas de π que cruzan s. Existen solamente $O(n)$ de estas aristas, puesto que hay sólo $O(n)$ aristas en $A(S)$ que están contenidas en s, y cada una de ellas es cruzada por π a lo más dos veces, una en cada dirección. Las aristas marcadas generan una partición de π en subsecuencias, tal que la unión de las celdas de $A(S)$ en una sola subsecuencia está o totalmente contenida dentro de s o está totalmente fuera de s. Entonces se puede construir un árbol balanceado de segmentos T sobre los nodos de π, en orden a través de π, y representar cada banda por una colección de secuencias de caras de $A(S)$ que están en el interior de s, cada subsecuencia puede verse como un segmento de π, esto puede verse

en la Figura 4.13. Se almacena cada uno de estos segmentos, para cada banda $s \in S$, en $O(\log n)$ nodos de T de una forma estándar. La estructura de datos deseada consiste en el árbol de segmentos resultante T, además de una estructura de localización de puntos eficiente para $A(S)$, donde cada cara de $A(S)$ apunta a alguna hoja de T, esto es a un nodo de π. Es fácil ver que la estructura de datos requiere espacio de $O(n^2 \log n)$, y que puede ser construida en $O(n^2 \log n)$. Dado un punto de consulta p, se localiza la cara f de $A(S)$ que contiene a p, localizando la hoja de T a la que f apunta, se construye el camino

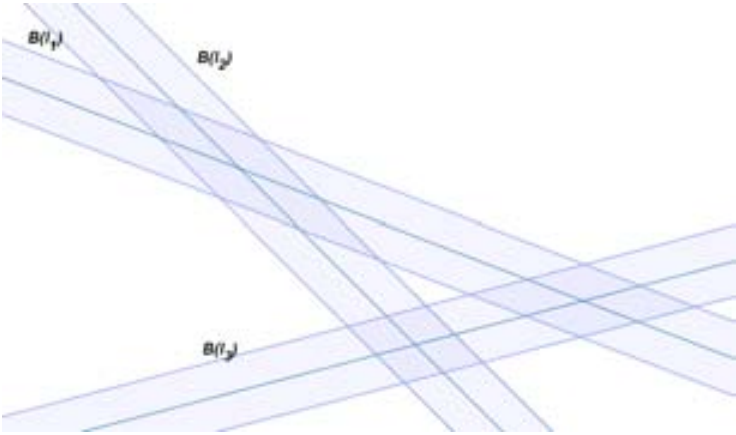

Figura 4.9: Bandas generadas a partir de la colección de rectas.

de T desde esta hoja a la raíz de T, se reporta el conjunto de bandas almacenado en cada uno de los $O(\log n)$ nodos a lo largo del camino. Es fácil ver que cada banda que contiene a p aparece exactamente en uno de estos conjuntos.

Ejecutar m consultas en la estructura de datos, se almacena p en cada nodo a lo largo del camino del árbol de segmentos trazado por la consulta. Después de ejecutar las m consultas, se produce la salida haciendo un recorrido por todos los nodos del árbol, y reportando, por cada nodo t, la gráfica bipartita completa formada entre el conjunto S_t

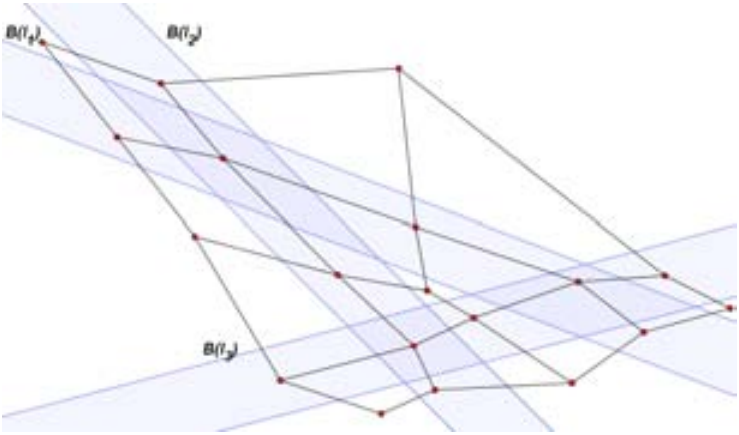

Figura 4.10: Gráfica dual de las bandas.

de bandas almacenado en t y el conjunto P_t de puntos almacenados en t, así se obtiene una colección de $O(m)$ gráficas bipartitas disjuntas en aristas $C_t \times P_t$, entonces, el costo del algoritmo es $O(n^2 \log n + m \log n)$.

Ahora se reportan las parejas deseadas para el subproblema en cada nodo v.

1. Se construyen las gráficas expansoras con la colección de parejas, sea G_t la unión de estas gráficas.

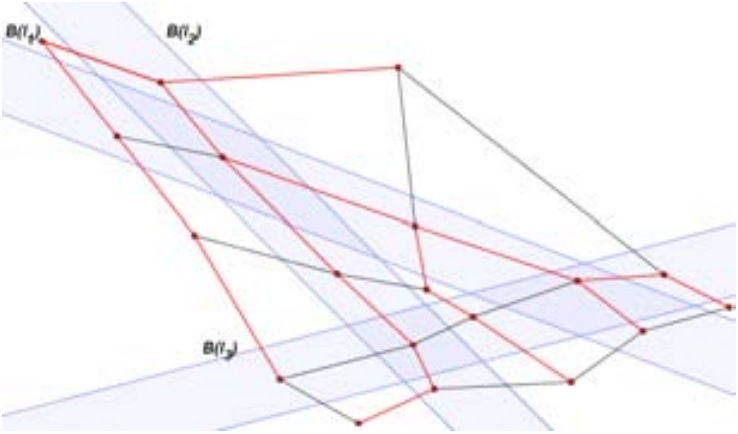

Figura 4.11: Árbol generador para gráfica dual.

2. Cada arista de G_t representa el radio del círculo tangente a las rectas definiendo S y la recta l.

3. Se ejecuta una búsqueda binaria sobre las parejas (L_t, R_t), utilizando el algoritmo de decisión para guiar la búsqueda, con esto se obtiene un nuevo intervalo $I_j = (\alpha_j, \beta_j)$ donde está r^*.

4. Con el nuevo intervalo I_j se repite el procedimiento completo hasta que el intervalo generado contenga pocos valores posibles de radios, tal que en un número constante de comparaciones se

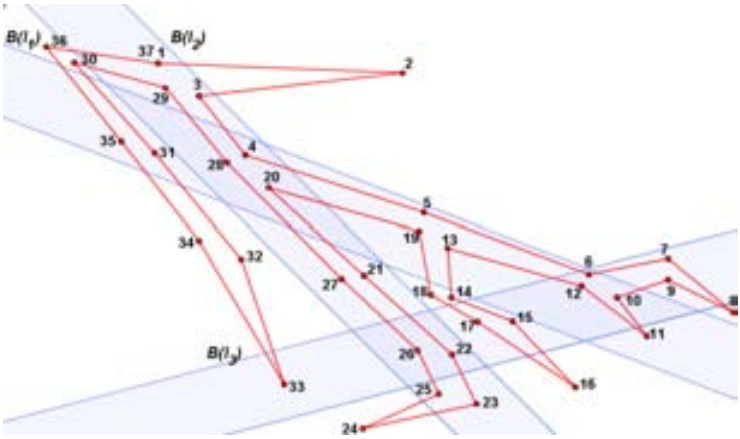

Figura 4.12: Camino euleriano numerado.

pueda determinar el valor de r^*.

Con el siguiente teorema se da el análisis del tiempo total de ejecución para el algoritmo anterior.

Teorema 4. *El algoritmo que utiliza la técnia basada en gráficas expansoras tiene una complejidad de* $O(n^2 \log^4 n)$.

Demostración. En la aplicación de la búsqueda de rangos se utiliza un árbol de cortes, cada nodo v representa un subproblema que consis-

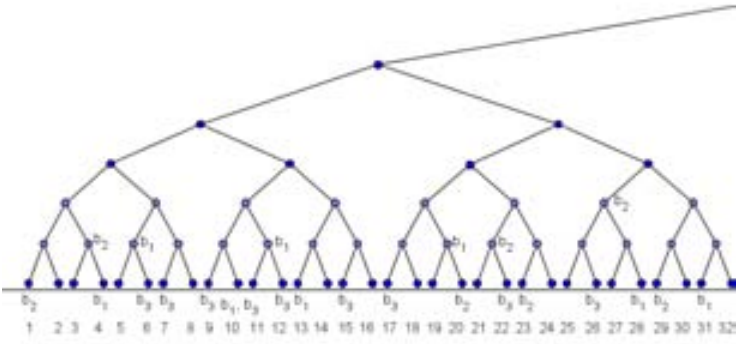

Figura 4.13: Árbol de segmentos correspondiente a camino euleriano.

te en reportar las parejas deseadas, esto toma $O(n_v^2 \log n_v + m_v \log n_v) = O(m \log n) = O(m \log m)$. El costo total de calcular las parejas para todos los nodos del k-ésimo nivel del árbol es $O(n^2 \log n)$, sumando todos los $O(n^2)$ nodos del árbol, entonces una sola aplicación de la búsqueda de rangos toma $O(n^2 \log^2 n)$ que además es el tamaño de las gráficas bipartitas. El algoritmo de decisión se ejecuta en $O(\log n)$, y este algoritmo cuesta $O(n^2 \log^3 n)$ entonces el tiempo total es $O(n^2 \log^4 n)$. □

Las gráficas expansoras sirven para podar el espacio de búsqueda, si existiera otra técnica, (de Geometría Computacional o de Optimización Geométrica) que permitiera podar de forma más eficiente el espacio de búsqueda, entonces se podría obtener un algoritmo con un mejor tiempo de ejecución.

Hasta aquí se presentan los resultados obtenidos para los problemas planteados en este trabajo de tesis, en el siguiente capítulo se hablará de las conclusiones y posibles extensiones de trabajo que genera este escrito.

5

Conclusiones y trabajo a futuro

En este trabajo se presentaron algoritmos para resolver una variante al problema del 2-centro. En lugar de puntos, esta variante considera una colección de rectas en el plano. El primer algoritmo propuesto resuelve el problema de la suma mínima de radios y tiene un tiempo de ejecución de $O(n^4 \log n)$, este algoritmo fue mejorado utilizando una técnica de Optimización Geométrica que se basa en gráficas expansoras, obteniendo una solución con un tiempo de ejecución de $O(n^2 \log^4 n)$.

Para el problema de decisión se obtuvo un primer algoritmo de $O(n^3 \log n)$ en tiempo de ejecución, esta solución fue mejorada y el

algoritmo obtenido tiene un tiempo de ejecución de $O(n^2 \log^3 n)$.

Un reto que se genera con este trabajo, es estudiar otras técnicas de Optimización Geométrica que puedan ser aplicadas a los problemas que se plantearon y resolvieron en este trabajo. Por ejemplo, la técnica de búsqueda paramétrica, la cual ha sido aplicada en muchos problemas geométricos, podría ser aplicada si se pudiera paralelizar el algoritmo de decisión. Además si esto fuera cierto, su complejidad sería mejor a la actual. Dicho resultado tendría un impacto directo en la complejidad del algoritmo obtenido para resolver el problema de suma mínima de radios; puesto que su complejidad está dada principalmente por el algoritmo de decisión. Entonces, un camino a tomar es paralelizar el algoritmo de decisión, podría ser con un número cuadrático de procesadores, como lo hace Meggido aplicando el cómputo paralelo en [26]; porque existen $O(n^2)$ intersecciones en la colección de rectas que a lo mejor podrían ser evaluadas de manera independiente.

Existen otras técnicas de Optimización Geométrica que pueden dar origen a otros resultados, una de ellas es la búsqueda paramétrica, esta técnica ha sido aplicada en muchos problemas geométricos. Otra de las técnicas es la de matrices ordenadas, esta técnica mejora en un factor logarítmico las soluciones obtenidas usando búsqueda paramétrica. Algunas de las ventajas de las matrices ordenadas, sobre

la búsqueda paramétrica, es que no requiere paralelizar el algoritmo de decisión y obtiene algoritmos, usualmente más eficientes que los obtenidos con búsqueda paramétrica.

Por otra parte el problema principal podría tener un enfoque diferente, por ejemplo, ¿qué pasaría si se considera que la colección a intersectar tiene diferentes tipos de objetos geométricos? Es decir, incluye rectas, segmentos de rectas, rayos, en el plano y se desea encontrar el círculo o los círculos de radio mínimo que intersecten a todos los objetos de la colección.

Otro enfoque es extender este trabajo de tesis generalizando las soluciones aquí dadas para encontrar ahora los 3 círculos de radio mínimo, tal que su unión intersecte a todos los miembros de una colección de objetos geométricos, ¿en qué tiempo de ejecución pueden encontrarse tales círculos?

Así como se podría aplicar la búsqueda paramétrica, en el trabajo de Chazelle *et al*, 1992, [9], hay métodos alternativos a esta técnica, la idea de Chazelle se basa en *epsilon nets*.

Finalmente se podrían cambiar los objetos que intersectan a la colección por cualquier figura geométrica, por ejemplo, ¿cómo sería la solución algorítmica si dada una colección de objetos geométricos en el plano, se desea encontrar el triángulo de menor área o perímetro que intersecte a todos los objetos de la colección? ¿Qué tan rápido

se encuentra tal triángulo? ¿Será posible encontrarlo en un tiempo subcúbico? Es posible que la búsqueda paramétrica pueda ser utilizada, habría que analizar el algoritmo para resolver el problema de decisión considerando estas variantes.

Bibliografía

[1] Pankaj K. Agarwal and Micha Sharir. Planar geometric location problems. *Algorithmica*, 1994.

[2] M. Ajtai and N. Meggido. A deterministic poly(log log n)-time n-processor algorithm for linear programming in fixed dimension. *SIAM J. Computating*, 25:1171–1195, 1996.

[3] N. Alon and J. Spencer. *The probabilistic method*. Wiley-Interscience, 1992.

[4] David L. Applegate, Robert E. Bixbyand Vasek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

[5] P. K. Awargal, M. Sharir, and S. Toledo. An efficient multi-dimensional searching technique and its applications. *Tech. Rep. CS-1993-20, Department of Computer Science, Duke University*, 1993.

[6] Binay K. Bhattacharya, Sreesh Jadhav, Asish Mukhopadhayay, and Jean-Marc Robert. Optimal algorithms for some smallest intersection radius problems. *ACM*, 1991.

[7] H. Bronnimann and B. Chazelle. Optimal slope selection via cuttings. *In Proceedings of the Sixth Canadian Conference on Computational Geometry*, pages 99–103, 1994.

[8] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 18(9):145–158, 1993.

[9] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Diameter, width, closest line pair, and paramerching. *Proc. 8th AMC Symp. on Computational Geometry*, pages 120–129, 1992.

[10] Mark de Berg, Otfried Cheong, Marc Kreveld, and Mark Overmars. *Computational Geometry Algorithms and Applications*. Springer Verlag, 2008.

[11] Z. Drezner. The planar two-center and two-median problems. *Transportacion Science*, 18:351–361, 1984.

[12] M. E. Dyer. Linear-time algorithm for two- and three- variable linear programs. *SIAM J. Computing*, 1984.

[13] M. E. Dyer. Linear programming in $o(n3^{d^2})$ time. *Inf. Process. Lett.*, 22:21–24, 1986.

[14] M. E. Dyer. On a multidimensional search technique and its application to the euclidean 1-center problem. *SIAM J. Computing*, 1986.

[15] M. Eisner and D. Severance. Mathematical techniques for efficient record segmentation in large shared databases. *ACM*, 23:619–635, 1976.

[16] G. N. Frederickson. Optimal algorithms for free partitioning. *In Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 168–177, 1991.

[17] G. N. Frederickson and D. B. Johnson. The complexity of selection and ranking in x + y and matrices with sorted rows and columns. *J. Comput. Syst. Sci.*, 24:197–208, 1982.

[18] G. N. Frederickson and D. B. Johnson. Finding kth paths and p-centers by generating and searching good data structures. *J. Alg.*, 4:61–80, 1983.

[19] G. N. Frederickson and D. B. Johnson. Generalized selection and ranking: Sorted matrices. *SIAM J. Computing*, 13:14–30, 1984.

[20] Alex Glozman, Klara Kedem, and Gregory Shpitalnik. On some geometric selection and optimization problems via sorted matrices. *Lecture notes in Computer Science*, (955):26–35, 1998.

[21] M. E. Houle, H. Imai, K. Imai, and J. M. Robert. Weighted orthogonal linear loo -approximation and applications. *Lecture notes in Computer Science Procs. of the Workshop on Algorithms and Data Structures*, 382:183–191, 1989.

[22] Miroslaw Kowaluk Jerzy W. Jaromczyk. An efficient algorithm for the euclidean two-center problem. *10th Computational Geometry ACM*, 1994.

[23] M. J. Katz. Improved algorithms in geometric optimization via expanders. *In Proceedings of the Third Israel Symposium on Theory of Computing and Systems*, pages 78–87, 1995.

[24] M. J. Katz and M. Sharir. Optimal slope selection via expanders. *Inf. Process. Lett.*, 47:115–122, 1993.

[25] J. Matousek. Range searching with efficient hierarchical cuttings. *Proc. 8th ACM Symp. on Computational Geometry*, pages 276–285, 1992.

[26] N. Meggido. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30:852–865, 1983.

[27] N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics Operation Res.*, 4:414–424, 1979.

[28] N. Megiddo. Linear-time algorithms for linear programming in r^3 and related problems. *SIAM J. Comput.*, 12:759–776, 1983.

[29] N. Megiddo. Linear programming in linear time when the dimension is fixed. *JACM*, 1984.

[30] N. Megiddo and K. Supowit. On the complexity of some common geometric location problems. *SIAM J. Computing*, 13:182–196, 1984.

[31] S. G. Mentzer. Lower bounds on metric k-center problems, manuscript. *Manuscript*, 1988.

[32] J. Nesetril, E. Milkova, and H. Nesetrilova. Otakar boruvka on minimum spanning tree problem : Translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233:3–36, 2001.

[33] M. Overmars and J. van Leeumeen. Maintenance of configurations in the plane. *J. Comp. System Sciences*, (23):166–204, 1981.

[34] Micha Sharir. A near-linear algorithm for the planar 2-center problem. 1996.

[35] L. Valiant. Parallelism in comparison problems. *SIAM Journal on Computing*, (4):348–355, 1975.