Universidad Nacional Autónoma de México

Posgrado en Ciencia e Ingeniería de la Computación

Caminos Heterocromáticos y Algoritmos de Localización de Servicios

T E S I S

QUE PARA OBTENER EL GRADO DE:
Maestra en Ciencias (Computación)

P R E S E N T A :

Ramírez Vigueras Adriana

Director de tesis:
Dr. Jorge Urrutia Galicia

2008

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Agradecimientos

A mis padres Marcos y Esperanza por ser siempre los ejemplos a seguir.

A mis hermanos Vale, Memo, Lulú, May, Checo y Juanito por apoyarme en todo, en especial a “Mi Vale” por estar siempre ahí cuando te necesite.

A mis cuñadas Gaby, Mary Carmen e Isabel y mi cuñado José por mostrar siempre interés en las cosas que hago.

Al mejor tutor de prosgrado que alguien pudiera tener: Jorge Urrutia, gracias por tus sabios consejos tanto en el ámbito académico como en el personal, gracias a tí he aprendido a salir de la bola de cristal en la que vivía.

A mis sinodales Sergio, Francisco, José y especialmente a Hernán por sus valiosos comentarios y sugerencias para que este trabajo saliera lo mejor posible.

A todos los asistentes del Primer Taller Iberoamericano de Geometría Combinatoria y Computacional, por sus comentarios que ayudaron mucho a la solución de los problemas presentados en este trabajo. En especial a José Miguel Díaz Báñez, Inmaculada Ventura, Gregorio Hernández Peñalver y Antoni Sellarès.

Al hombre más consentidor de este planeta, Gil, gracias por estar siempre, siempre en todo momento junto a mi.

A Mónica Leñero, por ser la mejor jefa y amiga del mundo.

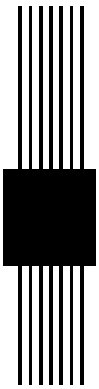
A Crevel Bautista por apoyarme siempre y por mostrarme que la vida no siempre es color de rosa, gracias Crevel.

A todos mis amigos por los momentos tan divertidos que he pasado con ustedes, en especial a Rene Villeda. La chispa de alegría y cooperación que siempre muestras hasta hicieron divertida la elaboración de un inútil microporcesador.

Agradezco a la UNAM y al CONACYT por el apoyo que me han brindado estos años. Gracias a estas dos instituciones he conocido a casi toda la gente mencionada en estos agradecimientos y además he creecido profesional y personalmente.

Finalmente agradezco a toda la gente que quiero y que olvide, pero la urgencia de tener el grado ya no puede esperar :-).

Índice general



Lista de figuras

Lista de tablas

Introducción 1

1.1 Origen del problema y utilidades

Durante los últimos años, uno de los temas más estudiados en el área de la Geometría Computacional ha sido la búsqueda de subestructuras heterocromáticas minimales [3, 1, 16], lo cual involucra directamente la búsqueda de caminos más cortos, es decir, que la distancia recorrida para ir de un sitio a otro, visitando un conjunto de lugares en particular, sea la menor posible. Por ejemplo, supongamos que una persona realizará una diligencia para cubrir una serie de trámites para obtener su pasaporte, lo cual obliga a dicha persona a recaudar algunos documentos necesarios para cubrir todos los requisitos establecidos, tales como, el pago requerido, un acta de nacimiento y algunas fotografías, esto se traduce en una visita al banco, al registro civil y al estudio fotográfico, respectivamente. Por tanto, es deseable conocer o hallar la ruta más corta, entre su casa (como punto de partida) y la delegación de la secretaría de relaciones exteriores (como destino), pasando por los lugares ya mencionados. En general a este tipo de problemas se les conoce como problemas de *localización de rutas óptimas*.

Un problema conocido dentro de la literatura es el **TPP** (*Trip Planning Problem*) [12] cuyo enunciado es: Dado un conjunto P_n de n puntos en el plano, pintados de k colores distintos[1], un punto de partida a y otro de destino b, deseamos encontrar el camino más corto que conecta a y b, que pasa por al menos un punto de cada color. Tal es el caso del problema descrito anteriormente para ir de la casa de una persona a una oficina consular y la manera de distinguir a cada punto, por ejemplo los bancos, pueden ser puntos azules, el registro civil un punto verde, etcétera. Es fácil ver que este problema es NP-duro, ya que en el caso de que todos los elementos de P_n tengan colores distintos, nuestro problema

[1] Comúnmente llamados conjuntos k coloreados

se transforma en el problema del agente viajero (*Traveling Salesman Problem*, **TSP**) que se sabe es un problema NP-duro [8, 13].

Debido a la importancia y a la imposibilidad de tener una solución polinomial del problema TPP, en este trabajo se impondrán algunas restricciones sobre sus variantes que nos permitirán obtener soluciones parciales a un subconjunto de instancias del problema original.

Otro problema que motivó el planteamiento de este trabajo es el siguiente: supongamos que nuestro agente viajero tiene un mapa de la ciudad que está visitando y desea encontrar el vecindario al que pertenece un atractivo turístico que quiere conocer, una manera de ayudar a nuestro agente viajero a encontrar dicho vecindario es usando una búsqueda de localización de puntos (*point location query*), cuyo enunciado, desde el punto de vista geométrico, es: dada una subdivisión del plano en regiones y un punto de búsqueda q, encontrar la región del plano que contiene dicho punto q. Estos conceptos y el tema en general serán tratados a detalle en los siguientes capítulos.

Por lo anterior, se definieron problemas con las siguientes restricciones:

- La dirección de la ruta. Esta restricción involucra el concepto de monotonía, la cual ha sido ampliamente estudiada en los problemas de transportes y modelación porque simplifica tanto la búsqueda de la estructura como la búsqueda misma de piezas monótonas que nos dan información sobre la monotonía implícita de un conjunto de datos espaciales [2, 6, 16].
- El orden de visita. Esta restricción la aplicamos para establecer un orden al realizar un recorrido. Por ejemplo, retomando el problema de tramitar el pasaporte, hay que considerar que es necesario hacer la primer parada en el banco para poder solventar los gastos de las paradas restantes.

Por lo tanto, el uso de este par de restricciones nos lleva a tener cuatro variantes posibles del problema original.

1.2 Problemas planteados

Las cuatro variantes mencionadas en la sección anterior se resumen fácilmente en la tabla 1.1, en la cual se observan todas las posibles combinaciones entre las restricciones de dirección de ruta (monotonía) y orden de visita.

Caso	Restricciones	
	Monotonía	Orden de visita
1	$\surd$	$\surd$
2	$\times$	$\surd$
3	$\surd$	$\times$
4	$\times$	$\times$

Tabla 1.1: Para los casos uno y tres la dirección de la ruta (monotonía) puede ser fija o no.

Las definiciones de los seis problemas que resume la tabla 1.1 tienen como hipótesis lo siguiente:

*D*ados n puntos en el plano, en posición general y $n = n_1 + n_2 + \cdots + n_k$ donde n_i es el número de puntos de color c_i:

Para el caso uno:

Problema 1 (Poligonal Monótona Orientada y Ordenada de Longitud Mínima (PMOO)). Encontrar la poligonal x-monótona de longitud mínima que visite exactamente un punto de cada color en un orden de colores de visita prefijado.

Problema 2 (Poligonal Monótona Ordenada de Longitud Mínima (PMO)). Encontrar una dirección $\theta \in [0, \pi)$ de forma que la poligonal óptima del problema PMOO con dirección θ es la menor de entre todas las poligonales θ-monótonas posibles con $\theta \in [0, \pi)$.

Para el caso dos:

Problema 3 (Poligonal Ordenada de Longitud Mínima (PO)). Encontrar la poligonal de longitud mínima que visite exactamente un punto de cada color en un orden de colores de visita prefijado.

Para el caso tres:

Problema 4 (Poligonal Monótona Orientada de Longitud Mínima (PMOrientada)). Encontrar la poligonal de longitud mínima y monótona en una dirección dada por $\theta \in [0, \pi)$ que contiene exactamente un punto de cada color.

Problema 5 (Poligonal Monótona de Longitud Mínima (PM)). Encontrar una dirección $\theta \in [0, \pi)$ de forma que la poligonal óptima del problema PMOrientada con dirección θ es la menor de entre todas las posibles poligonales θ-monótonas posibles.

El caso cuatro se trata del TPP.

Para una mejor comprensión y mayor flexibillidad en la notación, a lo largo del trabajo denotaremos a los colores como $c_1, c_2, c_3, \ldots, c_k$.

1.3 Soluciones obtenidas

En este trabajo se resuelven los problemas planteados en la sección anterior de la siguiente manera:

- En el caso de los problemas PMOO y PMO nuestras soluciones están basadas en el uso de algoritmos de localización eficiente de puntos (*point location*) en una subdivisión del plano dada por el cálculo de diagramas de Voronoi dinámicos, logrando complejidades de tiempo de $O(n \log^2 n)$ y $O(n^3 \log^2 n)$ para cada problema, respectivamente.
- La solución para el problema PO está basada fundamentalmente en el cálculo del diagrama de Voronoi para cada clase cromática, por lo cual obtenemos un algoritmo de complejidad de tiempo de $O(k(n \log n))$.
- La solución que tenemos por el momento para resolver los problemas PMOrientada y PM es sólo considerar todas las posibles permutaciones de los k colores y aplicar los algoritmos de PMOO y PMO obteniendo algoritmos de complejidades de tiempo $O(k!(n \log^2 n))$ para resolver el PMOrientada y $O(k!(n^3 \log^2 n))$ para resolver PM.

1.4 Estructura de este trabajo

Por último mostramos la estructura de los siguientes capítulos de este trabajo:

En el capítulo 2 se definen todos los conceptos necesarios para hacer un mejor estudio de los algoritmos mostrados en este trabajo.

En el capítulo 3 se explican a detalle los algoritmos obtenidos y se hace el análisis de la complejidad de tiempo y espacio que estos requieren para obtener las soluciones a los problemas planteados en la sección 1.2.

Finalmente en el capítulo 4 se plantean problemas para trabajar a futuro y las conclusiones de este trabajo de tesis.

Conceptos Fundamentales

En este capítulo se introducirán algunos conceptos y técnicas de la geometría computacional que son fundamentales para una mejor comprensión de los algoritmos desarrollados en este trabajo.

2.1 Definiciones

Dado que todos los problemas de este trabajo de tesis se basan principalmente en encontrar una poligonal de longitud mínima, procederemos a definir que es una poligonal.

Definición 1. Una poligonal P es una curva definida por una secuencia de puntos $p_1, p_2, \ldots, p_n$, llamados vértices, los cuales están conectados consecutivamente por segmentos de línea. Figura 2.1.

Si los segmentos de línea que representan a P no se intersectan entre si, diremos que P es una poligonal simple.

Algunos de los problemas planteados en el capítulo anterior tienen la restricción de que la poligonal debe ser monótona, dicha restricción se define formalmente como:

Definición 2. Sea P_M una poligonal y θ un ángulo tal que $\theta \in [0, \pi)$. Decimos que P_M es monótona con respecto a θ si toda recta con dirección $\theta + \frac{\pi}{2}$ intersecta a P_M en un conjunto conexo, donde un conjunto conexo es aquel que consiste de un punto o para cualesquiera dos puntos del conjunto existe una poligonal que los une. Figura 2.2.

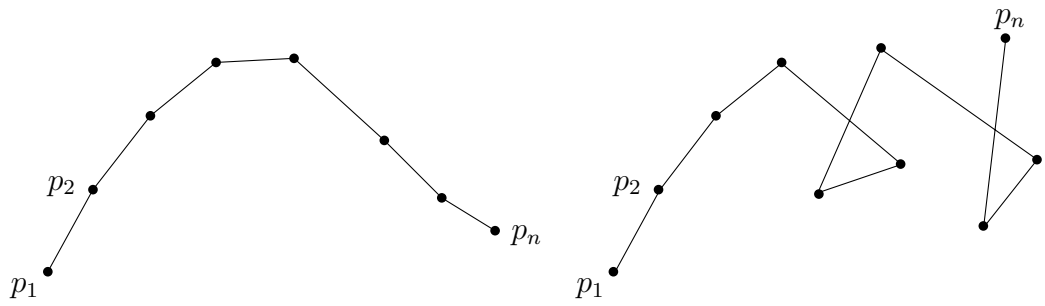


Figura 2.1: Poligonal simple y con cruces.

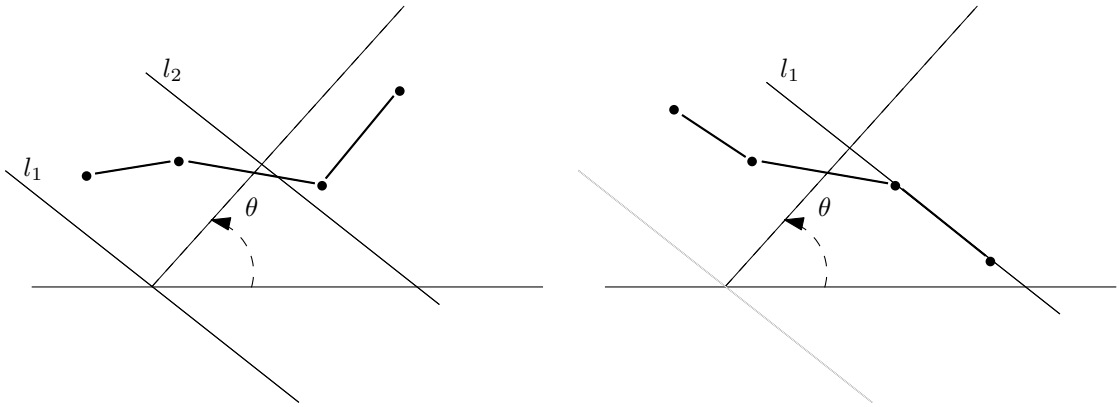


Figura 2.2: Monotonía.

Finalmente, otras características que debe cumplir la poligonal es que sea heterocromática y ordenada, por tanto definiremos estos conceptos de la siguiente manera:

Definición 3. Sea $S = \{p_1, p_2, p_3, \ldots, p_n\}$ un conjunto de n puntos en el plano coloreados con k colores distintos y $k \leq n$, si $P_H = (p_{i_1}, p_{i_2}, \ldots, p_{i_k})$ es una poligonal heterocromática, entonces sus vértices son al menos de dos colores distintos.[1] Figura 2.3.

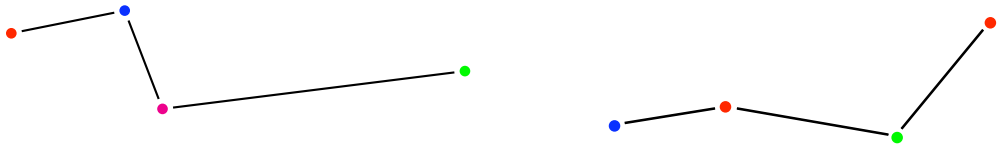


Figura 2.3: Poligonal heterocromática

[1]Haciendo un abuso de lenguaje, a lo largo del trabajo llamaremos a la poligonal formada por un vértice de cada color, poligonal heterocromática.

Definición 4. Sea $S = \{p_1, p_2, \ldots, p_n\}$ un conjunto de n puntos en el plano coloreados con k colores distintos y $k \leq n$, si $P_O = (p_{i_1}, p_{i_2}, \ldots, p_{i_k})$ es una poligonal con vértices $p_{i_1}, p_{i_2}, \ldots, p_{i_k}$, se dice que es ordenada, si sus vértices son recorridos de acuerdo a un orden de colores de visita prefijado $(i_1, i_2, \ldots, i_k)$.

Por ejemplo, la figura 2.4 muestra una poligonal ordenada con orden (● ● ● ●).

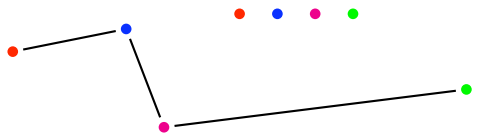


Figura 2.4: Poligonal ordenada.

Definición 5. La longitud de la poligonal $P = (p_1, p_2, \ldots, p_n)$ es definida como $l(P) = \sum_{i=1}^{n} d(p_i, p_{i+1})$ donde $d(p_i, p_{i+1})$ es la distancia euclidiana entre los puntos p_i y p_{i+1}. Figura 2.5.

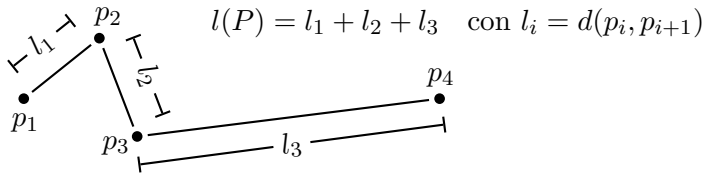


Figura 2.5: Definición de la longitud de la poligonal P

2.2 Diagrama de Voronoi

Un problema clásico en varias áreas de investigación es el problema de localización de servicios (*facility location*). Por ejemplo, en el área de bases de datos al tener grandes cantidades de información nos gustaría preparar, sondear y explorar la información oculta de los datos (*data mining*), en la medicina nos interesa buscar patrones a partir de ciertos bancos de información que nos ayuden a detectar alguna enfermedad, por último, otra área en la que se aplica la localización de servicios es en la de "geografía social", dado que un problema es suponer que somos dueños de una cadena de supermercados y deseamos abrir una nueva sucursal (en el contexto de geometría computacional es llamado un punto de servicio) por lo tanto, hay que pensar en respondernos dónde ubicarla de manera que dicha tienda nos lleve a tener el mayor beneficio. Además en dicho problema se consideran algunas restricciones como:

- Por parte del servicio, mantener los mismos precios.

- El precio de obtener un servicio es igual al precio del servicio más el precio de transladarse a la tienda.
- El precio de transladarse a la tienda, depende de la distancia euclidiana, fijando un precio por cada unidad que contenga dicha distancia.
- Por parte del cliente, minimizar los gastos que conllevan los servicios que este requiere.

Usualmente las restricciones anteriores no siempre son completamente satisfacibles, dado que algunos servicios son más baratos que otros o que el traslado de nuestra casa al punto de servicio problablemente no es lineal (en línea recta). Por lo tanto, las posibles soluciones a este problema se reducen a aproximar, en una región, el punto estratégico de colocación de nuestra nueva tienda, donde dicha región puede ser sumamente restringida o bien, puede abarcar el área completa de nuestro vecindario.

Nuestra principal motivación en el estudio de este tipo de problemas es la interpretación geométrica que podemos darle, es decir, que las restricciones citadas anteriormente inducen un modelo de subdivisión donde cada cliente (punto) tiene asignado el servicio (punto de servicio) más cercano, coincidiendo este modelo de subdivisión con la subdivisión inducida por el diagrama de Voronoi del conjunto de servicios.

2.2.1 Definición y algunas propiedades

El diagrama de Voronoi es una estructura geométrica versátil, ya que no sólo puede ser aplicado en "geografía social", como mencionamos en la sección 2.2 con el problema de ubicación de tiendas, sino también en otras áreas, por ejemplo, en la astronomía.

Definición 6. Sea $P = \{p_1, p_2, \ldots, p_n\}$ un conjunto de puntos en el plano en posición general. El diagrama de Voronoi de P es la subdivisión del plano en celdas, una por cada punto de P, con la propiedad de que un punto q pertenece a la celda del punto p_i si, y sólo si, $d(q, p_i) < d(q, p_j)$ para toda $j \neq i$. Hay que hacer notar que las aristas de la subdivisión están definidas por todos los puntos q tal que $d(q, p_i) = d(q, p_j)$ con $j \neq i$. Figuras 2.6 y 2.7.

Denotaremos al diagrama de Voronoi del conjunto de puntos P como $Vor(P)$ y a cada celda de $Vor(P)$ que corresponde a un punto p_i la denotaremos como $V(p_i)$. Figura 2.8.

Definición 7. Sean p y q dos puntos en el plano. Definimos al bisector de p y q como la recta perpendicular al segmento $\overline{pq}$ que pasa por su punto medio. Figura 2.9.

El bisector de los puntos p y q divide al plano, en dos mitades, uno que contiene al punto p denotado por $h(p, q)$ y otro que contiene al punto q denotado por $h(q, p)$. Hay que hacer notar que $r \in h(p, q)$ si, y sólo si, $d(r, p) \leq d(r, q)$.

A partir de esto obtenemos la siguiente observación:

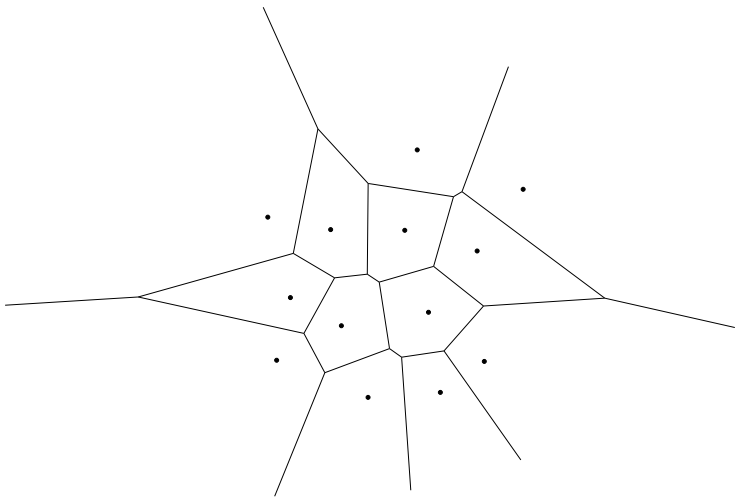


Figura 2.6: Diagrama de Voronoi.

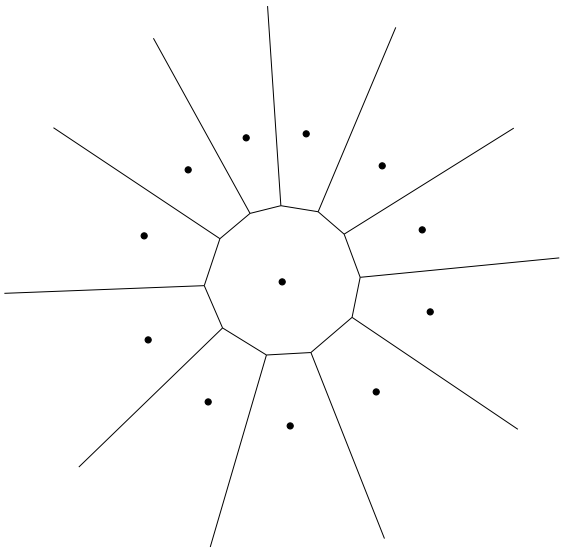


Figura 2.7: Diagrama de Voronoi.

Observación 1. $V(p_i) = \bigcap_{1 \leq j \leq n, j \neq i} h(p_i, p_j)$.

Por la la observación 1, cada región de Voronoi es la intersección de $n-1$ semi-planos que contienen al punto p_i. Por lo tanto tenemos la siguiente observación:

Observación 2. Toda celda de Voronoi $V(p_i)$ es una región abierta y convexa.

Una manera intuitiva de ver el diagrama de Voronoi de un conjunto finito de puntos

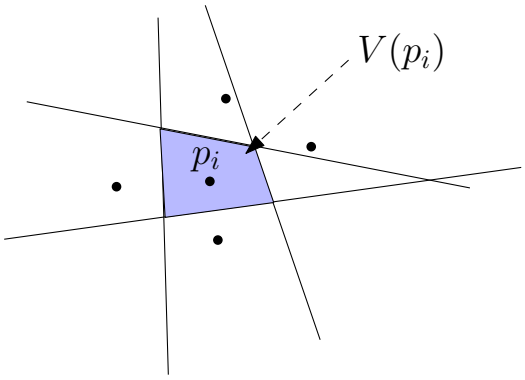


Figura 2.8: Celda de Voronoi del punto p_i, $(V(p_i))$.

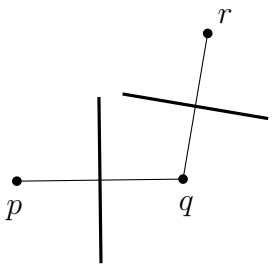


Figura 2.9: Bisectores de $\overline{pq}$ y de $\overline{qr}$

P y obtener más de sus propiedades es la siguiente: sea x un punto arbitrario en el plano. Fijamos el centro de un círculo C de diámetro cero en x, incrementamos el diámetro de C hasta tocar por primera vez uno o más puntos de P, este proceso de expansión nos lleva a 3 casos distintos que se presentan en el siguiente lema:

Lema 1. Si el círculo C:

- toca exactamente un punto $p_i \in P$, entonces $x \in V(p_i)$.
- toca exactamente dos puntos $p_i,\ p_j \in P$ al mismo tiempo, entonces x pertenece a la arista de Voronoi que separa las regiones $V(p_i)$ y $V(p_j)$.
- toca tres o más puntos de P al mismo tiempo, entonces x es el vértice de Voronoi que tienen en común todas las regiones definidas por los puntos que tocó.

Demostración. Si C sólo tocó al punto p_i, entonces p_i es el punto, de P, más cercano a x. Consecuentemente $x \in h(p_i, p_j)$ para todo $p_j \in P$ con $i \neq j$, por lo tanto $x \in V(p_i)$ y con esto queda mostrado el primer caso del lema. Si C tocó exactamente dos puntos p_i y p_j al mismo tiempo, entonces $x \in h(p_i, p_k)$ y $x \in h(p_j, p_k)$, para toda $k \neq i, j$ y el bisector de p_i y p_j delimita la frontera de $h(p_i, p_j)$ y $h(p_j, p_i)$ entonces, por la observación 1, $x \in V(p_i)$ y

$x \in V(p_j)$, con esto queda mostrado el segundo enumciado del lema. Finalmente el tercer caso se prueba de manera similar al caso dos.

□

Algunas otras propiedades observadas a partir de las características del diagrama de Voronoi son:

- Para cada vértice de Voronoi v, el círculo que tiene como diamétro el segmento $\overline{vp_i}$ no contiene otro punto $p_j \in P$ con $i \neq j$.
- Se puede verificar que para n puntos el diagrama de Voronoi tiene n celdas, a lo más $3n - 6$ aristas y $2n - 5$ vértices.

2.2.2 Calculando el diagrama de Voronoi

La observación 1 sugiere un algoritmo simple para calcular el diagrama de Voronoi: Sea $P = \{p_1, p_2, \ldots, p_n\}$ un conjunto de puntos en el plano. Por cada punto p_i calculamos la intersección de todos los semi-planos $h(p_i, p_j)$ con $j \neq i$. Esto nos lleva a tener una complejidad de tiempo $O(n \log n)$ por cada $V(p_i)$, así que calcular las celdas de Voronoi de todos los puntos p_i nos lleva a tener una complejidad total de $O(n^2 \log n)$.

2.3 Diagrama de Voronoi con barrido

En la sección 2.2.2, mostramos un algoritmo para calcular el diagrama de Voronoi, pero al tener como objetivo encontrar la mejor complejidad de tiempo y dado que la complejidad total del algoritmo mostrado en 2.2.2, como veremos más adelante no es la mejor, nos preguntamos ¿puede calcularse en mejor tiempo? La respuesta es sí y el algoritmo que lo hace ya es conocido como el algoritmo de *Fortune*. Dicho algoritmo calcula el diagrama de Voronoi en tiempo $O(n \log n)$. Después de esto podríamos hacernos la pregunta otra vez, que si podemos mejorar dicha complejidad, pero la respuesta es no, dado que el problema de calcular el diagrama de Voronoi de n puntos es reducible al problema de ordenar n números reales, por lo tanto, el algoritmo óptimo para calcular el diagrama de Voronoi debe tener complejidad de tiempo $O(n \log n)$ ya que se sabe que la mejor cota para ordenar n números reales es esa. La demostración de dicha reducción está dada por el siguiente lema:

Lema 2. Calcular el diagrama de Voronoi de n puntos se reduce al problema de ordenar n números reales.

Demostración. Sean $x_1, x_2, \ldots, x_n$, n números reales y T una transformación tal que $T(x_i) = x_i^2$ para $1 \leq i \leq n$. Denotemos por p_i al punto $(x_i, T(x_i))$ y consideremos al conjunto $P = \{p_1, p_2, \ldots, p_n\}$. Figura 2.10. Después, calculamos el diagrama de Voronoi de P. Figura 2.11

Ahora si tomamos el punto p_i con valor de abscisa más pequeño (p_2) y recorremos cada región de Voronoi en sentido contrario de las manecillas del reloj (Figura 2.11), obtenemos una secuencia de puntos ordenados por abscisa creciente, esto es, obtenemos los puntos

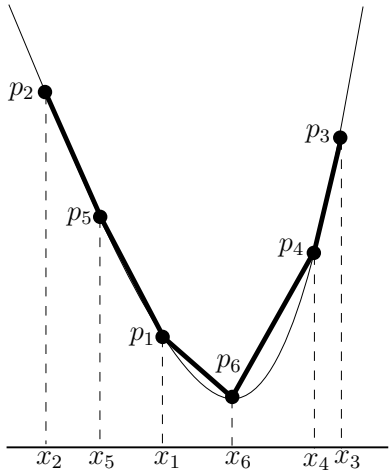


Figura 2.10: Conjunto de puntos construidos

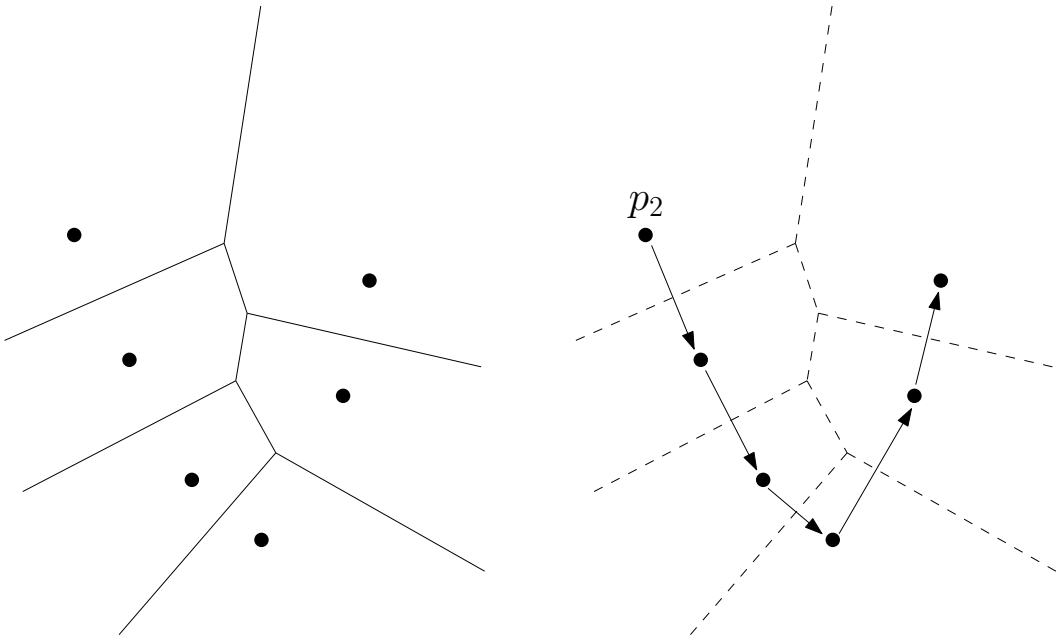


Figura 2.11: Diagrama de Voronoi y recorrido de regiones

$x_1, x_2, \ldots, x_n$ de forma ordenada. Por lo tanto, si el diagrama de Voronoi se hiciera en mejor tiempo de $\Omega(n\ log\ n)$, obtendríamos un algoritmo para ordenar una secuencia de números en un mejor tiempo que los algoritmos de ordenamiento conocidos. □

Debido al lema 2 podemos concluir que el algoritmo de *Fortune* es óptimo. Por lo tanto, calcular el diagrama de Voronoi tiene complejidad de tiempo $O(n \log n)$.

2.3.1 Algoritmo de Fortune para calcular el diagrama de Voronoi

El algoritmo de *Fortune* es un algoritmo basado principalmente en la técnica de barrido (*sweepline*), donde dicha técnica conceptualmente consiste en trasladar o mover una línea recta de arriba hacia abajo sobre el plano. Figura 2.12.

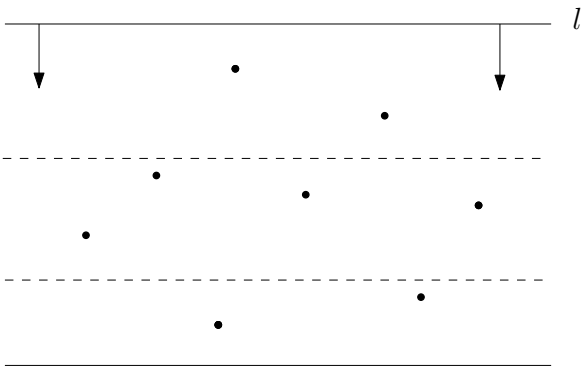


Figura 2.12: Estrategia de barrido en el plano

Una consideración mientras se aplica esta técnica es, que mientras la línea se mueve, el conjunto de puntos P definido en el plano se mantiene sin cambiar la estructura que se desea calcular (en este caso el diagrama de Voronoi), excepto en ciertos puntos especiales, llamados eventos puntuales, que generalmente ocurren cuando la línea de barrido se intersecta con algún punto de P. Figura 2.13.

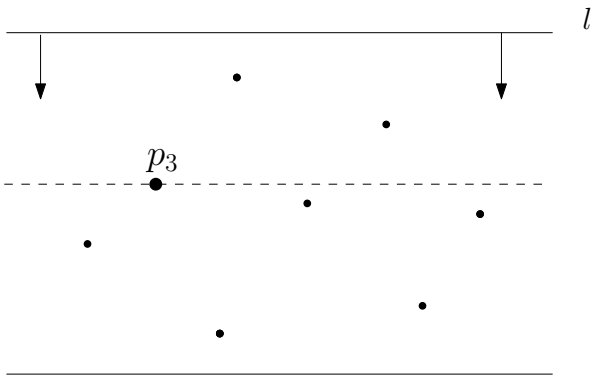


Figura 2.13: Evento puntual en p_3

A continuación explicaremos como hacer uso de esta técnica para calcular el diagrama de Voronoi de un conjunto de puntos $P = \{p_1, p_2, \ldots, p_n\}$ en el plano. Sin perdida de generalidad denotaremos a la línea de barrido como l. El barrido antes mencionado nos induce mantener la intersección del diagrama de Voronoi con l. Desafortunadamente esto no es fácil, porque el diagrama de Voronoi podría no sólo depender de los puntos que están arriba de l, sino también de los puntos que se encuentran debajo de l. Por ejemplo, se puede dar el caso de que el vértice más alto de la celda de Voronoi $V(p_i)$ del punto p_i, es intersectado por l antes que intersecte a p_i, lo cual implica que el diagrama de Voronoi construido hasta ahora con los puntos que están arriba de l va a ser modificado. Figura 2.14. Debido a esto no podemos mantener la intersección del diagrama de Voronoi de lo puntos hasta ahora barridos con l, porque no tenemos la información suficiente para calcular la celda de Voronoi de dicho vértice. Por esta razón la estrategia

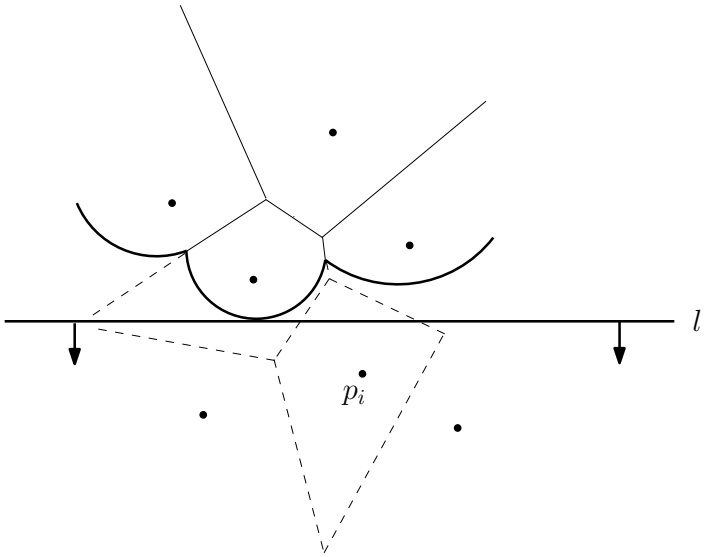


Figura 2.14: Diagrama de Voronoi y l

de barrido del plano será aplicada de manera ligeramente distinta, es decir, en vez de mantener la intersección de $Vor(P)$ con l, mantendremos sólo el diagrama de Voronoi de los puntos arriba de l que no va a ser cambiada por los puntos que están debajo de l. Figura 2.14 líneas sólidas. Para un uso más comodo del lenguaje denotaremos al conjunto de todos los puntos de P que están arriba y abajo de l como l^+ y l^-, respectivamente.

Una vez mencionado el cambio que se hará a la técnica de barrido, debemos aclarar cómo saber que parte del diagrama de Voronoi no va a cambiar. Dicho de otra manera, determinar para cuales puntos $q \in l^+$ garantizamos que su región de voronoi ya fué barrida. Vamos a mantener el diagrama de Voronoi si $d(q, l) > d(r, l)$ con $q \in l^+$ y $r \in l^-$. Garantizando así que el punto más cercano a q no pertenece al conjunto de puntos de l^- y si q está al menos tan cerca a cualquier punto $p_i \in l^+$ como q esta de l. Además

el lugar geométrico de estos puntos que son delimitados por algún punto $p_i \in l^+$ son acotados por una parábola y los que no son delimitados, entonces serán acotados por arcos parabólicos; llamando a esta secuencia de arcos parabólicos curva de contorno (*beach line*).

La manera de generar la curva de contorno es la siguiente: por cada valor x, se crea una línea vértical l_v que pase por el punto $(x, 0)$, cosideremos el valor de coordenada y más pequeño, entre los puntos de intersección de l_v con las parábolas, los puntos (x, y) así determinados generan la curva de contorno. Figura 2.15.

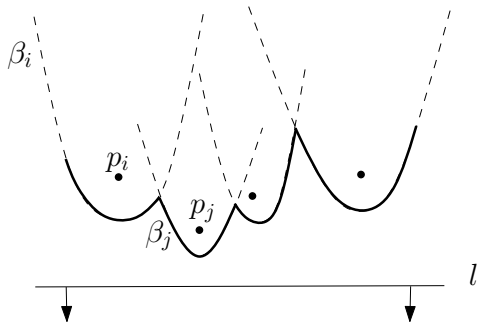


Figura 2.15: Curva de contorno

Es fácil de ver que la construcción de la curva de contorno que es x-monótona porque a cada coordenada x le asigna un sólo valor, por lo tanto, tenemos la siguiente observación:

Observación 3. La curva de contorno (l_c) es x-monótona, es decir, para toda línea vertical l_v intersecta a l_c exactamente en un punto.

Más adelante veremos que una parábola β_i puede modificar sólo una vez en l_c, también que los cruces (*breakpoints*) de los arcos parabólicos que forman l_c están sobre las aristas del diagrama de Voronoi. Esto no es una coincidencia, ya que mientras l hace el barrido de arriba hacia abajo, los cruces de los arcos parabólicos determinan el diagrama de Voronoi. Debido a esto, en vez de mantener la intersección de $Vor(P)$ con l, mantendremos la intersección con l_c conforme vamos moviendo l. Para realizar esta tarea, primeramente estudiaremos los siguientes casos: ¿cuándo aparece un nuevo arco parabólico?, esto es, ¿cuándo se genera un evento puntual? Y la respuesta es sencilla, esto ocurre cuando al mover l alcanza un nuevo punto de P. La parábola definida por este punto es, en principio, un segmento de línea vertical que conecta el nuevo punto con l_c. Conforme l continua moviéndose, la parábola se va haciendo más "ancha" hasta intersectar a l_c. Por lo tanto, la nueva curva de contorno está definida por la intersección de la nueva parábola y la curva de contorno que se tenía hasta antes de tocar el punto que genera la nueva parábola. Figura 2.16.

Cuando ocurre un evento puntual nos vemos en la necesidad de observar que cambio se da en l_c y más específicamente, que cambios surgieron en el diagrama de Voronoi, ya que, como se mencionó anteriormente, los cruces de la curva de contorno l_c determinan las aristas de Voronoi, porque cuando un nuevo punto es intersectado por l, dos nuevos

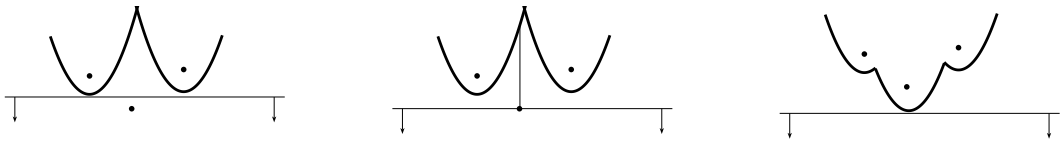


Figura 2.16: Proceso de obtener la curva de contorno

cruces aparecen, empezando a determinar las aristas de Voronoi. De hecho, los dos nuevos cruces son el mismo al inicio, cuando la parábola que va a ser generada es un segmento y al continuar aplicando el desplazamiento de l se determina una arista e de Voronoi. Figura 2.17. Inicialmente esta arista no está conectada con el diagrama de Voronoi arriba de l. Más adelante explicaremos brevemente cuando se conecta e con el resto del diagrama de Voronoi.

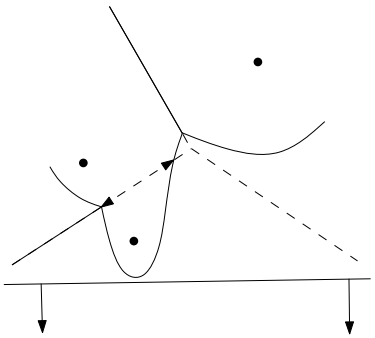


Figura 2.17: Determinando una arista de Voronoi

Resumiendo sabemos que cuando un nuevo punto es intersectado por l (evento puntual) un nuevo arco parabólico aparece en l_c y en consecuencia una nueva arista de Voronoi empieza a ser determinada, pero si esto no sucede, es decir, si no ocurre un evento puntual entonces no podemos agregar un nuevo arco parabólico en l_c. Esta afirmación es mostrada en el siguiente lema:

Lema 3. Sólo cuando ocurre un evento puntual aparece un nuevo arco en l_c.

Demostración. La prueba la haremos por contradicción, es decir, supongamos que existen dos formas de dividir a l_c con alguna parábola β_j definida por el punto p_j.

La primer forma es que β_j rompa en dos a un arco definido por la parábola β_i (Figura 2.18), esto nos lleva a tener que existe un momento en que β_j y β_i son tangentes, esto es, que en un momento sólo tienen un punto de intersección. Denotaremos a la coordena y de l como l_y justo cuando se da la tangencia. Si $p_j = (p_{jx}, p_{jy})$ entonces la parábola β_j con

foco en p_j y vértice en el punto medio entre el punto p_j y la línea l_y esta definida como:

$$\beta_j = y = \frac{1}{2(p_{jy} - l_y)}(x^2 - 2p_{jx}x + p_{jx}^2 + p_{jy}^2 - l_y^2)$$

De manera similar está definida la parábola β_i, usando que p_{jy} y p_{iy} son mayores que l_y, es fácil mostrar que es imposible que β_i y β_j tengan un sólo punto de intersección. Entonces podemos concluir que β_j nunca rompe en dos un arco de otra parábola β_i.

La segunda forma es que β_j aparezca entre dos arcos (Figura 2.19). Sean β_i y β_k las

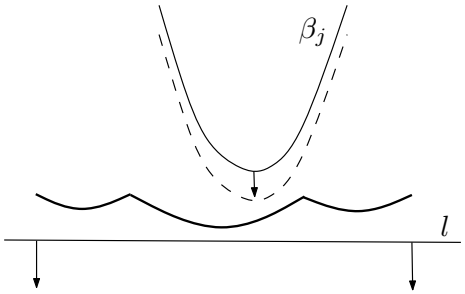


Figura 2.18: Dividiendo la línea de contorno (caso 1)

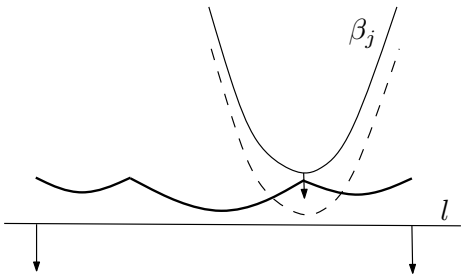


Figura 2.19: Dividiendo la línea de contorno entre dos arcos (caso 2)

parábolas a las que pertenecen estos dos arcos. Sea q el punto de intersección de β_i y β_k en el cual β_j está apareciendo en la línea de contorno. Sin pérdida de generalidad asumimos que la parte que define β_i en la línea de contorno está a la izquierda de q y la de β_k está a la derecha (Figura 2.20).

Entonces existe un círculo C que pasa por p_i, p_j y p_k, los cuales son los puntos de P que definen las parábolas. Este círculo también es tangente a l. Consideremos a los puntos sobre C en orden cíclico el cual inicia en el punto de tangencia con l y recorriendo a favor de las manecillas de reloj, nos encontramos con p_i, p_j y p_k y esto es porque se asumió que β_j, aparece entre los arcos β_i y β_k. Consideramos un barrido hacia abajo manteniendo a C tangente a l. Figura 2.21. Observamos que no existe manera de que C tenga su interior vacío y pase por el punto p_j ya que p_i o p_k podrían caer dentro de C.

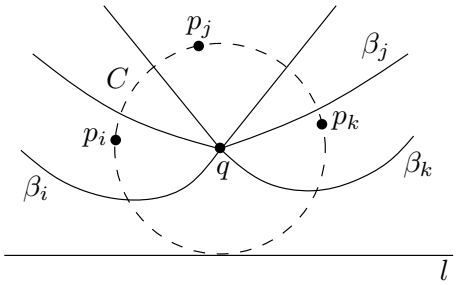


Figura 2.20: β_j empieza a aparecer en la línea de contorno

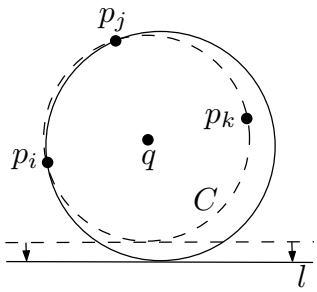


Figura 2.21: Círculo formado cuando la línea de barrido l avanza

Por lo tanto en una vecindad suficientemente pequeña de q, la parábola β_j no aparecería en la curva de contorno cuando l se mueve hacia abajo porque p_i o p_k podrían estar más cerca a l que p_j. □

Una consecuencia inmediata del lema anterior es que la curva de contorno consiste de a lo más $2n-1$ arcos parabólicos: cada punto encontrado aumenta un nuevo arco y divide a un arco existente en a lo más dos arcos y no existe otra manera en que un arco aparezca en la curva de contorno.

El segundo tipo de evento que puede ocurrir en el algoritmo de Fortune, es cuando un arco de la curva de contorno se reduce a un punto y desaparece, el cual llamaremos evento circular. Sea α' el arco que desaparece y sean α y α'' los arcos vecinos de α' antes de desaparecer. Afirmamos que los arcos α y α'' no pueden ser parte de la misma parábola, ya que este caso no puede ocurrir, como se vió en la primera parte de la demostración del lema 3. Entonces como consecuencia tenemos que los arcos α, α' y α'' son definidos por tres parábolas distintas definidas por tres puntos distintos p_i, p_j y p_k de P. En el momento que α' desaparece, las parábolas definidas por estos puntos pasan por un punto en común q, el cual es equidistante de l y de cada uno de los tres puntos. Entonces garantizamos que hay un círculo que pasa por p_i, p_j y p_k con centro q tal que su punto de coordenada y más pequeña esta sobre l. No puede haber un punto del conjunto P en el interior de este círculo, ya que dicho punto estaría más cerca de q que de l, contradiciendo que q esta sobre la línea de contorno. De esto se concluye que el punto q es un vértice del diagrama

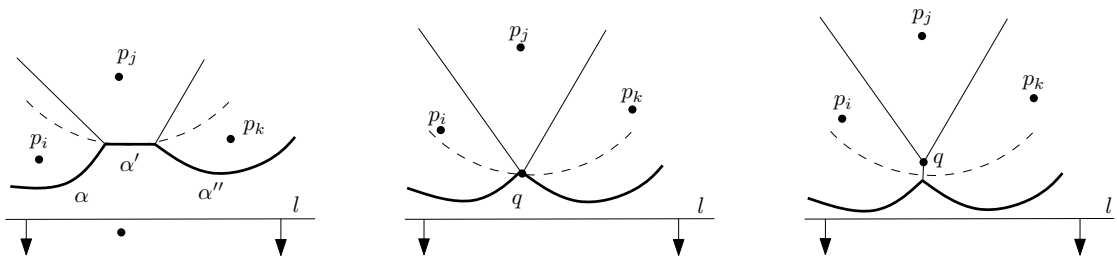


Figura 2.22: Reducción de un arco parabólico

de Voronoi. Esto último reafirma lo que se comentó anteriormente de que los cruces de la curva de contorno determinan el diagrama de Voronoi. Por lo tanto, tenemos que, cuando un arco desaparace de la curva de contorno y dos cruces se encuentran, dos aristas del diagrama de Voronoi se encuentran también.

Finalmente remarcamos que el evento círcular ocurre cuando l alcanza el punto con coordenada y más pequeña que pertenece al círculo definido por los tres puntos de los tres arcos consecutivos en la curva de contorno. Figura 2.22.
Dado lo anterior podemos concluir el siguiente lema:

Lema 4. El evento circular es el único evento que puede provocar que un arco desaparezca de la curva de contorno.

Dado los dos lemas anteriores (3, 4) tenemos identificado cuando la estructura combinatoria de la curva de contorno cambia:

- Cuando ocurre un evento puntual: un nuevo arco en la curva de contorno aparece.
- Cuando ocurre un evento circular: un arco se divide y desaparece.

Además sabemos que como estos dos eventos se relacionan bajo la construcción del diagrama de Voronoi, tenemos que en el evento puntual una nueva arista de Voronoi empieza a crecer y en el evento circular dos aristas crecen hasta formar un vértice de Voronoi.

Para concluir decimos que en el algoritmo para calcular el diagrama de Voronoi de n puntos, haciendo uso de la técnica de barrido, pueden ocurrir dos tipos de eventos, el primer evento es el puntual en el cual empiezan a generarse las aristas del diagrama de Voronoi como fué explicado en el lema 3 y el segundo evento es el círcular que es cuando aparece un nuevo vértice en el diagrama de Voronoi como se vió en el lema 4, finalmente al terminar el barrido de los n puntos obtenemos el diagrama de Voronoi.

2.4 Diagrama de Voronoi Ponderado

Para dar por terminado el estudio de los diagramas de Voronoi estudiaremos el diagrama de Voronoi ponderado [7], el cual nos ayudará a resolver los problemas principales de este trabajo. La definición de este tipo de diagrama difiere del diagrama de Voronoi

ordinario (sin pesos en los puntos) en lo siguiente: en vez de tomar la distancia euclidiana se toma la distancia pesada, que se define de la siguiente manera:

Definición 8. La distancia pesada (d_p) entre dos puntos p_i y p_j es $dp = d(p_i, p_j) - w_{p_j}$ donde d es la distancia euclidiana y w_{p_j} es el peso del punto p_j, el cual está dado por el diámetro del círculo con centro en p_j 2.23.

Nótese que puede ocurrir que $d_p(p_i, p_j) \neq d_p(p_j, p_i)$

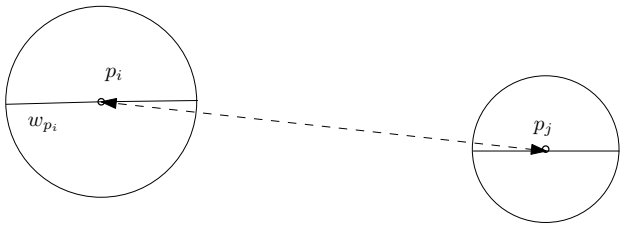


Figura 2.23: Distancia pesada de p_i a p_j.

Otro concepto involucrado en el diagrama de Voronoi es el bisector que en este caso, como los puntos tienen peso, nos lleva a definir al bisector de la siguiente manera:

Definición 9. El bisector pesado B_{ij} entre los puntos p_i y p_j se define como:

$$B_{ij} = \{p \in \mathbb{R}^2 \mid d_p(p, p_i) = d_p(p, p_j).\}$$

También para calcular el diagrama de Voronoi ponderado, *Fortune* hizo uso de la técnica de barrido y considera que cada punto p tiene asociado un peso positivo, el cual es representado por el diámetro de un círculo con centro en p.
El algoritmo de Fortune propuesto para calcular el diagrama de Voronoi ponderado sigue la filosofía de hacer una partición del plano en regiones de manera que cada región consista de los puntos más "cercanos" al punto en cuestión, donde ahora la forma de medir la cercanía es la distancia pesada.

El diagrama de Voronoi resultante tiene una apariencia similar al diagrama de Voronoi ordinario, salvo que ahora los bisectores ya no son líneas rectas sino segmentos de hipérbolas. Figura 2.24.

Observación 4. Si todos los puntos tienen el mismo peso el diagrama de Voronoi ponderado coincide con el diagrama de Voronoi de la sección 2.2. Figura 2.25.

También la región de Voronoi de un punto p_i en el diagrama de Voronoi ponderado puede estar completamente contenida en la región del punto p_j si su peso es mayor que la distancia pesada a p_j. Figura 2.26.

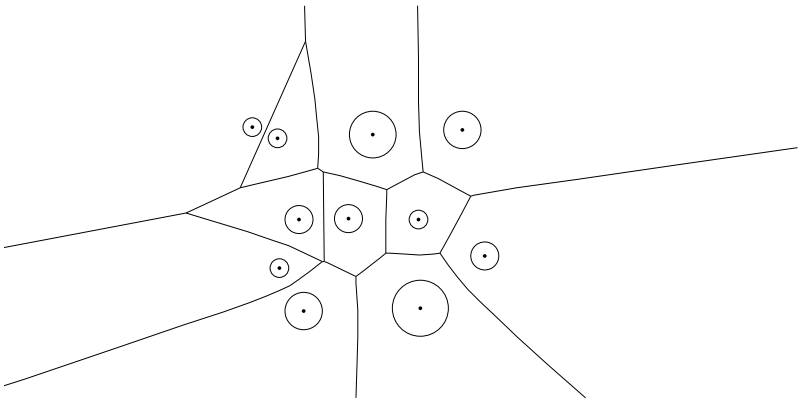


Figura 2.24: Diagrama de Voronoi ponderado de 11 puntos.

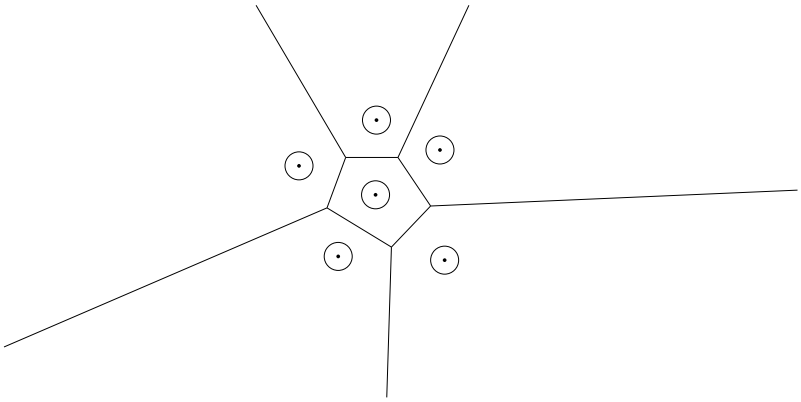


Figura 2.25: Diagrama de Voronoi con puntos del mismo peso.

Para terminar, el algoritmo para calcular el diagrama de Voronoi ponderado es casi idéntico al algortimo de la sección anterior, salvo en la métrica, y en considerar que una región de Voronoi puede estar completamente contenida en otra, lo cual nos lleva a tener un algoritmo de tiempo $O(n\ log\ n)$ para calcular el diagrama de Voronoi ponderado cuando los puntos tienen peso.

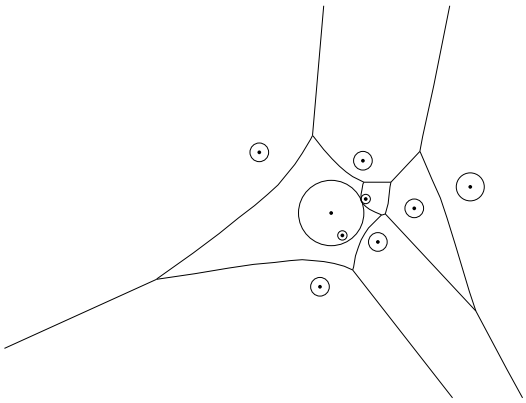


Figura 2.26: La región de Voronoi de un punto puede estar completamente contenida en otra.

2.5 Ubicación de Servicios (*Point Location)*

El problema de ubicación de servicios es un tema fundamental de la geometría computacional. Las áreas de aplicación de este tema son tratadas principalmente en el procesamiento de datos geométricos: graficación por computadora, sistemas de información geográfica (GIS), planeación de movimientos y en el apoyo de diseño computacional (CAD), así como también en el reconocimiento de patrones y análisis estadístico.
En términos generales el problema consiste en: dado un punto p y una subdivisón del espacio en regiones disjuntas, determinar que región de la subdivisión contiene a p [11].

Un ejemplo tradicional es el siguiente: dado un punto p y un polígono arbitrario P representado por un arreglo de n puntos $p_0, p_1, \ldots, p_n = p_0$, determinar si p está dentro o fuera del polígono P. [10]

Desde el punto de vista computacional, cualquier método que de solución al problema de ubicación de servicios puede ser evaluado con respecto a tres medidas [11]:

- El tiempo de búsqueda (*search time*), esto es, el número de operaciones requeridas para localizar un punto en la subdivisión.
- El tiempo de preprocesamiento (*preprocessing time*), es decir, el número de operaciones requeridas para construir las estructuras que se pretenden usar en el algoritmo de búsqueda;
- La cantidad de memoria requerida por nuestro programa para su ejecución.

Algunos conceptos involucrados en el tipo de subdivisión y en las estructuras usadas en este trabajo son los siguientes:

2.5.1 Subdivisiones monótonas

Definición 10. Una subdivisión disjunta S del plano, es una partición del plano en varios polígonos, llamados regiones de S. Figura 2.27.

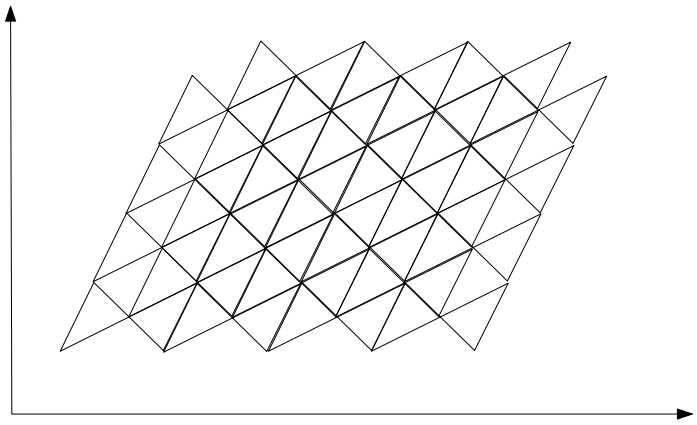


Figura 2.27: Subdivisión del plano.

Definición 11. Una región conexa, R, del plano se dice que es monótona si cualquier recta vertical intersecta a R en un subconjunto conexo. Figura 2.28.

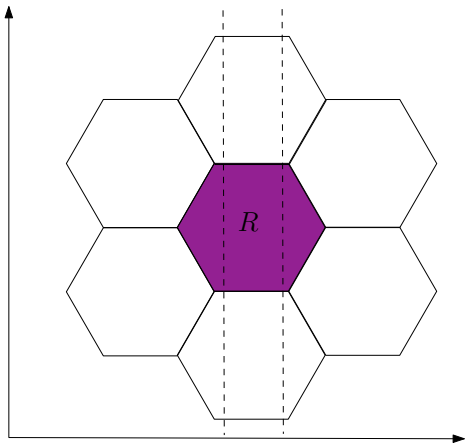


Figura 2.28: Región Monótona

Definición 12. Una subdivisión del plano es monótona si todas sus regiones son monótonas. Figura 2.29.

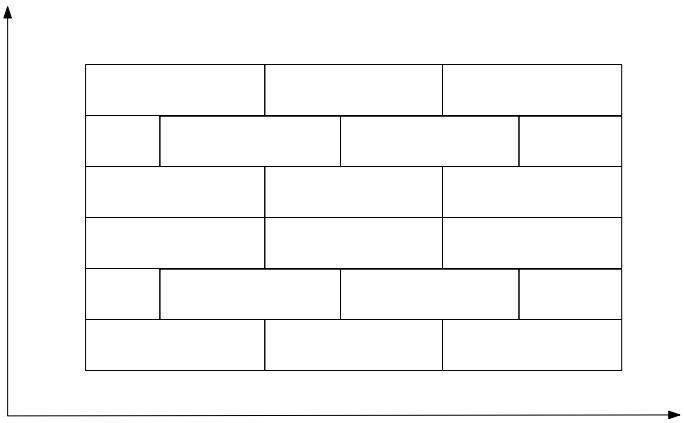


Figura 2.29: Subdivisión monótona.

2.5.2 Descomposición centroidal

Definición 13. Sea T un árbol formado por n vértices de grado a los más 3. Una arista centroidal de T es aquella arista tal que, al eliminarla de T, da como resultado dos árboles de tamaño (número de vértices) a lo más $1 + \frac{2n}{3}$.

Se sabe que para $n > 1$ dicha arista existe y puede ser encontrada en tiempo $O(n)$. [4].

2.6 Localización de puntos

Uno de los problemas estudiados en la geometría computacional es el de localización de puntos en subdivisiones del plano. Se sabe que este problema puede resolverse en tiempo $O(log^2\ n)$ tras un preprocesamiento de tiempo $O(n\ log\ n)$ y requiere $O(n)$ de almacenamiento [11]. La diferencia entre el algoritmo convencional y el que nosotros proponemos radica en que nuestro algoritmo es dinámico y la subdivisión del plano la actualizamos cuando se agrega un nuevo punto, a lo que llamaremos una subdivisión dinámica en el plano. Nuestro algoritmo usa dos métodos para la localización de un punto, el primer método consiste en localizarlo en una subdivisión monótona [9] y en el segundo no requerimos que la subdivisión sea monótona [5].

2.6.1 Ubicación de puntos en una subdivisión dinámica y monótona

El algoritmo mostrado en [9] describe un método para la localización de puntos en una estructura de datos formada a partir de una subdivisón del plano dinámica y monótona S. La principal aproximación está basada en mantener dos árboles de expansión entrelazados, uno para S y otro para su gráfica dual. Las búsquedas de los puntos son resueltas mediante descomposición centroidal del árbol dual. Los árboles construidos son mantenidos mediante las operaciones de inserción y corte mostradas por Sleator y Tarjan [15], llevándonos a

un esquema que garantiza una inserción y borrado de vértices en $O(log\ n)$, inserción y borrado de k aristas en una cadena monótona en $O(log\ n+k)$ y la solución de la búsqueda en tiempo $O(log^2\ n)$ y espacio $O(n)$, donde n es el número de puntos de S considerados en esa etapa del algoritmo. Finalmente la aproximación de los árboles entrelazados la aplican para una localización de puntos en línea (on-line) (cuando S incrementa su número de vértices) mejorando el tiempo de búsqueda a $O(\log n \log\log n)$ y en tiempo amortizado a $O(1)$, donde el tiempo amortizado se refiere a considerar que todas las operaciones requieren el mismo costo.

2.6.2 Ubicación de puntos en una subdivisión dinámica

Debido a que nuestros algoritmos que mostraremos en el siguiente capítulo hacen uso del algoritmo de Voronoi ponderado, obtenemos como resultado subdivisiones del plano no necesariamente monótonas, por lo tanto, la manera de localizar puntos de la sección anterior no satisface nuestras necesidades, por esta razón explicaremos brevemente el método para localización de puntos mostrado en [5].
El algoritmo presentado en [5] para localización de puntos sólo requiere que la subdivisión dinámica sea conexa, mostrando que la solución para el problema de localización de puntos se resuelve en tiempo de $O(log^2\ n)$, espacio de $O(n)$ y en tiempo de actualización en $O(log\ n)$. La inserción y el borrado de k aristas arbitrarias de una cadena incidente a una región son realizadas en tiempo $O(k\ log(n+k))$ y $O(k\ log\ n)$, respectivamente.

Resultados principales

3

Los conceptos y algoritmos del capítulo anterior nos inspiraron para encontrar una solución a los problemas definidos en el capítulo uno.

Sin embargo, existen conjuntos de puntos para los cuales no hay solución aún cuando tengamos las restricciones de monotonía y orden de colores de visita prefijados, en la figura 3.1 se muestra un conjunto de puntos para los cuales, dada cualquier dirección, no existe una poligonal monótona que visite los cuatro colores en el orden (● ● ● ●).

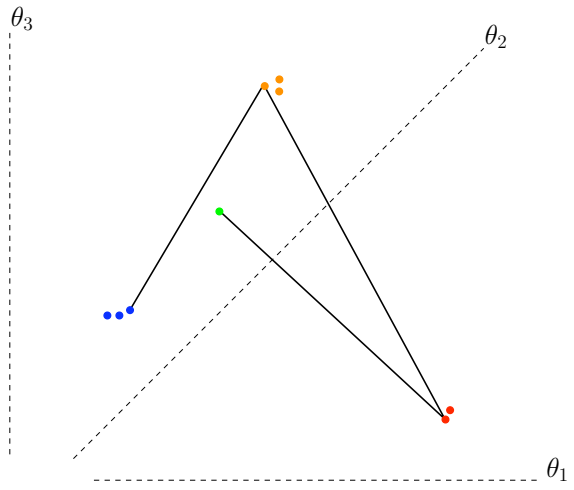


Figura 3.1: No existe una poligonal heterocromática monótona siguiendo el orden (● ● ● ●).

También existen conjuntos de puntos en los cuales se puede encontrar más de una

solución, por ejemplo, en la figura 3.2 observamos que existen dos poligonales x- monótonas de longitud mínima siguiendo el orden (● ● ● ●), llamaremos a estas poligonales solución poligonales óptimas.

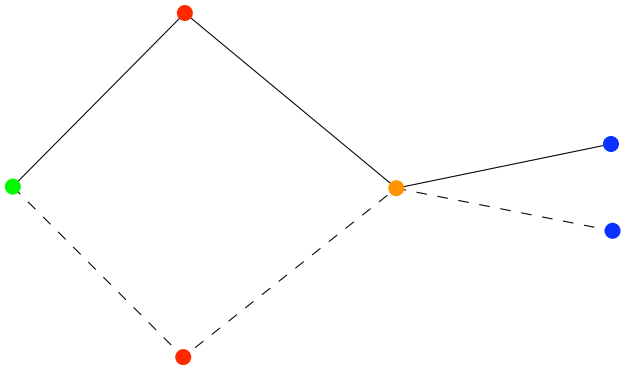


Figura 3.2: Dos poligonales óptimas siguiendo el orden (● ● ● ●).

En las siguientes secciones se mostrará como son resueltos los problemas definidos en el capítulo uno haciendo uso de programación dinámica y localización de puntos en subdivisiones formadas por el diagrama de Voronoi dinámico.

3.1 Poligonal Monótona Orientada y Ordenada (PMOO)

En esta sección mostraremos la solución del problema PMOO, por lo tanto, la poligonal solución que estamos buscando debe ser x-monótona. Por esta razón asumiremos que los puntos están ordenados según su abscisa de forma creciente y para cada clase cromática i etiquetaremos a sus puntos como $p_1(c_i), p_2(c_i), \ldots, p_n(c_i)$ de manera que si $l < m$ entonces la abscisa del punto $p_l(c_i)$ es menor o igual que la abscisa del punto $p_m(c_i)$. Por ejemplo, los puntos $p_1(c_1)$ y $p_2(c_3)$ denotan al primer punto de color c_1 y al segundo punto de color c_3, respectivamente.
Antes de describir el algoritmo que se propone hacemos un paréntesis para observar y mostrar las siguientes propiedades que serán utilizadas por nuestro algoritmo:

Observación 5. Sea P una poligonal heterocromática x-monótona con vértices $p_{i_1}(c_1), p_{i_2}(c_2), \ldots, p_{i_{k-1}}(c_{k-1}), p_{i_k}(c_k)$ de longitud mínima. Entonces $p_{i_k}(c_k)$ es el punto de color c_k más cercano al punto $p_{i_{k-1}}(c_{k-1})$.

Esta observación indica que el último vértice de la poligonal óptima es el punto más cercano al penúltimo vértice de ésta, pero esto no tiene porqué ocurrir para un vértice intermedio. Figura 3.3. Hay que hacer notar que si conocemos a la poligonal óptima hasta un punto, la observación anterior puede ser aplicada.

Lema 5. Sea P una poligonal heterocromática x-monótona con vértices $p_{i_1}(c_1), p_{i_2}(c_2), \ldots, p_{i_{k-1}}(c_{k-1}), p_{i_k}(c_k)$ de longitud mínima. Entonces la poligonal

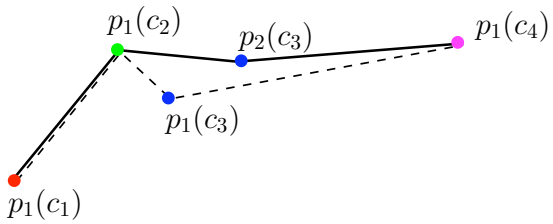


Figura 3.3: $p_2(c_3)$ es un vértice de la poligonal óptima y no es el punto de color c_3 más cercano a $p_1(c_2)$.

x-monótona heterocromática y ordenada de longitud mínima que visita los colores $c_1, c_2, \ldots, c_j$ con $j \leq k$ y termina en $p_{i_j}(c_j)$ tiene vértices $p_{i_1}(c_1), p_{i_2}(c_2), \ldots, p_{i_j}(c_j)$.

Demostración. Se demostrará por contradicción. Supongamos que existe otra poligonal P' que visita los colores $c_1, c_2, \ldots, c_j$ y termina en el punto $p_{i_j}(c_j)$. Sean $q_{i_1}(c_1), \ldots, q_{i_{j-1}}(c_{j-1}), p_{i_j}(c_j)$ los vértices de P'. Si la longitud de P' es menor que la longitud de P que contiene los vértices $p_{i_1}(c_1), \ldots, p_{i_j}(c_j)$, entonces la poligonal que contiene los vértices $q_{i_1}(c_1), \ldots, q_{i_{j-1}}(c_{j-1}), p_{i_j}(c_j), \ldots, p_{i_k}(c_k)$ tiene longitud menor que la longitud de P, lo cual nos lleva a una contradicción. □

Las propiedades anteriores sugieren usar un método de programación dinámica y localización de puntos para resolver el PMOO. La idea principal es la siguiente:

Para cada $p_{i_j}(c_j)$, $1 < j \leq k$, calculamos $l_{i_j}(c_j)$, que es la longitud de la poligonal x-monótona de longitud mínima que visita los colores $c_1, c_2, \ldots, c_j$ y termina en el punto $p_{i_j}(c_j)$. Sean $p_{i_1}(c_1), \ldots, p_{i_{j-1}}(c_{j-1}), p_{i_j}(c_j)$ los vértices de tal poligonal. Entonces creamos un apuntador $prev(p_{i_j}(c_j))$ que apunta a $p_{i_{j-1}}(c_{j-1})$. Usando la observación 5 y el conjunto de valores $\{l_{i_1}(c_1), \ldots, l_{i_j}(c_j)\}$, podemos calcular la longitud de la poligonal x-monótona de longitud mínima (si existe) que termina en un punto de color c_{j+1} a la derecha del punto $p_{i_j}(c_j)$, por lo tanto, si $p_{i_{j+1}}(c_{j+1})$ es el punto de color c_{j+1} buscado, entonces su etiqueta es la siguiente:

$$l_{i_{j+1}}(c_{j+1}) = \min_{i_{j+1}}\{l_{i_j}(c_j) + d(p_{i_j}(c_j), p_{i_{j+1}}(c_{j+1}))\}$$

esta fórmula podría ser aplicada en la combinación de programación dinámica y localización de puntos en diagramas de Voronoi ponderados, de manera que el peso de cada punto $p_{i_j}(c_j)$ sea $l_{i_j}(c_j)$.

3.1.1 Algoritmo para calcular PMOO

Sean $\{p_{i_1}(c_i), \ldots, p_{i_{n_i}}(c_i)\}$ los puntos de color c_i, ordenados según su abscisa de forma creciente y $l_{i_j}(c_i)$ el peso del punto $p_{i_j}(c_i)$. El algoritmo que proponemos consta de

$k-1$ pasos que denotaremos por $ETAPA(i)$, con $i = 2, \ldots, k$. En cada una de ellas sólo se consideran las longitudes $l_{i_1}(c_i), l_{i_2}(c_i), \ldots, l_{i_{n_i}}(c_i)$, las cuales son obtenidas usando $l_{i_1}(c_{i-1}), l_{i_2}(c_{i-1}), \ldots, l_{i_{n_{i-1}}}(c_{i-1})$ y calculamos los apuntadores *prev* de los puntos de color c_i. Inicialmente el peso de todos los puntos de color c_1 $(l_{i_j}(c_1))$ es cero. Una vez que se describió a grandes los cálculos de cada etapa de nuestro algoritmo procederemos a describirlo a detalle: para cada $Etapa(i)$ con $2 \leq i \leq k$, tenemos una lista de puntos de color c_{i-1} con sus correspondientes apuntadores y etiquetas, con los cuales haremos lo siguiente:

Si todos los puntos de color c_i están a la izquierda de todos los puntos de color c_{i-1}, el algoritmo finalizará aquí, pues el problema no tiene solución porque no hay manera de construir una poligonal x-monótona con el orden de colores c_{i-1} y c_i. Figura 3.4.

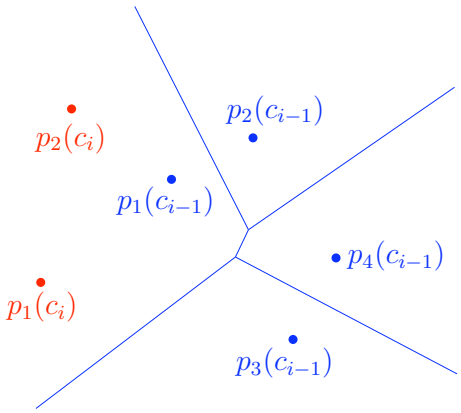


Figura 3.4: No existe poligonal heterocromática x-monótona con orden de visita de colores c_{i-1} y c_i.

En otro caso, realizamos un barrido con una recta vértical de izquierda a derecha parando en cada uno de los puntos de color c_i y calculamos el punto más cercano de color c_{i-1} a su izquierda, figura 3.5.

Este proceso se realiza usando la técnica de barrido dinámico mostrado en [7] que calcula el diagrama de Voronoi ponderado y la estructura de datos presentada en [5] para resolver eficientemente la búsqueda del punto de color c_{i-1} más próximo a la izquierda del punto de color c_i. Para los siguientes puntos de color c_i que aparecen en el barrido (paradas del barrido), $p_{i_j}(c_i)$, $i_j = 2, \ldots, n_i$, hacemos lo siguiente:

1. Si no hay puntos nuevos de color c_{i-1} a la izquierda de $p_{i_j}(c_i)$, (Figura 3.6.a), entonces se obtiene el punto más cercano al punto $p_{i_j(c_i)}$ que está a la izquierda de color c_{i-1} sin modificar el diagrama de Voronoi ya construido hasta el punto $p_{i_{j-1}}(c_i)$, calculamos $l_{i_j(c_i)}$ y la asociamos como peso al punto $p_{i_j(c_i)}$.

2. Si hay nuevos puntos del color c_{i-1} a la izquierda de $p_{i_j}(c_i)$ (Figura 3.6.b) utilizamos el algoritmo de barrido de Fortune [7] para calcular el diagrama de Voronoi dinámicamente y actualizamos de la misma manera la estructura de datos para responder

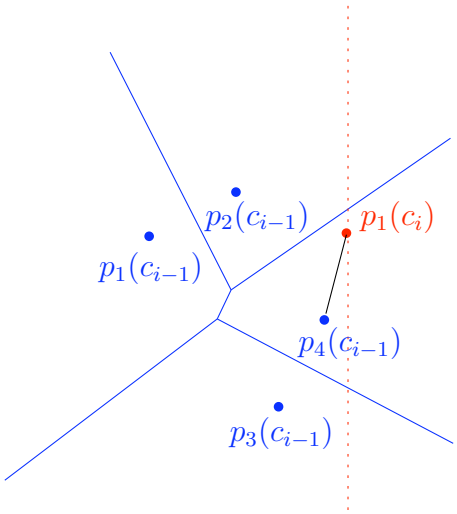


Figura 3.5: Barrido y Localización de un punto.

la pregunta del punto más cercano al punto $p_{i_j}(c_i)$ ($p_{i_{j-1}}(c_{i-1})$ en la Figura 3.6.b), usamos para ello las estructuras de [9], finalmente al punto $p_{i_j}(c_i)$ se le asocia como peso el valor $l_{i_j}(c_i)$.

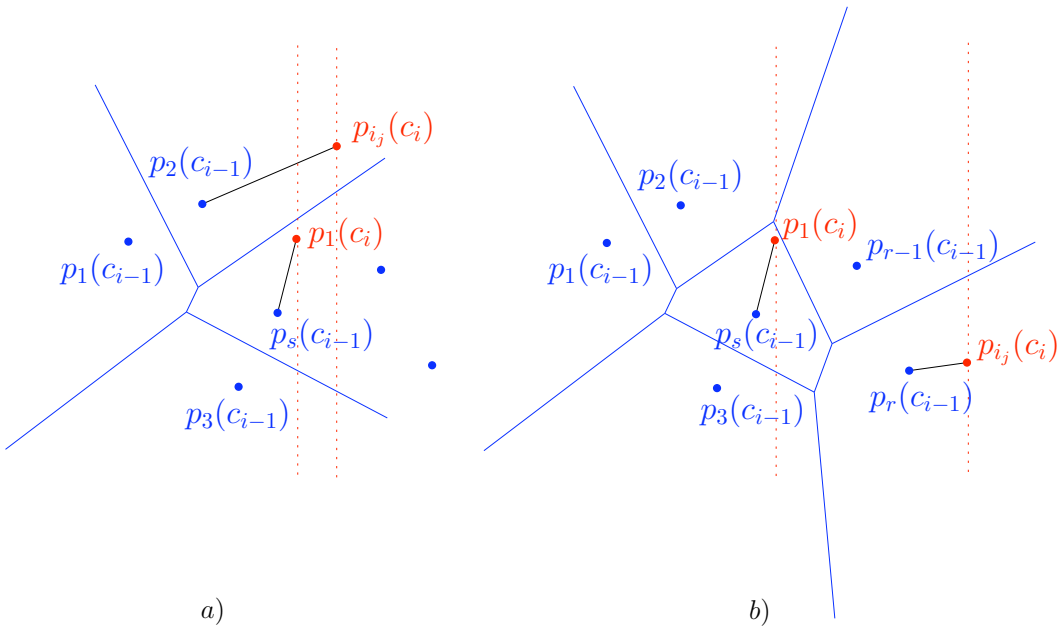


Figura 3.6: Casos en el proceso de barrido.

Hacemos notar que en la etapa inicial se eliminan todos los puntos de color c_2 con abscisa menor a la abscisa de todos los puntos de color c_1. De esta forma, al finalizar el barrido de la lista de color c_2, cada punto tendrá asociado un peso y si elegimos el menor peso tendremos la poligonal bicromática óptima para el conjunto total de puntos.

El método de localización de puntos de [5] sólo require que la subgráfica subyacente de la subdivisión dinámica plana sea conexa, mientras que en [9] la localización es más eficiente, pero si cada cara de la subdivisión es un polígono monótono, lo cual desafortunadamente no nos ayuda porque el diagrama de Voronoi de pesos aditivos no es en general una subdivisión monótona, por lo tanto en la $ETAPA(1)$ usaremos [9].

Durante la ejecución de la $ETAPA(i)$, con $i = 2, \ldots, n_i$, cada vez que la línea de barrido se detiene en un punto de la clase cromática i $(p_{i_j}(c_i))$, usamos la estructura de datos en [5], para obtener la región de Voronoi (del diagrama de Voronoi del conjunto de puntos de color c_{i-1} a la izquierda de $p_{i_j}(c_i)$ que lo contiene). Después asignamos al punto $p_{i_j}(c_i)$ el peso dado por el valor $l_{i_j}(c_i)$ y al apuntador $prev(p_{i_j}(c_i)) = p_{i_{j-1}}(c_{i-1})$, donde $p_{i_{j-1}}(c_{i-1})$ es el punto que está sobre la región de Voronoi Figura 3.7. Observamos que ninguno de los puntos de color c_i que están a la izquierda de los puntos de color c_{i-1} son considerados en el proceso.

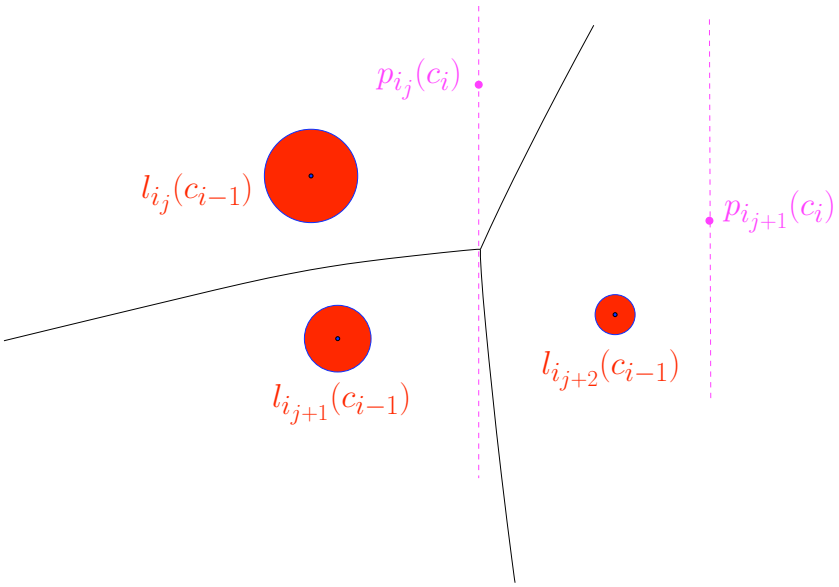


Figura 3.7: Barriendo y localizando al vecino más próximo a la izquierda.

Al finalizar la $ETAPA(k)$ obtenemos a lo más n_k etiquetas y nos quedamos con la que tenga el menor peso. La poligonal óptima puede ser recalculada usando los apuntadores y las etiquetas de derecha a izquierda.

3.1.2 Análisis de la Complejidad

Cada etapa requiere fundamentalmente de dos tareas que se procesan dinámicamente durante el barrido:

- La primera es calcular el diagrama de Voronoi ponderado, lo cual se realiza en $O(n_{i-1}\ log\ n_{i-1})$ tiempo y $O(n_{i-1})$ espacio para $i = 2, \ldots, k$.
- La segunda tarea es construir la estructura de datos necesaria para responder a la localización de puntos en una subdivisión dinámica plana de [9], es decir, para cada punto de color c_i se debe localizar el punto más cercano de color c_{i-1} a su izquierda, dicha localización es obtenida en tiempo $O(log^2(n_{i-1}))$ y $O(n_{i-1})$ espacio con un tiempo de actualización de la estructura de $O(log\ n_{i-1})$.

Por el análisis anterior, el tiempo total que requiere nuestro algoritmo es

$$O(n_1 \log n_1 + n_2 \log^2 n_1) + \ldots + O(n_{k-1} \log n_{k-1} + n_k \log^2 n_{k-1}) \leq O(n \log^2 n)$$

y con respecto al espacio tenemos que las estructuras de datos usadas requieren espacio lineal. Debido al análisis anterior de la complejidad de tiempo y espacio podemos enunciar el siguiente resultado:

Teorema 1. El problema **PMOO** se puede resolver en tiempo $O(n \log^2 n)$ y espacio $O(n)$.

3.2 Poligonal Monótona Ordenada (PMO)

En esta sección abordamos el problema general no orientado, esto es, buscamos la mejor orientación posible que permite construir la poligonal monótona ordena de longitud mínima en esa dirección.

A continuación veremos cómo reducir este problema a un número cuadrático de instancias del problema x-monótono. Consideremos una orientación fija $\theta \in [0, \pi)$. Denotamos por $l(\theta)$ la recta que pasa por el origen en la orientación dada por θ.

Si consideramos una caracterización equivalente de monotonía que dice, dada una poligonal P con vértices en el orden $p_1, p_2, \ldots, p_n$ es monótona respecto a la dirección de $l(\theta)$ si las proyecciones $p'_1, p'_2, \ldots, p'_n$ de los puntos $p_1, p_2, \ldots, p_n$ aparecen en $l(\theta)$ en el mismo orden que en P [14].

Si rotamos la recta $l(\theta)$ alrededor del origen con θ de 0 a π, la ordenación de los puntos proyectados en $l(\theta)$ cambia en determinados instantes que denominamos *eventos*. En la figura 3.8 observamos que la poligonal $P = \{p_1, p_2$ y $p_3\}$ es monótona en las direcciones $l(\theta)$ y $l(\theta_1)$, pero en la dirección $l(\theta_3)$ no lo es, ya que las proyecciones de sus puntos sobre $l(\theta_3)$ cambian su orden a p_1, p_3 y p_2.

Analizando el proceso de rotación obtenemos un total de $O(n^2)$ eventos que corresponden a distintas ordenaciones de las proyecciones. Diremos que dos orientaciones θ y θ' son *equivalentes* si el orden de las proyecciones correspondientes a cada ángulo no cambian. Esto define una partición de $[0, \pi)$ en un conjunto de intervalos, en la cual, para cualesquiera dos ángulos en dicho intervalo estos son equivalentes.

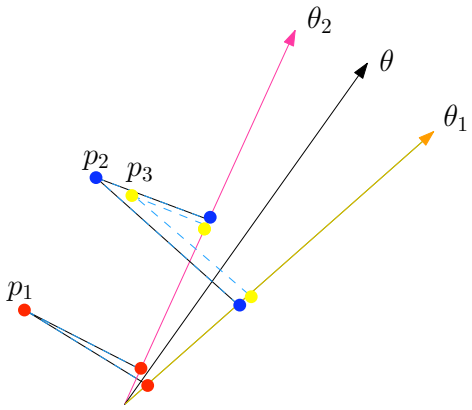


Figura 3.8: Eventos al rotar la recta de dirección $l(\theta)$.

Para cada una de las $O(n^2)$ clases de equivalencia elegimos una orientación y aplicamos el algoritmo de la sección anterior para buscar una poligonal monótona en esa dirección. Notamos aquí que esta solución no cambia para cualquier otra dirección de la misma clase de equivalencia. Si nos permitimos usar esta técnica para cada clase de equivalencia, obtenemos un algoritmo simple de tiempo $O(n^3 \log^2 n)$ y espacio $O(n^2)$.

A pesar de este resultado hay que análizar casos en los cuales al cambiar la dirección de monotonía, se tiene la misma solución, es decir si no ha ocurrido un evento entonces no es necesario hacer otra vez el cálculo de la poligonal dado que es la misma, más adelante explicaremos esto a detalle. Sea $P^\theta = \{p_{i_1}(c_1), p_{i_2}(c_2), \ldots, p_{i_k}(c_k)\}$ una poligonal monótona heterocromática y ordenada de longitud mínima con respecto a la dirección θ que visita un punto de cada clase cromática en el orden $c_1, c_2, \ldots, c_k$. Proponemos una estructura de datos para resolver de manera eficientente el problema PMO usando la información acumudada por los eventos anteriores, esta propuesta nos lleva directamente al siguiente resultado:

Lema 6. Sea $P^\theta = \{p_{i_1}(c_1), \ldots, p_{i_j}(c_j), \ldots, p_{i_k}(c_k)\}$. Entonces la poligonal heterocromática θ-monótona ordenada de longitud mínima que visita los colores $c_1, c_2, \ldots, c_j$ en ese orden y termina en el punto $p_{i_j}(c_j)$, tiene vértices $p_{i_1}(c_1), \ldots, p_{i_j}(c_j)$. Sin embargo la poligonal heterocromática θ-monótona ordena de longitud mínima que visita los colores $c_j, c_{j+1}, \ldots, c_{k-1}, c_k$ en ese orden y comienza en el punto $p_{i_j}(c_j)$ tiene vértices $p_{i_j}(c_j), p_{i_{j+1}}(c_{j+1}), \ldots, p_{i_{k-1}}(c_{k-1}), p_{i_k}(c_k)$.

Demostración. La primera parte de este lema es una consecuencia directa del lema 5, por lo tanto queda mostrado. La demostración de la segunda parte la haremos por contradicción. Supongamos que existe otra poligonal θ-monótona $P^{\theta'}$ con vértices $\{p_{i_j}(c_j), q_{i_{j+1}}(c_{j+1}), \ldots, q_{i_k}(c_k)\}$ que visita los colores $c_j, c_{j+1}, \ldots, c_k$ y en ese orden. Si la longitud de P^θ es menor que la longitud de la poligonal con vértices $\{p_{i_j}(c_j), \ldots, p_{i_{k-1}}(c_{k-1}), p_{i_k}(c_k)\}$, entonces la poligonal con vértices $\{p_{i_1}(c_1), \ldots, p_{i_j}(c_j), q_{i_{j+1}}(c_{j+1}), \ldots, q_{i_k}(c_k)\}$ es menor que $P^{\theta'}$, lo que lleva a una contra-

dicción. □

El lema 6 implica que si se conoce el último punto de la poligonal óptima, digase $p_{i_j}(c_j)$, está puede ser calculada recursivamente usando los apuntadores de derecha a izquierda, esto nos lleva a no tener que recalcular la poligonal por cada cambio de dirección que ocurra, en vez de esto proponemos mantener una estructura de datos al aplicar el barrido rotacional dado por $l(\theta)$.

La nueva estructura de datos es la siguiente:
El estado del barrido se mantendrá en un arreglo de tamaño $O(n^2)$. Dando como valor inicial $\theta = 0$. Aplicamos dos veces el algoritmo para resolver $PMOO$, de izquierda a derecha y de derecha a izquierda y guardamos la siguiente información por cada punto $p_{i_j}(c_j)$:

- Dos etiquetas, $l^-(p_{i_j}(c_j))$ y $l^+(p_{i_j}(c_j))$; las cuales marcan la longitud de la poligonal óptima que termina en el punto $p_{i_j}(c_j)$ y con orden de visita de colores $c_1, c_2, \ldots, c_j$ y la longitud de la poligonal óptima que empieza en el punto $p_{i_j}(c_j)$ y con orden de visita de colores $c_{j+1}, c_{j+2}, \ldots, c_k$, respectivamente.

- Dos listas ordenadas, $list(p_{i_j}(c_j), izquierda)$ y $list(p_{i_j}(c_j), derecha)$ que contienen a los puntos de color $c_1, c_2, \ldots, c_{j-1}$ y $c_{j+1}, c_{j+2}, \ldots, c_k$ localizados a la izquierda y derecha del punto $p_{i_j}(c_j)$ respectivamente que pertenecen a P. Las listas son almacenadas en orden incremental con respecto a θ.

- Dos apuntadores, $prev(p_{i_j}(c_j))$ y $next(p_{i_j}(c_j))$. El apuntador $prev(p_{i_j}(c_j))$ apunta al elemento más cercano a $p_{i_j}(c_j)$ en la lista $list(p_{i_j}(c_j), izquierda)$, esto es, señala al punto de color c_{j-1} a la izquierda de $p_{i_j}(c_j)$ que minimiza $d(p_{i_j}(c_j), p_{i_{j-1}}(c_{j-1})) + l^-(p_{i_{j-1}}(c_{j-1}))$ sobre todos los puntos $p_{i_{j-1}}(c_{j-1})$ de color c_{j-1} a la izquierda de $p_{i_j}(c_j)$. El apuntador $next(p_{i_j}(c_j))$ apunta al elemento más cercano a $p_{i_j}(c_j)$ en la lista $list(p_{i_j}(c_j), derecha)$, esto es, señala al punto de color c_{j+1} a la derecha de $p_{i_j}(c_j)$ que minimiza $d(p_{i_j}(c_j), p_{i_{j+1}}(c_{j+1})) + l^+(p_{i_{j+1}}(c_{j+1}))$ sobre todos los puntos $p_{i_{j+1}}(c_{j+1})$ de color c_{j+1} a la derecha de $p_{i_j}(c_j)$.

El arreglo es inicializado con $\theta = 0$ y se hará uso del algoritmo $PMOO$ para resolver el problema PMO presentado en la sección 3.1.1. Siguiendo un escaneo lineal en el arreglo haciendo uso de todos los apuntadores fijados a a cada elemento. Cada punto tiene asignadas dos listas de complejidad lineal, así que la complejidad total de la esctructura propuesta es de $O(n^2)$.

Finalmente el pre-procesamiento es el siguiente: mientras incrementamos a θ, mantenemos la nueva estructura propuesta y la pareja (θ, l_θ), donde l_θ es la longitud de la poligonal óptima para la dirección marcada por el ángulo θ. Cada vez que un nuevo evento ocurre, decimos que $(p_{i_j}(c_j), p_{i_t}(c_t))$, borraremos o insertaremos un valor en la lista de puntos involucrados y actualizamos a los apuntadores. Esto se realiza en tiempo $O(\log n)$, después de este evento, las posiciones relativas de los otros puntos no cambian, pero sus listas podrían ser modificadas. Figura 3.9. Entonces para actualizar la estructura de datos, es necesario actualizar las listas $list(p_{i_m}(c_m), izquierda)$ (y/o $list(p_{i_m}(c_m), derecha)$) y los

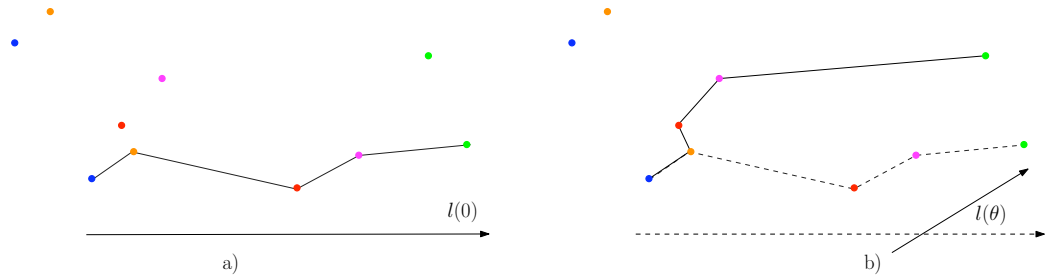


Figura 3.9: a) La solución para $\theta = 0$ y orden ● ● ● ● ●. b) La solución después del primer evento.

apuntadores para los otros puntos de color c_m, $m \neq j, t$ que pueden ser afectados. Por lo tanto la estructura de datos puede ser mantenida en tiempo $O(n \log n)$ por evento. Al finalizar el barrido se obtiene la orientación óptima θ^* y la longitud correspondiente l_{θ^*}. La poligonal óptima puede ser recalculada usando el algoritmo de $PMOO$ para $\theta = \theta^*$.

En consecuencia, tenemos el siguiente resultado:

Teorema 2. El problema **PMO** cuando la dirección de monotonía no está fijada, se puede resolver en tiempo $O(n^3 \log n)$ y espacio $O(n^2)$.

Finalmente a pesar de tener el teorema 2 hacemos la observación de que el problema PMO puede ser resuelto fácilmente en $O(n \log n)$ para dos y tres colores. Esto se logra a partir de que una poligonal con dos o tres vértices es siempre monótona en alguna dirección. Por ejemplo para el caso en que sean dos colores c_1 y c_2, el problema se reduce en encontrar el vecino más cercano para cada punto de color c_1 y para el caso en que sean tres colores c_1, c_2 y c_3 el problema se reduce en encontrar para cada punto de color c_2 sus vecinos más cercano de color c_1 y de color c_2.

Como se mencionó en el capítulo uno, otras variantes de estos problemas es quitando una o las dos restricciones (monotonía y orden). Así que a continuación explicaremos las soluciones obtenidas para los problemas faltantes definidos en el capítulo uno.

3.3 Poligonal Ordenada (PO)

Este problema puede resolverse con una estrategia de localización de puntos incremental. Para ello se tiene un algoritmo muy parecido al de la sección 3.1.1, es decir, se trata también de un algoritmo dinámico el cual consiste en aplicar una serie de etapas. Otra similitud con este algoritmo es la notación, dado que, usaremos la misma forma para identificar a los puntos de distintas clases cromáticas, el peso y el apuntador *prev* para la reconstrucción de la poligonal. Las etapas de nuestro algoritmo son las siguientes:

- $ETAPA(1)$:

  Calcular el diagrama de Voronoi de los puntos $p_{i_j}(c_1)$ con $1 \leq i_j \leq n_1$, donde n_1 es el número de puntos de color c_1 y buscar para cada uno de ellos el punto $p_{i_{j'}}(c_2)$ con

$1 \leq i_{j'} \leq n_2$, más cercano, observemos que para esta búsqueda ya no es necesario usar una estructura de localización de puntos, dado que no importa que el punto más próximo de color c_2 cumpla con estar en una dirección requerida (izquierda, derecha, arriba, abajo) con respecto al punto de color c_1, porque en la definición del problema PO la restricción de monotonía fue eliminada. Figura 3.10.

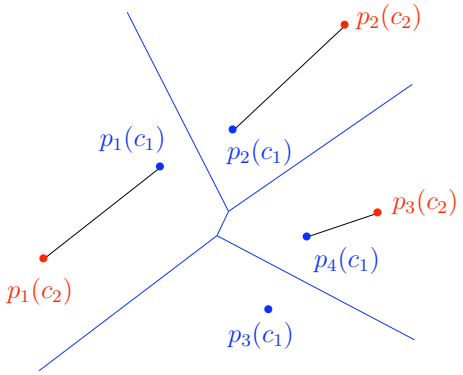


Figura 3.10: Etapa 1 de PO

Una vez que se encontró el vecino más proximo de cada punto de color c_2, se le asigna como peso, la longitud del segmento que lo une con el punto de color c_1 ($l_{(c_2;r)}$). Figura 3.11.

- $ETAPA(i)$, $i = 2, \ldots, k$:

  Para cada color c_i, con $i = 2, \ldots, k$ realizamos lo siguiente:

  Como ya disponemos de una lista de los puntos de color c_{i-1}, con sus correspondientes pesos, obtenemos el diagrama de Voronoi ponderado de los puntos de color c_{i-1} y para cada uno de estos puntos buscamos su vecino más próximo de color c_i y al igual que en la etapa anterior, al punto de color c_i le asignamos la longitud del segmento que une al punto de color c_{i-1} y c_i más el peso del punto de color c_{i-1}. Y de esta manera al terminar el algoritmo vamos a tener a lo más k poligonales heterocromáticas y elegimos la poligonal de longitud mínima.

3.3.1 Análisis de la Complejidad

Cada etapa requiere fundamentalmente calcular el diagrama de Voronoi, en la etapa uno el diagrama de Voronoi convencional y, en las etapas 2 hasta k el diagrama de Voronoi ponderado. Para realizar estos cálculos usamos el método propuesto en [7], el cual en tiempo $O(n \log n)$ obtiene dichos diagramas. Entonces al sumar la complejidad que implica cada etapa, el tiempo total que requiere nuestro algoritmo es

$$O(n_1 \log n_1) + O(n_2 \log n_2) + \cdots + O(n_k \log n_k) = (O(k(nlogn))).$$

Dado lo anterior podemos enunciar el siguiente resultado:

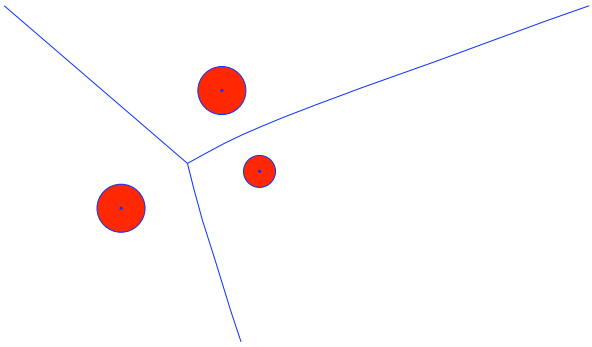


Figura 3.11: Diagrama de Voronoi ponderado de los puntos de color 2 de la figura 3.10

Teorema 3. El problema **PO** se puede resolver en tiempo $O(k(n \log n))$ y espacio $O(n)$.

3.4 Poligonal Monótona Orientada (PMOrientada)

Recordemos que al definir este problema, en el capítulo uno la restricción de orden fue eliminada, pero conservamos la restricción de monotonía, considerándola fija. En este caso, un algoritmo simple para resolver este problema con k colores distintos y de orden lineal lo obtenemos al considerar todas las permutaciones de los k colores y aplicar el algoritmo de PMOO. Por lo tanto tenemos el siguiente teorema:

Teorema 4. El problema **PMOrientada** puede reolverse en tiempo $k!(O(n \log^2 n))$ y espacio $O(n^2)$.

Observación: Para valores de k asintóticamente crecientes se tiene la sospecha que se trata de un problema NP-duro.

3.5 Poligonal Monótona (PM)

La definición del problema PM nos obliga a no considerar el orden, pero si la dirección de monotonía. Así que una manera de solucionar PM es aplicando el algoritmo que resuelve el problema PMO por cada permutación posible de los k colores distintos con $k \in O(n)$.

Teorema 5. El problema **PM** puede resolverse en tiempo $k!(O(n^3 \log n))$ y espacio $O(n^2)$.

De igual manera como para el problema PMOrientada, se tiene la sospecha que para valores de k, asintóticamente crecientes, se trata de un problema NP-duro.

Conclusiones 4

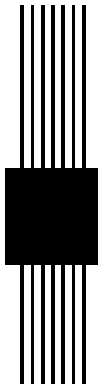
En este trabajo de tesis logramos probar que el problema de encontrar la poligonal x-monótona y ordenada de longitud mínima (PMOO) puede ser resuelto en tiempo $O(n \log^2 n)$ y $O(n^2)$ espacio, y para encontrar la poligonal monótona orientada y ordenada de longitud mínima (PMO) el algoritmo propuesto lo hace en tiempo $O(n^3 \log n)$ y $O(n^2)$ espacio. Además para el problema de encontrar la poligonal ordenada de longitud mínima (PO) se mostró un algoritmo que la obtiene en $O(k(n \log n))$ tiempo y $O(n)$ espacio. Dados estos resultados se derivaron otras soluciones no tan óptimas pero que resuelven el problema de encontrar la poligonal monótona orientada de longitud minima (PMOrientada) y encontrar la poligonal monótona de longitud mínima (PM).

4.1 Trabajo a futuro

Durante el estudio de los problemas PMOO y PMO, encontramos que no siempre existe una solución cubriendo los k-colores o respetando el orden, por tanto algunos de los problemas que podrían trabajarse a futuro son:

- Si no hay solución monótona orientada y ordenada. ¿Cuál es el mayor número de colores que podemos conseguir, considerando o no la longitud mínima?
- Supongamos que en la situación anterior se permiten "retrocesos". ¿Cuál es el menor número de retrocesos?
- Encontrar algoritmos con mejor complejidad para los problemas PMOrientada y PM.

Referencias



[1] M. K. A. Kaneko and K. Suzuki. Three edge-disjoint multicolored spanning trees in complete graphs. *Preprint*, 2002.

[2] E. M. Arkin, R. Connelly, and J. S. Mitchell. On monotone paths among obstacles with applications to planning assemblies. In *SCG '89: Proceedings of the fifth annual symposium on Computational geometry*, pages 334–343. ACM, 1989.

[3] R. A. Brualdi and S. Hollingsworth. Multicolored forests in complete bipartite graphs. *Discrete Math.*, 240(1-3):239–245, 2001.

[4] B. Chazelle. A theorem on polygon cutting with applications. *Proc. 23rd Ann. IEEE Sympos. Found. Comput. Sci.*, pages 339–349, 1982.

[5] S. W. Cheng and R. Janardan. New results on dynamic planar point location. *SIAM J. Comput.*, 21(5):972–999, 1992.

[6] J. M. Díaz-Bánez, F. Gómez, and F. Hurtado. Approximation of point sets by 1-corner polygonal chains. *INFORMS J. on Computing*, 12(4):317–323, 2000.

[7] S. Fortune. A sweepline algorithm for voronoi diagrams. In *SCG '86: Proceedings of the second annual symposium on Computational geometry*, pages 313–322, New York, NY, USA, 1986. ACM.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[9] M. T. Goodrich and R. Tamassia. Dynamic trees and dynamic point location. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 523–533, New York, NY, USA, 1991. ACM.

[10] K. Hormann and A. Agathos. The point in polygon problem for arbitrary polygons. *Computational Geometry*, 20(3):131–144, 2001.

[11] D. T. Lee and F. P. Preparata. Location of a point in a planar subdivision and its applications. In *STOC '76: Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 231–235, New York, NY, USA, 1976. ACM.

[12] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng. On trip planning queries in spatial databases. In *SSTD*, pages 273–290, 2005.

[13] C. Papadimitriou. The euclidean travelling salesman problem is np-complete. *Princeton University*, 189(75).

[14] F. P. Preparata and K. J. Supowit. Testing a simple polygon for monotonicity. *Inf. Process. Lett.*, 12(4):161–164, 1981.

[15] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 114–122, New York, NY, USA, 1981. ACM.

[16] Y. Yan, D. Lemire, and M. Brooks. Monotone pieces analysis for qualitative modeling. In *Proceedings of ECAI MONET'04*, 2004.