Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

UNIVERSIDAD DEL BÍO-BÍO, CHILE
Facultad de Ciencias Empresariales
Departamento de Sistemas de Información
Escuela de Ingeniería Civil en Informática

UTILIZACIÓN DE ALGORITMOS DE INTERSECCIONES BICROMÁTICAS ENTRE DOS CONJUNTOS DE SEGMENTOS: ROJOS Y AZULES

PROYECTO DE TÍTULO PRESENTADO POR
JOEL SEBASTIÁN TORRES CARRASCO Y CRISTIAN MANUEL VALLEJOS VEGA
PARA OBTENER EL TÍTULO DE INGENIERO CIVIL EN INFORMÁTICA
DIRIGIDA POR: BRUNNY ANGÉLICA TRONCOSO PANTOJA

2014

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

a DIOS...

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Agradecimientos

Este proyecto siempre estuvo sustentado por el ánimo, apoyo, estímulo y comprensión de nuestros seres amados. En este apartado se desea corresponder a su aporte en nuestra vida, que también está plasmado en este trabajo. Se comenzará a título personal de cada integrante y, posteriormente, a los agradecimientos conjuntos.

Primeramente a DIOS, quien es mi salvador, soporte y guía. A mi familia por su amor, apoyo, comprensión y muchas más cosas que me han dado en toda mi vida; gracias papá y mamá por estar conmigo, gracias a mis hermanos Fernando Esteban y Moisés, por entretenerme siempre, y con ellos a toda mi familia Carrasco, mi Papi y Mami (mis abuelos) quienes formaron una gran y hermosa familia, y a mi familia Torres, gracias a ellos, a mis tíos, tías, primos y primas. A mi amada Keyti Salazar Ramírez por su amor y ánimo en todo este tiempo, gracias mi amor. A la Primera Iglesia Bautista de Lota, a cada hermano y hermana que en ella hay, que me han visto crecer y formarme hasta hoy, por todo su apoyo, comprensión y bendiciones. Gracias a Cristian Vallejos Vega por ser mi compañero y amigo en este proyecto y en toda nuestra aventura universitaria, gracias por luchar a mi lado, pasara lo que pasara.

JOEL SEBASTIÁN TORRES CARRASCO

En primer lugar, quiero agradecer infinitamente a Dios, quien día a día me da razones para vivir y seguir adelante, y me ha entregado valiosas oportunidades. A mi amada familia, que sin ellos quién sabe dónde estaría perdido en este mundo; a mi padre, por escucharme y sacarme más de una sonrisa cada vez que no me iba tan bien; a mi madre, por esperar semana a semana mi regreso y entender cada vez que no me era posible volver a casa; a ambos por todo el amor que desbordan y me entregan; a mis hermanas, Camila

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

y Emily, que siempre me hacen ser más feliz con sus locuras y son las más ”hermanables” de la vida (mención especial a Emily, que me salvó durante varios meses prestándome su notebook); a mi abuela, que siempre ha estado presente para mí. A mis tíos/padrinos, Luis y Elsa, y sus hijos, que me recibieron en su hogar surante 5 años y me dieron una segunda famlia, adoptándome como un miembro más de la suya. A todos mis amigos, por acompañarme siempre, en especial a José, mi mejor amigo, que ha estado conmigo en las buenas y, sobre todo, en las mala; también a mi amiga Marita, por compartir muchas conversaciones enriquecedoras y ayudarme cuando la necesité. A mi gran amigo de la universidad, Joel Torres Carrasco, con quien siempre nos acompañamos, compartiendo muchos trabajos y anécdotas juntos, y quien emprendió este proyecto junto a mí. A mis compañeros de Jupach y a todos los niños que me entregan su cariño semana a semana, gracias por darme mil alegrías y hacerme sentir que sirvo de algo en este mundo; y en general a todas aquellas personas que han orado por mí y han estado pendientes de mi avance en la U. No puedo dejar fuera de estos agradeciemientos a una persona que en el último tiempo se encargó de alegrar mis días y de darme ánimos para continuar; se dio el tiempo de irme conociendo y compartiendo conmigo, gracias Fabiola. Finalmente, cómo no agradecer a mi nueva familia, a mis queridos amigos del departamento: Cele, Lalo, Caro y Maka; mil gracias por todos los buenos momentos.

CRISTIAN MANUEL VALLEJOS VEGA

También agradecemos a la persona que nos dio esta oportunidad y nos ha guiado en el proyecto todo este tiempo, gracias a la profesora Brunny Troncoso Pantoja, por aceptarnos en esta aventura, entendernos, confortarnos en los momentos difíciles y retarnos ante los momentos de porfía. A los docentes Alejandra Segura, Pedro Rodríguez y Gilberto Gutierrez que nos aconsejaron académica y personalmente cuando más lo necesitamos.

A nuestros amigos y amigas, que han estado animándonos a terminar este proyecto, a los ”siete fantásticos”, nuestra hermandad, conformados por Paola Torres Ferrada, Renato Ávila Momberg, Alejandro Neira San Martín, Heber Gálvez Ojeda, Jorge Elgueta Morales y nosotros dos, por estos cinco años y un poco más de amistad que van más allá de ser solo compañeros de carrera, gracias por su amistad incondicional.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Agradecemos a aquellos que nos brindaron ayuda, a Andrea Vidal Riveros, Luis Cabrera Crot y Raúl Arredondo Flores, y con ellos a todos los demás que en alguna medida nos ayudaron a lograr terminar este proyecto. Gracias a todos ellos por creer en nuestro trabajo, darnos fuerzas y tendernos una mano cuando fue necesario.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Abstract

This current graduation project examines studies of an algorithm proposed by academics from the University of Sevilla, who present a new algorithm in order to resolve a specific variation of a problem with the bichromatic intersections of segments. The issue in concern is to report the set of blue segments that are intersected by at least one red segment, and similarly, the set of red segments that are intersected by at least one blue segment.

Furthermore, the algorithm is implemented by utilizing the techniques and methods of computational geometry. This implementation will permit one to observe the behavior of the algorithm in different situations by means of the experimentation with different sets of data in distinct distributions and quantities of data.

Finally, this work contributes an adaptable implementation for future applications in the same field, providing information about the algorithm experience with real data. Therefore, this creates a better knowledge of the behavior of this new algorithm and its potential uses in other areas of study.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Resumen

En el presente proyecto de titulación se realiza un estudio de un algoritmo propuesto por académicos de la Universidad de Sevilla, quienes presentan un nuevo algoritmo para resolver una variación específica de un problema de intersecciones bicromáticas de segmentos. El problema a solucionar es reportar el conjunto de segmentos azules que son intersectados por al menos un segmento rojo y, de forma análoga, el conjunto de segmentos rojos que son intersectados por al menos un segmento azul.

Además, se implementa el algoritmo utilizando las técnicas y métodos de la geometría computacional. Esta implementación permitirá observar el comportamiento del algoritmo en diferentes situaciones mediante la experimentación con diferentes conjuntos de datos en distintas distribuciones y volúmenes de datos.

Finalmente, este trabajo aporta una implementación adaptable para futuras aplicaciones en la misma área, entregando información de la experiencia del algoritmo con datos reales. Por lo tanto, se crea conocimiento sobre el comportamiento de este nuevo algoritmo y sus potenciales usos en otras áreas.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Índice general

1. Introducción **1**

2. Conceptos Fundamentales desde la Teoría **4**

2.1. Teoría de Grafos 4
2.2. Teoría de Geometría Computacional 7
2.2.1. Definición 7
2.2.2. Orígenes 7
2.2.3. Aplicaciones de la Geometría Computacional 8
2.3. Cierre Convexo 8
2.3.1. Definición 9
2.4. Cáliper Rotatorio 14
2.4.1. Historia y Evolución 14
2.4.2. Definiciones 15
2.4.3. Algoritmo 16
2.5. Arreglo Dual de Líneas 19
2.5.1. Arreglo de Líneas 19
2.5.2. Dualidad 20
2.6. Conclusión de los Conceptos Fundamentales de la Teoría de la Geometría Computacional 21

3. Definición del Proyecto **24**

3.1. Objetivos del proyecto 24
3.1.1. Objetivos Generales: 24
3.1.2. Objetivos Específicos: 24

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

3.2. Ambiente de Ingeniería de Software 25
3.3. Definiciones, Siglas y Abreviaciones 26

4. Algoritmo BR **28**
4.0.1. Algoritmo BR 30
4.0.2. Explicando el Algoritmo BR 31
4.0.3. Parte 1: Cierre convexo $CH(R)$ y conjuntos $V(Ge)$ y $V(Gi)$ 34
4.0.4. Parte 2: Obtener $E(\overline{Ge})$ 35
4.0.5. Parte 3: Obtener $E(\overline{Gi})$ 37
4.0.6. Parte 4: Obtener $\{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$ 41
4.0.7. Parte 5: Obtener Sr 43
4.1. Conclusión del Algoritmo BR 44

5. Especificación de Requerimientos de Software **46**
5.1. Alcances 46
5.2. Objetivo del software 47
5.2.1. Objetivo General 47
5.2.2. Objetivos Específicos 47
5.3. Requerimientos Específicos 48
5.3.1. Requerimientos Funcionales del sistema 48
5.3.2. Interfaces externas de entrada 49
5.3.3. Interfaces externas de Salida 49
5.3.4. Atributos del producto 50

6. Diseño **51**
6.1. Diagrama de Clases 51
6.1.1. Análisis de Diseño 51
6.2. Diseño de Archivos de Entrada 55
6.2.1. Estructura del Archivo 56
6.2.2. Tipos de Datos y Formato 57
6.2.3. Extensión del Archivo 58
6.2.4. Usos Y Beneficios 58
6.3. Diseño interfaz y navegación 58

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

7. Construcción de BR-Software **63**
7.1. Introducción 63
7.2. Orden Rotacional 64
7.2.1. Caso de Estudio 67
7.3. Intersecciones Segmento-Segmento 73
7.4. Partición de Polígonos Convexos 75
7.5. Ubicación de un Punto en un Polígono 79
7.6. Localización de puntos en un polígono convexo dividido en regiones convexas 81
7.7. Desarrollo del Algoritmo BR 84

8. Pruebas de Software **90**
8.1. Prueba de Software 90
8.2. Resultados de Pruebas de Software 91
8.3. Conclusiones de las Pruebas de Software 92

9. Experimentación **93**
9.1. Metodología de Experimentación 93
9.2. Detalle de los Experimentos 94
9.2.1. Desarrollo de la experimentación 94
9.2.2. Resultados de la Experimentación 95
9.2.3. Análisis de la Experimentación 99

10. Líneas de Trabajo Futuro **105**

11. Resumen de Esfuerzo Requerido **109**

12. Conclusiones **111**

Referencias **113**

A. PLANIFICACIÓN INICIAL DEL PROYECTO **117**

B. ESPECIFICACIÓN DE LAS PRUEBAS **119**
B.1. Pruebas de Unidad 119

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

C. ESPECIFICACIÓN DE LOS EXPERIMENTOS **121**
C.1. Conjuntos de Datos usados en la Experimentación 121

D. CÓDIGO DE LA IMPLEMENTACIÓN **132**

E. MANUAL DE USUARIO DEL BR-SOFTWARE **208**

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Índice de figuras

1.1. Tipos de intersección, (a) Intersección Bicromática y (b) Intersección Monocromática 2

2.1. (a) Vértice y (b) Arista 5
2.2. Ejemplo de un Grafo no Dirigido 6
2.3. Ejemplos de aplicaciones en Teoría de Grafos 6
2.4. Ejemplos de polígono convexo y no convexo 9
2.5. Puntos en el plano con su cierre convexo 10
2.6. Ordenamiento angular de los puntos de la nube respecto a un punto específico 11
2.7. Secuencia de búsqueda del cierre convexo por Graham Scan 12
2.8. Cáliper Rotatorio (Shamos, 1978) 16
2.9. Cáliper Rotatorio (Shamos, 1978) 17
2.10. Forma de giro de un Cáliper Rotatorio (Toussaint, 1983) 22
2.11. Arreglo de líneas de 10 líneas. $V = 45$, $E = 100$ y $F = 56$. 23
2.12. Perturbación de arreglos de líneas no simples. 23

4.1. Ejemplo de Sep(S) (Cortés et al., 2012) 29
4.2. 4.2a es $E(\overline{Ge})$, 4.2b es $E(\overline{Gi})$ y 4.2c es $\{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$ 33
4.3. Cierre Convexo $CH(R)$ 34
4.4. Puntos externos e internos al Polígono 35
4.5. Lema 2, Intersección de un Polígono(Cortés et al., 2012) 37
4.6. Instancia de las regiones convexas creadas hasta la iteración 5 39

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

4.7. Ejemplo de búsqueda de $E(\overline{Gi})$. En (a) se puede ver dos grafos (rojo y azul). En (b) se observa $CH(R)$. En (c) ya se realizaron las particiones correspondientes. En (d) se resalta los segmentos azules que cruzan de una región a otra (se eliminó segmentos rojos para mejorar visibilidad). En (e) se resalta $E(\overline{Gi})$ dentro de $CH(R)$ 40
4.8. Selección de aristas $\overline{uv}$ 42
4.9. 4.9a es $E(\overline{Ge})$, 4.9b es $E(\overline{Gi})$ y 4.9c es $\{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$ 44

6.1. Diagrama de Clases 54
6.2. Representación de las Listas de vértices y aristas 56
6.3. Estructura de Archivo de Entrada de Datos 57
6.4. Interfaz de Usuario - Estructura 59
6.5. Interfaz de Usuario - Versión Final 60
6.6. Ícono del Software 61
6.7. Diagrama de Jerarquía 62

7.1. Vértices de Tangencia de p sobre P (Devadoss y O'Rourke, 2011) 64
7.2. Polígono Convexo P y Conjunto de Puntos Externos Q 67
7.3. Conjuntos de tangentes desde cada punto externo hasta el polígono convexo. 68
7.4. Representación del Orden Rotacional en la Tabla 7.2 70
7.5. Seguimiento de orden rotacional 71
7.6. Seguimiento de Cáliper Rotatorio 72
7.7. Se presenta $p(s) = a + A$; $p(\frac{1}{2}) = a + \frac{1}{2}A$ 73
7.8. S_{jk} está estricatamente contenido en $C_{k-1}(b_j)$ 77
7.9. S_{jk} está contenido en $C_{k-1}(b_j)$ y uno de sus vértices pertenece a $C_{k-1}(b_j)$. . . 77
7.10. S_{jk} está contenido en $C_{k-1}(b_j)$ y sus dos vértices pertenecen a $C_{k-1}(b_j)$. . . 78
7.11. S_{jk} no está contenido en $C_{k-1}(b_j)$ 78
7.12. Algoritmo de identificación de vértices externos e internos a un polígono (Haines, 1989) 80
7.13. Mapa trapezoidal generado por tres segmentos s_1, s_2 y s_3, basado en (de Berg et al., 1997) 83
7.14. Ejemplo de árbol de búsqueda para ubicación de un punto en una subdivisión planar, basado en (de Berg et al., 1997) 84

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

9.1. Experimento 1 Parte 1 - Cierre Convexo y Separación de Conjuntos 100
9.2. Experimento 2 Parte 2 - Obtener $E(\overline{Ge})$ 100
9.3. Experimento 3 Parte 3 - Obtener $E(\overline{Gi})$ 101
9.4. Experimento 4- Paso 4: Obtener $\{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$ 101
9.5. Experimento 5- Algoritmo BR Completo 102
9.6. Comparativa de Comportamiento por Etapa del Algoritmo BR 102
9.7. Experimento 6- Comportamiento en grandes Volúmenes de Datos 103

10.1. Estructura R-tree 106
10.2. Estructura VA-files 107
10.3. Grafo de la Web 108

A.1. carta Gantt - Proyecto de Título 118

C.1. Caso 1: Conjuntos Disjuntos 121
C.2. Caso 2: conjuntos Solapados pero con ausencia de intersecciones 122
C.3. Caso 3: Conjuntos Extendidos en los Cuatro cuadrantes 124
C.4. Caso 4: Conjuntos con Intersecciones en el Exterior 125
C.5. Caso 5: Conjuntos con Intersecciones en el Interior 126
C.6. Clase Grafo 128
C.7. Clase AlgoritmoBR 129
C.8. Clase arista 129
C.9. Clase vertice 130
C.10. Clase tangente 130
C.11. Clase Caliper Rotatorio 131
C.12. Clase dualRojo 131
C.13. Clase rectaDual 131
C.14. Clase regionConvexa 131

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Índice de tablas

3.1. Tabla de Expresiones, Definiciones y Abreviaturas 27

5.1. Requierimientos Funcionales del Sistema 48
5.2. Interfaces Externas de Entrada 49
5.3. Interfaces Externas de Salida 49

7.1. Listado de Tangentes Izquierdas y Tangentes Derechos 69
7.2. Orden Rotacional 70
7.3. Cuadro Comparativo que hace un balance entre las complejidades algorítmicas propuestas por (Cortés et al., 2012) y las complejidades algorítmicas logradas en el Proyecto de Título 88

8.1. Tabla de Resumen de Resultados de Pruebas de Software 91

9.1. Equipos de experimentación 94
9.2. Experimento 1- Paso 1: Cierre Convexo y Separación de Conjuntos 96
9.3. Experimento 2- Paso 2: Obtener $E(\overline{Ge})$ 96
9.4. Experimento 3- Paso 3: Obtener $E(\overline{Gi})$ 97
9.5. Experimento 4- Paso 4: Obtener $\{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$ 97
9.6. Experimento 5- Algoritmo BR Completo 98
9.7. Experimento 6- Prueba de Volumen de Datos 99
9.8. Experimento 6- Prueba de Volumen de Datos 99

11.1. Resumen de Esfuerzo Requerido 110

B.1. Prueba de Software - validación de Formatos de archivos 119

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

B.2. Prueba de Software - validación módulo de generación de Grafo Aleatorio . 120
B.3. Prueba de Software - validación módulo de generación de grafo manual . . 120

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Índice de algoritmos

1. Cierre Convexo - Graham Scan - Versión 1 13
2. Cierre Convexo - Graham Scan - Versión 2 14
3. Vértices Tangentes (Ver (Devadoss y O'Rourke, 2011)) 65
4. Vértices Tangentes Modificado Con vértices de P Ordenados por el eje X . 66
5. Algoritmo de Localización de un punto Interno o Externo a un Polígono . . 79
6. Algoritmo de Localización de un punto en un polígono particionado 81

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Capítulo 1

Introducción

Por muchos años en la geometría computacional, las intersecciones han sido un reto y muchos investigadores han dedicado sus trabajos a ello, la investigación puede considerar a segmentos, planos y cuerpos geométricos en general. Hasta hoy, las intersecciones son una materia de estudio, atendiendo nuevos problemas, variaciones de problemas, casos especiales y optimización de algoritmos.

Un grupo de académicos del Departamento de Matemática Aplicada la Universidad de Sevilla han estudiado esta temática, y han propuesto un algoritmo (Cortés et al., 2012) que obtiene la solución a un problema específico de intersecciones bicromáticas de segmentos.

El problema es reportar el conjunto de segmentos azules que son intersectados por al menos un segmento rojo y, de forma análoga, el conjunto de segmentos rojos que son intersectados por al menos un segmento azul. Además, en los conjuntos de puntos existen intersecciones monocromáticas, las cuales no son parte de la solución del problema, por lo que no son incluidas dentro del conjunto solución.

Por lo tanto, en esta variación el problema además de buscar estas intersecciones bicromáticas, se discrimina la consulta de las intersecciones monocromáticas; de hecho, evitar intersecciones es una de las principales dificultades en muchos problemas asociados a problemas de este tipo.

Este problema es considerado de un nivel de complejidad 3-Sum hard, es decir, es comparable con el problema que consiste en preguntar sí un conjunto de n números enteros, contiene a tres elementos que suman cero (ver (Gajentaan y Overmars, 1995)). Este problema es tomado como modelo para comparación, ya que el problema puede ser ho-

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

mologado a muchos otras problemáticas; más concretamente, muchos de los problemas fundamentales de la Geometría Computacional caen en esta clasificación, pues muchos de los problemas geométricos tienen soluciones que involucran la unión de triángulos. Un problema de clase 3-Sum Hard es de tipo Polinomial, que en el mejor de los casos puede solucionarse en una complejidad $O(n^2)$. En (Cortés et al., 2012), se presenta al Algoritmo BR clasificado como un problema de tipo 3-Sum Hard, y además presenta una demostración teórico-matemática que indica que la solución planteada en dicho algoritmo es de complejidad $O(n^2)$, por lo que se concluye que el algoritmo BR es un procedimiento Optimizado.

Una intersección dada entre un segmento azul y otro segmento rojo es llamada *Intersección Bicromática de segmento*, ya que dos segmentos de distinto color se relacionan (ver Figura 1.1). En cambio, en el caso de una intersección dada entre segmentos de un mismo color se les llama *Intersección Monocromática de segmento* (ver Figura 1.1), la que se distingue para enfatizar la existencia de las Intersecciones Bicromática.

Figura 1.1: Tipos de intersección, (a) Intersección Bicromática y (b) Intersección Monocromática

La motivación de este tipo de investigación tiene sentido en aplicaciones tales como empaquetamiento y cobertura geométrica, modelado de sólidos, detección de colisiones, visibilidad y búsqueda de espacios óptimos, entre otros. Además, las nuevas tecnologías móviles, geolocalización, modelado de sistemas e inteligencia de negocio o de procesos, han demandado nuevas soluciones, por ejemplo, en sistemas de control de procesos, sistemas para el estudio de terrenos o modelado de zonas urbanas, entre otras.

En este proyecto se estudiarán tópicos de esta amplia línea de investigación con el fin desarrollar una implementación en lenguaje Java del algoritmo que determina las intersecciones bicromáticas, el que en adelante será llamado *algoritmo BR*. Una vez desarrollado

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

este software, se iniciará una serie de experimentos donde se expondrá el comportamiento del algoritmo frente a diferentes casos.

En el capítulo 2 se tratarán temas y conceptos fundamentales de la teoría de la geometría computacional. En el capítulo 3, se expone la definición del proyecto. El capítulo 4 enfatiza la descripción del algoritmo a trabajar en el proyecto. El capítulo 5 realiza la especificación de requerimientos del software. El capítulo 6 presenta el diseño para el proyecto, incluyendo el diseño de clases, diseño de archivos de entrada y el diseño de interfaz. Por lo que luego, en el capítulo 7 se realiza toda la construcción del software. En el capítulo 9 se experimenta varios casos en la implementación del algoritmo. En el capítulo 10 se argumentan las líneas de trabajos futuros sobre este tema. El capítulo 11 detalla un resumen de esfuerzo realizado para este proyecto. En las conclusiones se muestran los resultados obtenidos comparados con los objetivos del proyecto. En el capítulo 12 se muestra un listado de Referencias Bibliográficas utilizadas. Luego, se presenta información complementaria en los anexos A, B, **??**, C y D para complementar lo expuesto en el proyecto.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Capítulo 2

Conceptos Fundamentales desde la Teoría

2.1. Teoría de Grafos

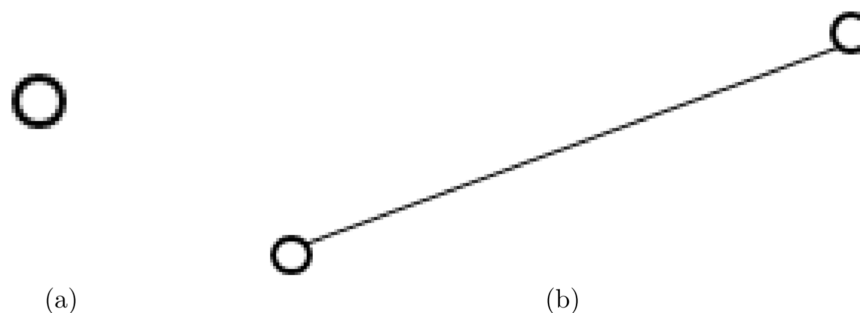
Dentro de los campos de estudio de la geometría computacional se encuentra la teoría de grafos, que estudia las propiedades y aplicaciones en problemas que puedan representarse en forma de grafo. Además, es materia de estudio de las Matemáticas Discretas, y Matemáticas Aplicadas, en áreas de análisis de combinatoria, álgebra abstracta, probabilidad, geometría de polígonos, aritmética y topología, entre otras áreas de estudio.

La teoría de grafos se remonta al siglo XVIII con el problema de los puentes de Königsberg (Kalingrado), el cual consistía en encontrar un camino que recorriera los siete puentes del río Pregel pasando solo una vez por cada puente. Muchos otros investigadores de la época comenzaron a utilizar este tipo de representación para resolver problemáticas de otras áreas. Mientras que el término de *grafo* (derivada de la expresión *"graphic notation"* o *notación gráfica*) fue utilizado por primera vez por el matemático James Sylvester en 1878 y por el químico Edward Frankland (Frankland, 1884), posteriormente adoptado por Alexander Crum Brown en 1884 , donde se hacía referencia a la representación gráfica de los enlaces entre los átomos de una molécula. El primer libro sobre la teoría de grafos fue escrito por Dénes König publicado en 1936 (König, 1936).

Un grafo es una estructura que consta de un conjunto de vértices, nodos o puntos, los que están conectados mediante aristas, las que forman líneas, segmentos o lados en el

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

grafo. Con estos elementos un grafo es capaz de representar todo tipo de formas y permite obtener todo tipo de propiedades sobre las formaciones obtenidas.



(a) (b)

Figura 2.1: (a) Vértice y (b) Arista

Existen diferentes tipos de grafos para representar distintas situaciones, los tipos son los siguientes: Un grafo simple es aquel que acepta una sola arista uniendo dos vértices cualesquiera (ver Figura 2.1b), es decir, que una arista es la única que puede unir dos vértices específicos. Un multigrafo es un grafo que acepta más de una arista entre vértices, entonces los grafos simples son subclases de un multigrafo (ver Figura 2.2). Un grafo dirigido se obtiene añadiendo una orientación a los segmentos que unen los vértices, obligando la dirección entre un vértice al otro. Análogamente, un grafo que no contenga ningún segmento o arista dirigida, es un grafo no dirigido (ver Figura 2.2). Un grafo etiquetado se caracteriza por añadir un peso de importancia a las aristas o a los vértices. Un Grafo aleatorio tiene aristas asociadas por probabilidad. Un hipergrafo es un grafo donde las aristas tienen más de dos extremos, es decir, las aristas son incidentes a tres o más vértices. Un grafo completo es aquel que todos los vértices están unidos por aristas.

La teoría de grafos ha logrado representar diferentes tipos de problemáticas, tales como, el diseño de circuitos electrónicos, dibujo computacional, planificación de transportes, optimización de tráfico, administración de proyectos a través de grafos y esquemas. En las ciencias sociales; redes sociales, en control de producción y distribución espacial de equipos. En genética, biología, entre muchas otras aplicaciones.

Tal como se puede apreciar en la Figura 2.3, es posible representar problemas de

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

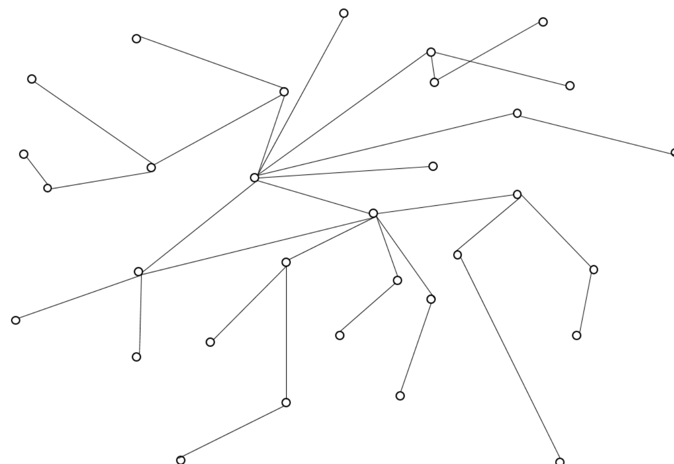
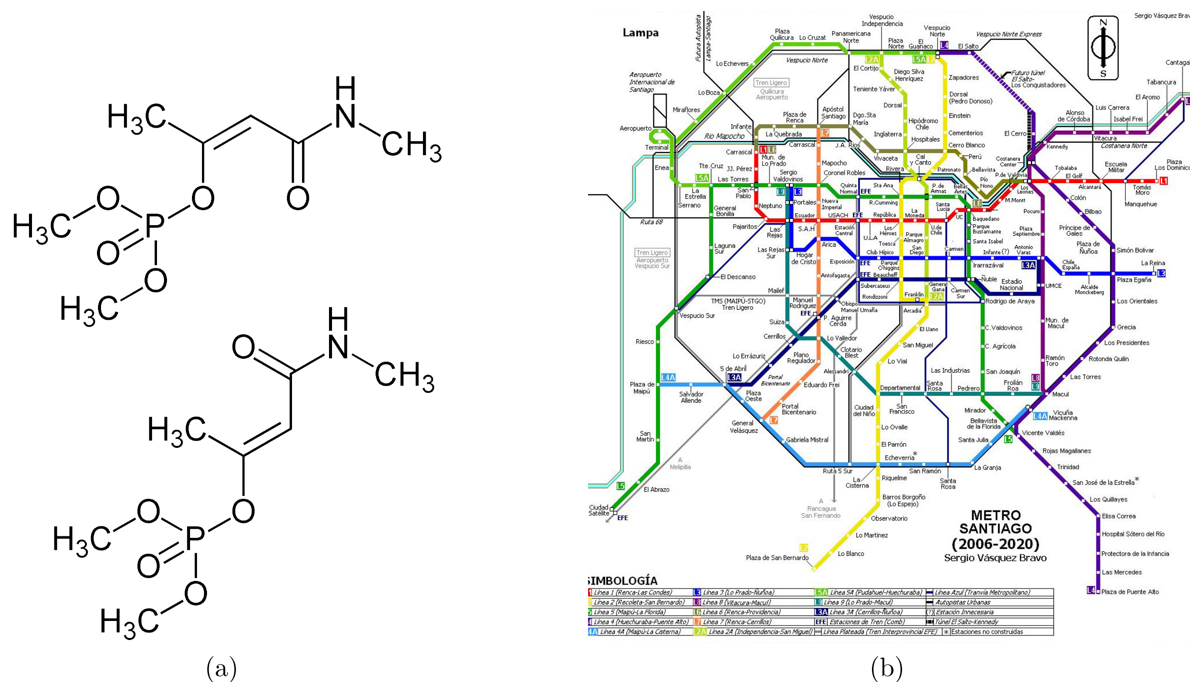


Figura 2.2: Ejemplo de un Grafo no Dirigido



(a) (b)

Figura 2.3: Ejemplos de aplicaciones en Teoría de Grafos

diferentes áreas de estudio a traves de la Teoría de Grafos, ya que se pueden definir elementos que puedan caracterizar vértices o nodos, y aristas o enlaces.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Por ejemplo, la figura 2.3a es la representación de una Formula Estructural típica del área de la Química; por otro lado, la figura 2.3b representa el trazado de los recorridos de las líneas de Metro de Santiago. Ambas figuras contienen elementos distintivos que ayudan a la representación en forma de grafo, puntos y líneas, por lo que resulta natural utilizar estos puntos como vértices y las líneas que las unen como aristas. De esta forma, un Grafo puede ayudar a representar estos escenarios de la vida cotidiana para lograr una solución.

2.2. Teoría de Geometría Computacional

2.2.1. Definición

La geometría Computacional es una de las ramas de las ciencias de la Computación dedicada al estudio de temáticas y problemas que puedan ser expresados en términos geométricos con el fin de desarrollar e implementar soluciones en términos computacionales, es decir, a través de métodos, técnicas, procedimiento y algoritmos. A través del estudio y diseño de algoritmo eficientes para resolver problemas de tipo geométrico.

2.2.2. Orígenes

Muchos autores tales como Mendoza (Mendoza, 1997) y Mark de Berg (de Berg et al., 1997) entre otros, concuerdan que el inicio de este campo del conocimiento estuvo dado de la mano del PhD. Michael Ian Shamos en su tesis doctoral que otorgó el nombre del área (Shamos, 1978). A través de este detallado documento se sentaron las bases del estudio de esta rama, ya que comprendía un variado número de procedimientos, métodos, técnicas y algoritmos que permiten el procesamiento en un entorno computacional, esto quiere decir que está descrito de tal forma que se puede expresar en un lenguaje computacional e implementar una solución.

Posteriormente, el campo de estudio de la geometría computacional se concreta con el libro publicado por el investigador Franco P. Preparata y Michael I. Shamos titulado "Computational Geometry – An Introduction" (Preparata y Shamos, 1985).

Sin embargo, el propio Michael Shamos reconoce que la resolución de problemas representados en términos geométricos llevan varios milenios de práctica, un ejemplo claro

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

serían los escritos egipcios y griegos sobre estas materias en formas básicas de resolución y con técnicas rudimentarias.

La geometría computacional es considerada una ciencia abstracta, debido a que no se consideran aspectos de la tecnología de los equipos computacionales usados, sino que se limita a construir soluciones en términos que se puedan solucionar con un procedimiento, algoritmo, técnica o método determinado. Para obtener estas soluciones, es necesario recurrir a campos de estudio como la geometría clásica, la topología, el álgebra lineal y la teoría de conjuntos.

Hoy en día, existe mucha documentación en diferentes líneas investigativas, literatura académica abundante como puede verse en (Bærentzen et al., 2012; Devadoss y O'Rourke, 2011; Held, 1991).

2.2.3. Aplicaciones de la Geometría Computacional

La Geometría Computacional se ha aplicado en varias áreas del conocimiento, tales como: la robótica, el reconocimiento de voz y patrones de sonido, diseño gráfico en CAD/CAM, sistemas que reciben información de un dispositivo GPS (GPS dependientes) o de información geográfica, aplicaciones como el modelado molecular, reconocimiento de patrones, bases de datos multidimensionales, sólo por mencionar algunas.

Por otro lado, cuando se es posible observar un determinado problema en forma gráfica, entonces es posible que el problema pueda tener solución aplicando las técnicas, algoritmos y métodos de la geometría computacional.

2.3. Cierre Convexo

Como punto especial, el Cierre Convexo es el tema más presente en la geometría computacional. Representa algo así como un caso de éxito en la geometría computacional. Uno de los primeros documentos identificados en el área que se refiere al cálculo del cierre convexo, y corresponde al trabajo publicado por Ronald Lewis Graham en el año 1972 (Graham, 1972). Desde entonces, ha habido una sorprendente variedad de investigación referente al tema, que conduce a la creación de varios algoritmos de diversas complejidades.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

2.3.1. Definición

Se llama convexo a un conjunto S del plano sí y sólo sí para cualquier par de puntos pertenecientes a S, el segmento cerrado que los une queda dentro de S (O'Rourke, 1998; Shamos, 1978) (Ver figura 2.4) El cierre convexo de un conjunto S es el conjunto más pequeño que contiene a S (O'Rourke, 1998; Shamos, 1978).

Intuitivamente, el concepto de cierre convexo es fácil de entender, el problema es lograr una idea definitiva de cómo expresarlo computacionalmente. Se sabe que la entrada es un conjunto de puntos no ordenados, pero no se sabe con exactitud cuál es la salida (en términos de estructuras de datos). Una forma prácticamente natural de representar el cierre convexo es a través de una lista de vértices ordenados en sentido anti horario, comenzando por un vértice arbitrario. Con lo anterior, el problema a resolver es:

"Dado un conjunto de puntos en el plano $P = \{p_1, p_2, p_3, ..., p_n\}$, obtener la lista de puntos de P que son vértices del polígono convexo que define la cerradura convexa de P, ordenados en sentido anti horario"

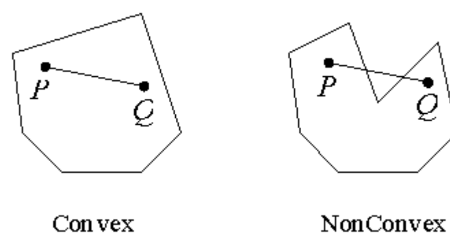


Figura 2.4: Ejemplos de polígono convexo y no convexo

Tomando como ejemplo la figura 2.5, la entrada es la lista no ordenada de puntos $P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8$ y la salida corresponde a la lista ordenada P_8, P_5, P_1, P_2, P_6.

Un algoritmo para calcular el cierre convexo de una nube de puntos en el plano, que por lo demás es bastante intuitivo, consiste en eliminar los puntos interiores del cierre. Lo anterior se basa en que los puntos del cierre son extremos (encierran a todos los demás) por lo que para obtener el cierre convexo se hace necesario encontrar estos puntos extremos. Teniendo en cuenta que un punto es interior sí y sólo sí pertenece a un triángulo determinado por otros tres puntos, se puede eliminar los puntos interiores seleccionando

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

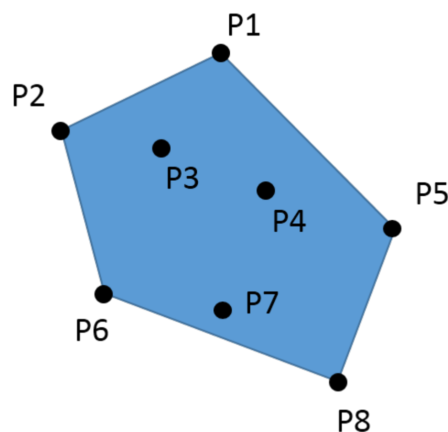


Figura 2.5: Puntos en el plano con su cierre convexo

los puntos del conjunto de tres en tres de todas las formas posibles y eliminando los puntos que estén incluidos en alguno de dichos triángulos. De esta manera solo quedarán los puntos extremos.

El algoritmo anterior es de complejidad $O(n^4)$, ya que se deben consultar todas las combinaciones posibles de triángulos formados por los vértices de la nube de puntos, lo que en terminos de combinatoria resulta en $\binom{n}{3}$ combinacions de puntos, multiplicados por los $(n-3)$ puntos restantes.En términos de programación, el algoritmo se traduce en tres ciclos for anidados para encontrar cada triángulo, y por cada triángulo se debe consultar por todos los puntos restantes (otro ciclo for anidado).

Una mejora se realizó posteriormente basándose en una propiedad que poseen las aristas del cierre convexo: la recta que determinan deja al resto de los puntos en un mismo semiplano de los que esta recta define. Empleando esta propiedad, un algoritmo para hallar el cierre convexo consiste en escoger los puntos de dos en dos de todas las formas posibles y, para cada elección, comprobar si el resto de los puntos están en un mismo semiplano. Haciendo esto, se encuentran las aristas del cierre convexo.

Según la definición combinatoria de coeficiente binomial, la cantidad de combinaciones posibles de pares de puntos corresponde a $\binom{n}{2}$. Para cada uno de estos pares es necesario realizar $(n-2)$ comparaciones, por lo que el resultado es que se debe realizar al menos un número de operaciones de orden $O(n^3)$, lo que en código se vería reflejado en tres ciclos

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

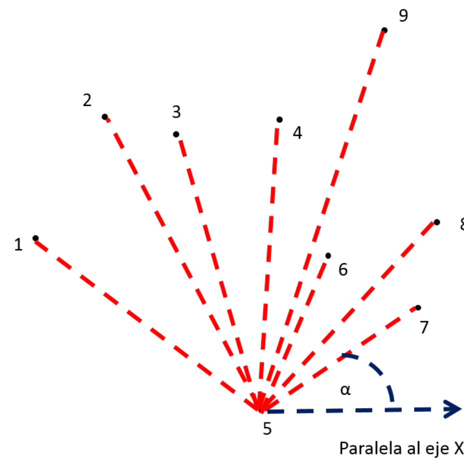


Figura 2.6: Ordenamiento angular de los puntos de la nube respecto a un punto específico

for anidados.

Luego de un par de intentos en los que se logró algoritmos de $O(n^2)$, Ronald L. Graham creó un nuevo algoritmo (Graham, 1972), por encargo de Bell Laboratoires, quienes usaban una aplicación que requería calcular el cierre convexo para 10,000 puntos, lo cual era demasiado para un algoritmo de $O(n^2)$.

Este nuevo algoritmo, llamado Graham Scan, fue uno de los primeros en ser publicados. Su funcionamiento se basa en generar una lista ordenada con los puntos de la nube. Este ordenamiento consiste en ordenar angularmente (de menor a mayor ángulo, en sentido antihorario) todos los puntos de la nube con respecto al eje horizontal de un punto interior escogido aleatoriamente. El algoritmo parte buscando este punto interior, nombrándolo como p_0 y haciéndolo el primero de la lista ordenada. Luego se procede a ordenar angularmente los demás puntos respecto a p_0 (Ver figura 2.6).

Una vez que los puntos están ordenados en la lista, se toman los dos primeros puntos y se guardan en una pila, la que finalmente contendrá los puntos del cierre convexo ordenados en sentido antihorario. A continuación, se realiza una serie de iteraciones, una por cada punto restante, en las cuales se toma el siguiente punto de la lista ordenada (siguiendo el orden dado por ésta) y se verifica la condición de giro entre este punto y los dos puntos en el tope de la pila.

La condición de giro indica de qué lado de la recta, formada por los dos puntos anteriores, se encuentra este nuevo punto, vale decir, izquierda, derecha o en línea. En otras

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

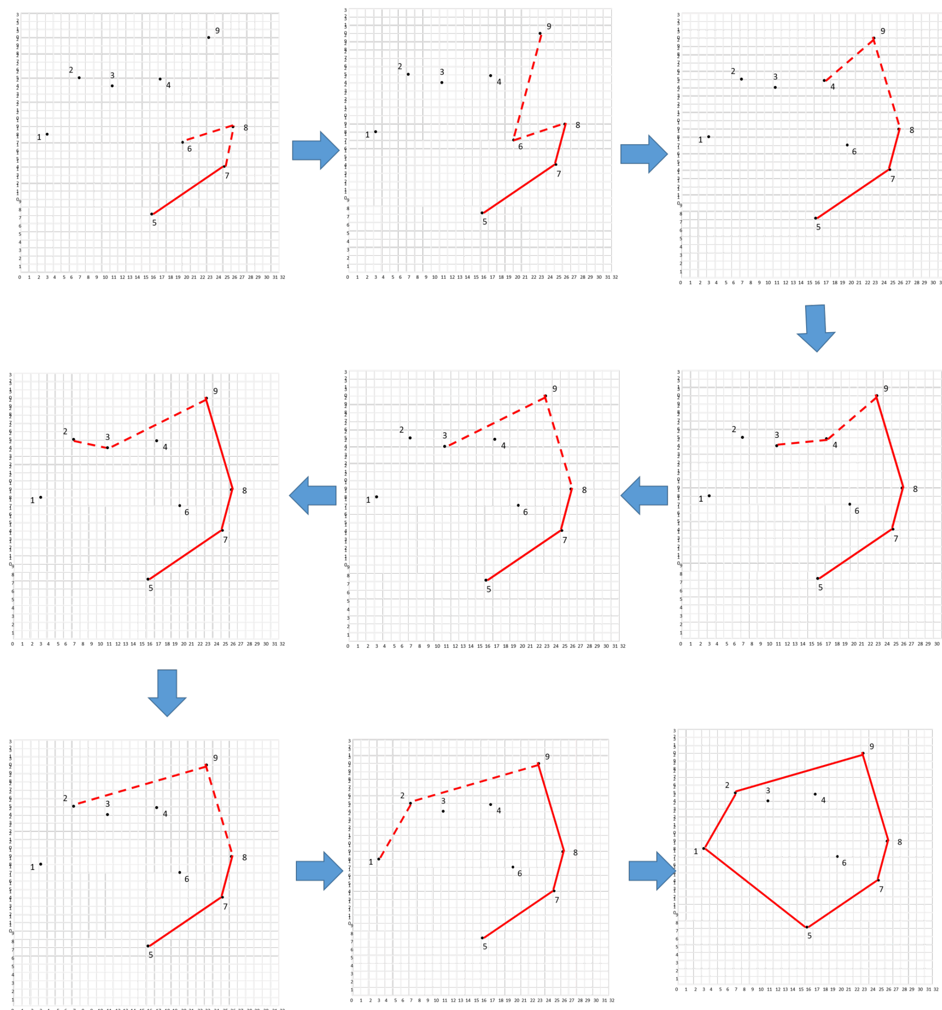


Figura 2.7: Secuencia de búsqueda del cierre convexo por Graham Scan

palabras, si recorriéramos estos tres puntos, con los dos primeros trazamos un ”camino” recto y para llegar al tercer punto será necesario girar a uno de los dos lados (si es que no se debe seguir derecho para el caso de puntos alineados).

Hay diferentes formas de saber hacia dónde es el giro, una de ellas, la señalada en (Shamos, 1978) y la que se utiliza en este proyecto, es realizando un cálculo sobre los tres puntos $i = (x_1, y_1)$, $j = (x_2, y_2)$, $k = (x_3, y_3)$, el cual consiste en encontrar el producto vectorial entre los vectores definidos por las coordenadas de $\overrightarrow{ij}$ e $\overrightarrow{ik}$, dando como resultado la ecuación $(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$, donde si el resultado es positivo, el giro

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Algoritmo 1 Cierre Convexo - Graham Scan - Versión 1

Entrada: Conjunto P de puntos no ordenados.
Salida: Conjunto S de puntos del cierre convexo.

1: Encontrar un punto interior x; nombrarlo como p_0
2: Ordenar angularmente los demás puntos respecto a x; nombrarlos $p_1, ..., p_{n-1}$
3: Pila $S = (p_2, p_1) = (p_t, p_{t-1})$; t indexa la cima.
4: $i = 3$
5: **mientras** $i < n$ **hacer**
6:     **si** p_i está a la izquierda de (p_{t-1}, p_t) **entonces**
7:         Push (p_i, S)
8:         $i = i + 1$
9:     **si no**
10:         Pop(S)
11:     **fin si**
12: **fin mientras**

es hacia la izquierda; si el resultado es negativo, es un giro a la derecha; y si el resultado es cero, los puntos se encuentran alineados (dos puntos iguales, también se consideran alineados).

Si se toman los puntos del cierre convexo en sentido antihorario, se puede observar claramente que todos sus puntos, al tomarlos de tres en tres de forma ordenada (cada punto con sus dos anteriores), forman giros hacia la izquierda, por lo cual solo deben incluirse en la pila los puntos que, al unirse con sus dos antecesores, formen un giro a la izquierda.

Siguiendo con el algoritmo de Graham, una vez que se verifica la condición de giro explicada en los párrafos anteriores, si el giro es a la izquierda, el punto es apilado; en caso contrario, se saca de la pila el punto que está en el tope y se prueba la condición de giro con los nuevos tres puntos dados (bajo las mismas condiciones anteriores: primer punto de la lista con los dos topes de la pila). El algoritmo finaliza cuando recorre todos los puntos de la lista ordenada. Para más detalles, ver algoritmo 1.

En una segunda versión del algoritmo, propuesta en (O'Rourke, 1998), se hace una variación: el punto de referencia, para ordenar la lista de puntos de la nube, ya no corresponde a un punto interior al cierre, sino que corresponde a un punto del mismo cierre, el cual es encontrado buscando el punto de más abajo y más a la derecha de entre los puntos de la nube. Luego se procede de igual forma, ordenando angularmente los demás puntos con

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Algoritmo 2 Cierre Convexo - Graham Scan - Versión 2

Entrada: Conjunto P de puntos no ordenados.
Salida: Conjunto S de puntos del cierre convexo.

```
1:  Encontrar el punto de más abajo y más a la derecha; nombrarlo como p_0
2:  Ordenar angularmente los demás puntos respecto a p_0; nombrarlos p_1, ..., p_{n-1}
3:  Pila S = (p_1, p_0) = (p_t, p_{t-1}); t indexa la cima.
4:  i = 2
5:  mientras i < n hacer
6:     si p_i está estrictamente a la izquierda de (p_{t-1}, p_t) entonces
7:        Push (p_i, S)
8:        i = i + 1
9:     si no
10:       Pop(S)
11:    fin si
12: fin mientras
```

respecto a este primero y realizando las operaciones con la pila (descritas anteriormente). Ver algoritmo 2 y figura 2.7 con el seguimiento del algoritmo.

2.4. Cáliper Rotatorio

El Cáliper Rotatorio es un procedimiento que puede apoyar la resolución de un problema que se puede representar a través de uno o más polígonos convexos. Los beneficios de usar este tipo de técnicas es la reducción de la complejidad en tiempo de ejecución y espacio de memoria. La utilización de éste, puede proporcionar soluciones a aplicaciones referentes a visibilidad, colisiones, evitación, ajuste de rango, separabilidad lineal y el cálculo de la distancia Grenander, según explica Toussaint (Toussaint, 1983).

2.4.1. Historia y Evolución

Los primeros vestigios de este procedimiento fueron registrados en el algoritmo propuesto por Shamos en su tesis doctoral (Shamos, 1978) sobre la solución al diámetro de un polígono, donde el autor propone realizar la búsqueda de un par de vértices del polígono que formen la distancia más grande de entre los vértices que componen el cierre convexo, por lo que elimina gran parte de las consultas inútiles. Sin embargo, Shamos limitó el uso

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

de este procedimiento y no comprendió el potencial al aplicarlo en otros algoritmos.

Por otro lado, Godfried Toussaint en (Toussaint, 1983) observa el potencial de este procedimiento, recopilando varias soluciones basadas en lo anterior y reconociendo su utilidad no tan solo en un polígono convexo, sino que en más de un polígono convexo y más de un cáliper rotatorio en un polígono. Además, este autor fue quien dio nombre a este procedimiento bautizándolo como *Cáliper Rotatorio*, haciendo alusión al *Cáliper de Vernier*.

Posteriormente se han desarrollado más algoritmos basados en este procedimiento, descubriendo nuevas propiedades, lemas y pruebas.

2.4.2. Definiciones

"Sea $P = \{p_1, p_2, ..., p_n\}$ un polígono convexo con n vértices en su forma estándar (por ejemplo, según las coordenadas cartesianas ordenadas en sentido contrarreloj) y asumiendo que no contienen tres puntos consecutivos co-lineales. Una Línea L es una línea de soporte de P si el interior de P se encuentra completamente a un lado de L." (Toussaint, 1983).

A partir de esta definición de una línea de soporte, se puede expandir el concepto a una segunda línea de soprote simultánea y paralela, estas líneas se encuentran en vértices opuestos del polígono convexo, llamados puntos antipodales. Más formalmente: *"Un par p_i, p_j es un par de vértices antipodales si admiten líneas paralelas de soporte"*

Tomando estas dos formalidades anteriores se puede construir las bases del concepto de Cáliper, como se puede ver en la Figura 2.8, y añadiendo la propiedad dinámica de este procedimiento a través de la siguiente aseveración. *"Para encontrar el siguiente par de vértices antipodales es necesario rotar un par de Cálipers dinámicamente ajustables una vez sobre el polígono"*. Consolidando el comportamiento del cáliper rotatorio respecto del polígono convexo existente, reflejándose en la variación de la pendiente y el movimiento en los puntos antipodales (ver Figura 2.9).

En la Figura 2.9, se muestra cómo las líneas paralelas de soporte rotan hacía la siguiente posición en el polígono. En la posición inicial, el par antipodal es el nodo A y en nodo D, para pasar a la siguiente posición, donde el par anntipodal cambia al nodo A con el nodo E. Este par cambiará constantemente mientras el Cáliper Rotatorio siga girando.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

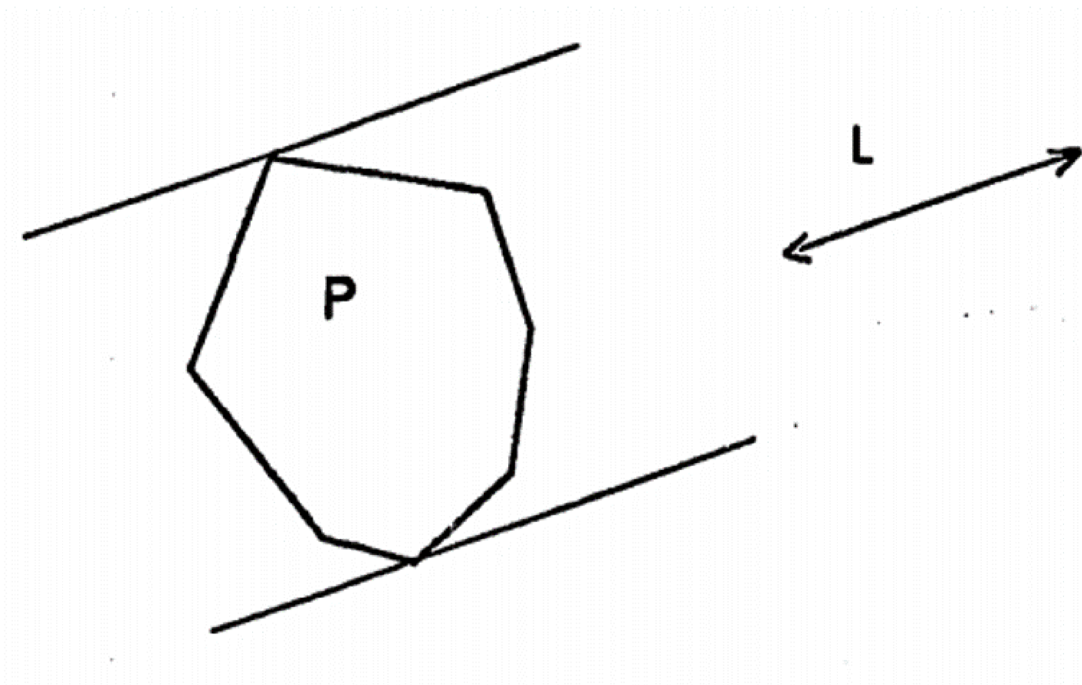


Figura 2.8: Cáliper Rotatorio (Shamos, 1978)

Ajustando todo esto se pueden comprender las siguientes definiciones:

DEFINICIÓN 1: *"Una Línea de Soporte de un conjunto de puntos P es aquella que pasa por un punto de conjunto y deja a todo P en uno de los dos semiplanos que delimita".*

DEFINICIÓN 2: *"Las Líneas Paralelas de Soporte son dos rectas de soporte paralelas".*

DEFINICIÓN 3: *"Dos puntos de P se dicen antipodales si por ellos pasan líneas paralelas de soporte".*

2.4.3. Algoritmo

La implementación de un Cáliper Rotatorio requiere de la aplicación de ciertos lemas que controlan el movimiento de los puntos antipodales. Básicamente, es el control sobre la composición del par Antipodal que soporta al cáliper rotatorio.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

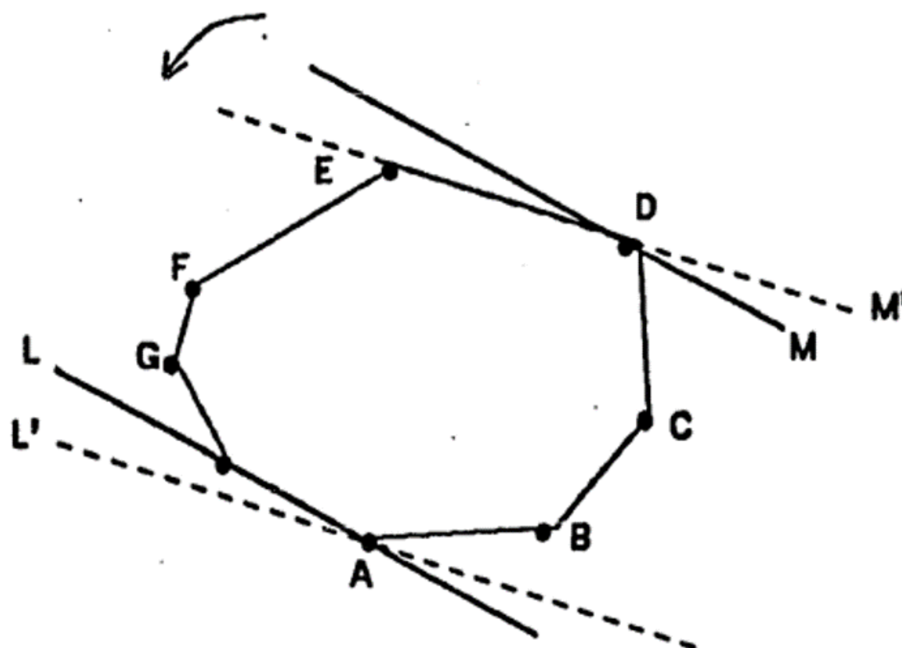


Figura 2.9: Cáliper Rotatorio (Shamos, 1978)

LEMA 1: *Sea $u_{k-1}u_k$ una arista de un polígono convexo P, tal que se recorrerá los vértices de P en el orden contrarreloj, comenzando desde el vértice u_k. Sea u_i el primer vértice más lejano de $u_{k-1}u_k$ en el recorrido, entonces ningún vértice entre u_k y u_i forma un par antipodal con u_k.*

LEMA 2: *Sea $u_{k-1}u_k$ una arista de un polígono convexo P, tal que se recorrerá los vértices de P en el orden reloj, comenzando desde el vértice u_k. Sea u_r el último vértice más lejano de $u_{k-1}u_k$ en el recorrido, entonces ningún vértice entre u_r y u_{k-1} (en orden contrarreloj) forma un par antipodal con u_{k-1}.*

Estos lemas indican que un vértice u_k sólo puede ser par antipodal de otro vértice que se encuentre en el otro extremo opuesto más lejano, sea en sentido horario y antihorario, y siguiendo la pendiente del Cáliper Rotatorio.

Considerando esto, se presenta un algoritmo descriptivo de la implementación de un

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

cáliper rotatorio. A modo de un instructivo general sobre cualquier problema, cabe destacar que la implementación de un cáliper siempre estará condicionada a la necesidad de la aplicación en particular:

1 Se requiere de un polígono convexo (o varios dependiendo de la aplicación) de entrada para el algoritmo.

2 Para iniciar el algoritmo se precisa una dirección, puede ser paralela al eje x o con respecto al eje y.

3 Una vez elegida, se seleccionan dos vértices antipodales p_i y p_j (pueden ser encontrados en tiempo $O(n)$).

4 Para generar el próximo par antipodal se consideran los ángulos formados por las líneas de soporte a p_i y p_j y las aristas p_ip_{i+1} y p_jp_{j+1}, respectivamente (Ver figura 2.10). Si el ángulo $\theta_j < \theta_i$, entonces se rotan las líneas de soporte por el ángulo θ_j, y p_{j+1}, p_i viene a ser el próximo par antipodal. O bien, se decide respecto al objetivo de la aplicación de la implementación del Cáliper Rotatorio.

5 Este proceso se continúa repitiendo hasta completar el círculo en la posición inicial. En el caso que $\theta_j = \theta_i$ se generan tres nuevos pares antipodales. O bien, al cumplir con el objetivo de la aplicación de la implementación del Cáliper Rotatorio.

6 La salida del algoritmo es la información requerida para cumplir el objetivo de la aplicación de la implementación del Caliper Rotatorio.

El algoritmo de Cáliper Rotatorio tiene esta estructura, luego identifica su posición en el polígono, decide su giro respecto del interés del problema (Orden Rotacional) y responde a la necesidad planteada en el problema específico al girar alrededor del polígono (Ver Figura 2.10).

Es importante mencionar que el concepto de Cáliper Rotatorio presenta una alta flexibilidad a la aplicación en algoritmos, ya que un Cáliper Rotatorio se ha usado sobre un polígono convexo o sobre más de dos polígonos convexos, comparte líneas paralelas de soporte entre polígonos convexos, el sentido del giro puede ser tanto antihorario, como horario, entre más facilidades para su implementación; más concretamente, la flexibilidad se puede ver en la condición de rotación y en la misma rotación del cáliper Rotatorio respecto del o los polígonos convexos involucrados. Si bien, la versión clásica muestra la forma en

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

que un cáliper puede girar guiado por los ángulos formados por los puntos antipodales, existen variadas formas que logran ser igualmente de efectivas, pero en los casos partículas de las soluciones a los problemas planteados.

Cortés et al (Cortés et al., 2012) menciona que al obtener un Orden Rotacional del polígono, el cáliper rotatorio puede seguir este Orden y bajar su complejidad de tiempo a $O(|puntosexternos| + nlogn)$, aunque este procedimiento sigue siendo funcional para la aplicación en particular, sin embargo puede ser un criterio de rotación que sea útil para otras aplicaciones.

Este Orden Rotacional, al contrario de las aplicaciones anteriomente propuestas, depende netamente de factores externos al polígono convexo P, ya que el cáliper girará guiado por cada punto externo a P (vease en apartado 4).

2.5. Arreglo Dual de Líneas

Un arreglo dual de líneas surge al trabajar con arreglos de líneas (*Arrangements of lines*, en inglés) en conjunto con el concepto de *Dualidad*. Para comprender un arreglo dual de líneas, entonces, es necesario comprender estos conceptos recién mencionados.

2.5.1. Arreglo de Líneas

Un arreglo de líneas es una coleción de líneas *organizadas* en el plano, de tal manera que inducen una partición de este en regiones convexas (caras), segmentos o aristas y vértices. Forman la tercera estructura importante usada en geometría computacional, tan importante como el cierre convexo y los diagramas de Voronoi (O'Rourke, 1998).

La figura 2.11 muestra un arreglo de 10 líneas, el cual, en base al teorema 2.5.1.1 que se muestra más adelante, tendría $V = 45$ vértices, $E = 100$ aristas, y $F = 56$ caras. Cabe destacar que, dado que el plano es infinito y la ventana de la figura es limitada, muchas intersecciones entre líneas no son visibles, por lo que no todos los vértices, aristas y caras son visibles en la figura.

Un arreglo de líneas es llamado simple si cada par de líneas se encuentra en exactamente un punto (lo que implica la no existencia de líneas paralelas) y no hay tres líneas que se encuentren en un mismo punto. Los arreglos ”no simples” son ”degenerados” en algún

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

sentido y, a menudo, los teoremas y algoritmos son más fáciles con arreglos simples.

Los arreglos simples de n líneas tienen exactamente el mismo número de vértices, aristas y caras; lo que se expresa en el siguiente teorema.

Teorema 2.5.1.1 *En un arreglo simple de n líneas, el número de vértices, aristas y caras es $V = \binom{n}{2}$, $E = n^2$ y $F = \binom{n}{2} + n + 1$, respectivamente, y los arreglos no simples no exceden esas cantidades (O'Rourke, 1998).*

Una explicación visual de porque los arreglos no simples no exceden las cantidades indicadas en el teorema anterior puede verse en la figura 2.12, donde al lado izquierdo se muestra un arreglo no simple para dos casos distintos (en (a) se muestran más de dos líneas que se cruzan en un punto; en (b), dos líneas paralelas), los cuales son perturbados, lográndose solo aumentar el número de vértices, aristas y caras. Para una explicación detallada del teorema, ver (O'Rourke, 1998).

2.5.2. Dualidad

Dentro de las variadas aplicaciones que tienen los arreglos de líneas, como por ejemplo la planifiiación de caminos mediante grafos de visibilidad, la clave está en un importante concepto conocido como *Dualidad*. La idea básica es que, debido a que pueden ser especificadas por dos números, las líneas pueden ser asociadas con el *punto* cuyas coordenadas son esos dos números. Por ejemplo, una línea especificada por $y = mx + b$ puede ser asociada con el punto (m, b). A este proceso de cambiar entre líneas y puntos aplicando dualidad se le conoce como mapeo de dualidad.

Debido a que el mapeo desde líneas a puntos es determinado, puede revertirse; así, cualquier punto en el plano puede ser visto como representando una línea si sus coordenadas son interpretadas como pendiente e intersección, por poner un ejemplo. Ambos mapeos, en conjunto, determinan una dualidad entre puntos y líneas: cada línea es asociada con un único punto, y cada punto con una única línea. Existen muchos mapeos de dualidad punto-línea posibles, dependiendo de las convenciones de la representación de una línea. Cada mapeo tiene sus ventajas y desventajas en contextos particulares. Como ejemplo se tiene el ya mencionado $L : y = mx + b \Leftrightarrow p : (m, b)$, el cual tiene la ventaja de asociarse a la familiaridad que se tiene con los conceptos de pendiente e intersección; otro ejemplo

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

es el mapeo $L : ax + by = 1 \Leftrightarrow p : (a, b)$, que define lo que es conocido como dualidad polar y que tiene propiedades geométricas agradables. Un último mapeo muy utilizado en (O'Rourke, 1998) es $L : y = 2ax - b \Leftrightarrow p : (a, b)$. Se usa el símbolo $\mathcal{D}$ para representar un mapeo; así $\mathcal{D}(L) = p$ y $\mathcal{D}(p) = L$.

Ya con los conceptos de *Arreglos de Líneas* y *Dualidad* entendidos en forma general, se puede decir que un arreglo dual de líneas es un arreglo de líneas obtenido de aplicar un mapeo de dualidad a un conjunto de puntos.

Existe un algoritmo incremental para la creación de un arreglo de líneas, el cual utiliza estructuras de datos complejas (como la DCEL) y que tiene complejidad $O(n^2)$ tiempo. Este algoritmo no será utilizado en la implementación del algoritmo BR debido a que solo interesa saber el orden rotacional obtenido a partir de las intersecciones de las rectas duales azules con las rojas (Ver algoritmo 4.0.1).

2.6. Conclusión de los Conceptos Fundamentales de la Teoría de la Geometría Computacional

Los temas planteados anteriormente son parte importante del desarrollo del proyecto, ya que cada uno de ellos provee de técnicas y métodos posibles de aplicar en la resolución de secciones bien determinadas del algoritmo BR en construcción.

Cada una de las temáticas han proporcionado conocimiento sobre su funcionalidad en particular, sus propiedades, su comportamiento y aplicación. Además, de este conocimiento se puede apreciar el trabajo de años de estudio resumidos en textos, charlas y herramientas; Con ello, también se pueden vislumbrar perspectivas para el futuro en cuento a nuevas investigaciones, nuevas herramientas y nuevas técnicas, todo a raíz del conocimiento ya recopilado.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

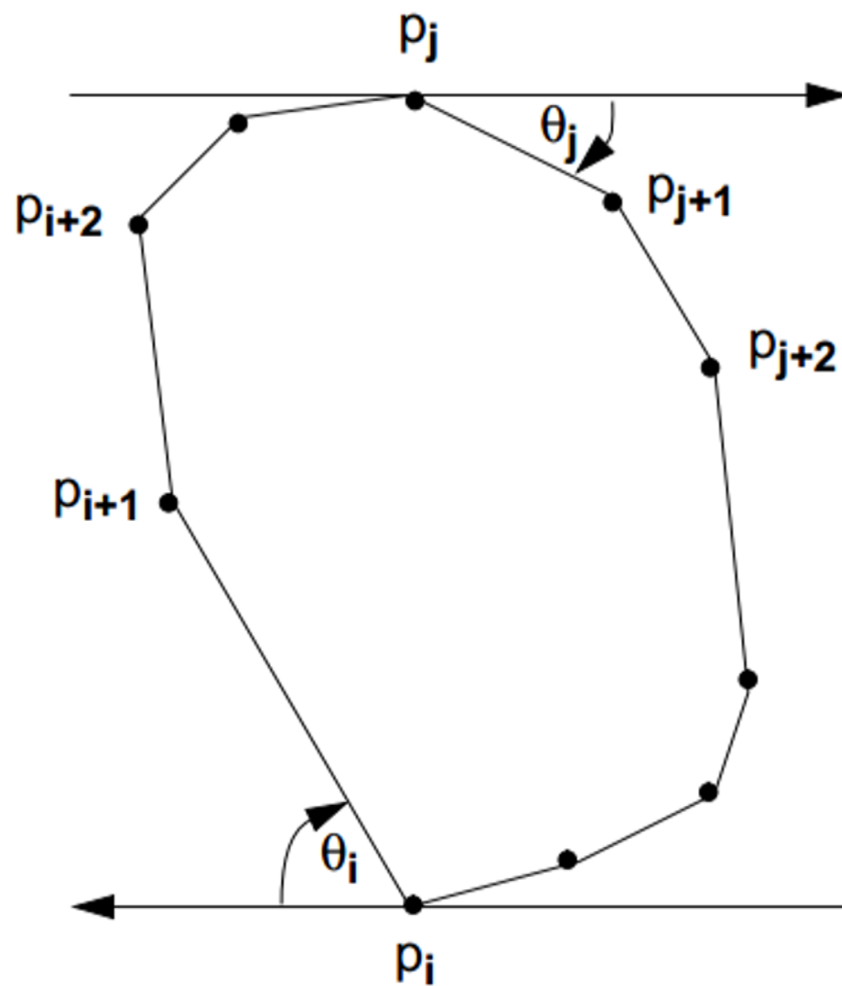


Figura 2.10: Forma de giro de un Cáliper Rotatorio (Toussaint, 1983)

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

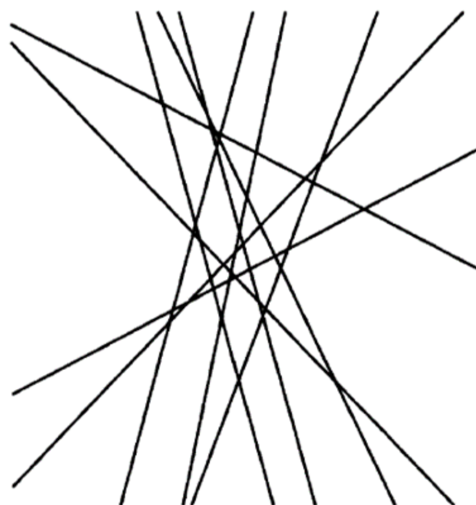


Figura 2.11: Arreglo de líneas de 10 líneas. $V = 45$, $E = 100$ y $F = 56$.

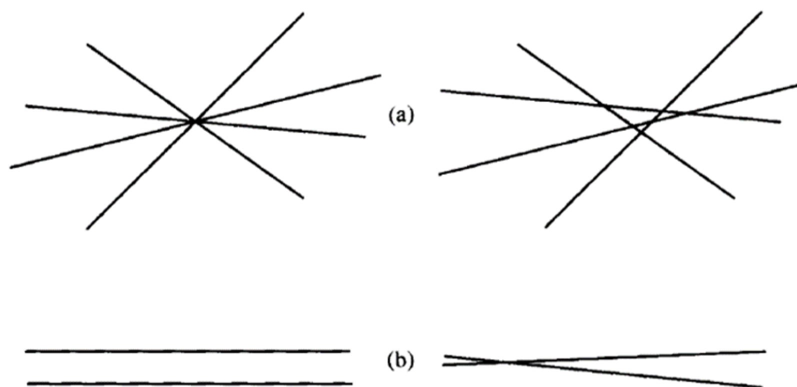


Figura 2.12: Perturbación de arreglos de líneas no simples.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Capítulo 3

Definición del Proyecto

3.1. Objetivos del proyecto

3.1.1. Objetivos Generales:

Sea el siguiente problema Geométrico: dados dos conjuntos de puntos B (azules) y R (rojos) en el plano, que definen dos conjuntos de segmentos, azules y rojos, reportar los segmentos de cada color que son intersecados por los segmentos del otro color.

Recientemente, en 2012, se ha publicado un algoritmo matemático que permite resolver el problema en $O(n^2)$ (en adelante denominado algoritmo BR) para el problema planteado.

El objetivo de este proyecto de título es aplicar conceptos derivados de la geometría computacional para implementar en lenguaje *Java* el algoritmo BR, de manera tal que pueda ser utilizado por usuarios para distintas aplicaciones.

3.1.2. Objetivos Específicos:

1. Analizar y comprender el algoritmo BR. El algoritmo BR implica varios otros algoritmos de Geometría Computacional, por ejemplo, los algoritmos que encuentran la Cerradura Convexa y la implementación de un caliper rotatorio.
2. Diseñar un modelo de clases (puntos, segmentos, cerradura convexa, etc.) apropiado a la implementación del algoritmo, así como también las estructuras de datos anexas (Stack, listas, arreglos, etc) necesarias.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

3. Diseñar una interfaz de usuario apropiada para la utilización del algoritmo BR.
4. Implementar, en lenguaje de programación Java, el algoritmo BR utilizando el diseño previo.
5. Experimentación y evaluación de la implementación del algoritmo BR con conjuntos de datos de diferente tamaño para lograr la verificación del orden $O(n^2)$ proveniente desde la teoría. No se compara con otros algoritmos puesto que matemáticamente (en la publicación) se demostró que es el algoritmo con el mejor orden posible de obtener.

3.2. Ambiente de Ingeniería de Software

El desarrollo de esta actividad de titulación se llevará a cabo siguiendo un proceso metodológico que permita un aprendizaje acabado sobre algoritmos de geometría Computacional que conduzca a la implementación del Algoritmo BR de intersección bicromática entre dos conjuntos de segmentos.

Para ello se considera un primer periodo de introducción y comprensión de los algoritmos relativos a la geometría computacional existentes, a través de la lectura de artículos de los autores de la problemática, publicaciones relacionadas; para luego analizarlos y obtener un conocimiento que permita alcanzar los objetivos planteados.

De esta manera, el estudio de los distintos algoritmos de geometría computacional existentes, relacionados con la problemática propuesta, es muy importante. En base a estos algoritmos se construirá la implementación del algoritmo BR.

Una vez comprendidos estos algoritmos, se procederá al planteamiento de la implementación de este algoritmo en particular. Este periodo comprende otras actividades, como el aprendizaje y adaptación a las herramientas de desarrollo, el diseño de un modelo de clases apropiado a la implementación del algoritmo, el diseño de una interfaz de usuario adecuada para la problemática y la Implementación de las clases mediante los procedimientos ya diseñados con la respectiva interfaz de usuario diseñada. Estas actividades se desarrollarán bajo una metodología *CASCADA CON ITERACIÓN*.

La metodología usada para la implementación del Algoritmo BR está orientada a avanzar cuidadosamente en el desarrollo experimental, por lo que se propone una metodología que provea simpleza pero estructurada en las etapas de trabajo, enfocada en rescatar el

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

avance en la resolución de problemas sobre conjuntos de segmentos de colores opuestos.

Para efectos de la evaluación de la implementación se experimentará el comportamiento en diferentes casos, variando parámetros, midiendo tiempos de respuesta y documentando los resultados obtenidos.

3.3. Definiciones, Siglas y Abreviaciones

Para el desarrollo del presente proyecto, se debe dejar en claro los usos de las abreviaturas y definiciones usadas en el contexto propuesto. Además, la naturaleza del proyecto basado en la geometría computacional, requiere del uso de expresiones para describir correctamente la problemática. Se presenta la siguiente tabla con las expresiones, definiciones y abreviaturas requeridas.

Definiciones, Siglas y Abreviaciones	
Expresiones, Definiciones y Abreviaturas	**Significado**
R	Grafo. Conjunto de puntos Rojos
B	Grafo. Conjunto de puntos Azules
Sb	El Conjunto de Segmentos Azules que intersectan al menos un segmento Rojo.
Sr	El Conjunto de Segmentos Rojos que intersectan al menos un segmento Azul.
nr	Tamaño del Conjunto R
nb	Tamaño del Conunto B
S	R U B (Unión de los Conjuntos R y B)
n	nr + nb (la suma de los tamaños de los conjuntos R y B)
sb	Tamaño de Sb
sr	Tamaño de Sr
CH( )	Operación Cierre Convexo (Convex Hull) sobre un grafo
G (V(G), E(G))	Grafo geométrico. Unión de dos Grafos Disjuntos.

Sigue en la página siguiente.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Definiciones, Siglas y Abreviaciones	
Expresiones, Definiciones y Abreviaturas	**Significado**
V(G)	vértices del Grafo
E(G)	Conjunto de segmentos de un Color que no se intersecta con otro segmento de color. Complemento de Sb.
Be	SubConjunto de los puntos azules Externos al CH(R).
Re	SubConjunto de los Puntos Rojos Externos al CH(B).
Bi	SubConjunto de los Puntos azules Internos al CH(R).
Ri	SubConjunto de los puntos Rojos Internos al CH(B).
Ge (V(Ge) , E(Ge))	Grafo Externo.
Gi (V(Gi), E(Gi))	Grafo Interno.

Tabla 3.1: Tabla de Expresiones, Definiciones y Abreviaturas

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Capítulo 4

Algoritmo BR

El algoritmo propuesto en (Cortés et al., 2012) se presenta en forma teórica como una solución óptima en un tiempo $O(n^2)$ sobre un problema calificado de tipo $3 - Sum$, lo que indica que el problema solo puede ser optimizado a un orden cuadrático, tal como se expone. Este procedimiento obtiene todos los segmentos azules que intersectan por lo menos un segmento rojo y sus respectivas cardinalidades (ver Algoritmo 4.0.1).

Este algoritmo detecta cuales segmentos están fuera del conflicto propuesto, pues son zonas en donde no se generan intersecciones bicromáticas (llamadas también zonas de intersecciones monocromáticas) y las aísla para generar una búsqueda limpia en las zonas restantes. Además, evitar intersecciones es una de las principales dificultades en muchos problemas geométricos asociados a los segmentos rojos y azules. Por ello, algunas veces no es necesaria la cantidad de intersecciones bicromáticas que se puedan obtener en un segmento dado, pero sí puede serlo una intersección que se considere prohibida, de este hecho, es que el algoritmo prioriza la descriminación de conjuntos, reduciendo la búsqueda, mejorando el rendimiento y obteniendo conjuntos significativos (explicados más adelante).

De esta forma, la búsqueda está enfocada en grupos de datos acotados y aplican distintos métodos y procedimientos ya desarrollados por otros investigadores para obtener una solución correcta en un tiempo de ejecución considerado óptimo para la clasificación del problema enfrentado.

Los autores han llamado a esta operación $Sep(S)$, donde S es el grafo que posee el espacio total que contiene a los dos conjuntos de puntos ($Sep(S)$ es llamado el algoritmo BR en este trabajo). En la figura 4.1 se muestra el resultado esperado de esta operación.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

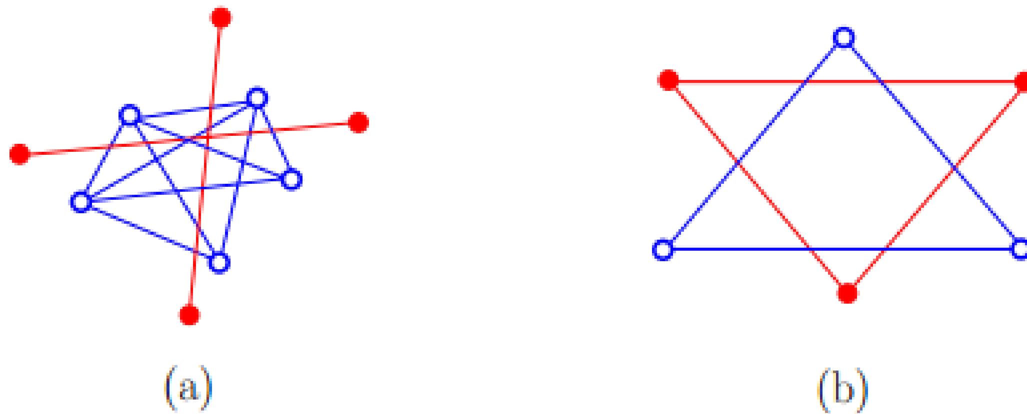


Figura 4.1: Ejemplo de Sep(S) (Cortés et al., 2012)

En la Ilustración 4.1, la imagen (a) tiene un $sb = 9$ y un $sr = 2$, mientras que la imagen (b) tiene $sb = 3$ y $sr = 3$. Además, se aprecia que la solución del algoritmo BR no es afectada si los conjuntos de puntos son del mismo tamaño y forma. Por otro lado, la solución no se compone de contabilizar el total de intersecciones bicromáticas de segmentos, sino que la solución se compone del conjunto de segmentos de cada color que provocan intersecciones bicromáticas de segmento. Lo anterior, se puede asimilar a un problema de discriminación de puntos rojos y azules, donde el criterio aplicable es la existencia de un segmento rojo dado por dos puntos rojos que separen algunos pares de puntos azules; en otras palabras, se desea obtener conjuntos disjuntas de puntos azules que no interfieren o se ven afectadas por pares de puntos rojos.

Se presenta el algoritmo propuesto por (Cortés et al., 2012) de forma descriptiva, mencionando los métodos y técnicas ordenadamente usadas para construir una solución a esta problemática. Cada paso y sub-procedimientos son enmarcadas en los estudios previos de muchos autores de la geometría computacional, de los cuales han estudiado problemas similares o totalmente distintos, sin embargo sus proposiciones son aplicables y exportables a este problema. En adelante el algoritmo.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

4.0.1. Algoritmo BR

Algoritmo $Sep(S)$ **(BR): Segmentos Intersectados Rojo-Azul (RED-BLUE)**
Entrada: $S = B \cup R, |B| = |R| = n.$
Salida: $Sep(S) = (Sb, Sr)$ y $(sb, sr).$

1. Calcular V(Ge) y V(Gi) en tiempo de $O(nlogn)$ determinando los puntos azules que están en el exterior e interior al $CH(R)$, respectivamente.

2. "Cálculo de $E(\overline{Ge})$" :
   a) En tiempo O(n log n) calcular ambos órdenes rotacionales, horario y antihorario, de los puntos azules de V(Ge) con respecto a $CH(R)$.
   b) Calcular el conjunto $E(\overline{Ge})$ de los segmentos azules con vértices en V(Ge) intersectando a $CH(R)$ usando un cáliper rotatorio sobre $CH(R)$. La complejidad de tiempo es $O(|E(\overline{Ge})| + nlogn)$, es decir, que depende de la cardinalidad de su salida que a lo sumo es $O(n^2)$.

3. "Cálculo de $E(\overline{Gi})$" : Sea V(Gi) = $\{b_1, ..., b_m\}$ y $C_1(b_1) := CH(R)$.
   a) En tiempo $O(mn) = O(n^2)$ construir el arreglo dual A de líneas desde los puntos de $R \cup V(Gi)$.
   b) **For** $k = 2, ..., m$ **do**
      i) En tiempo $O(logn)$ calcular la región convexa actual donde b_k es contenida.
      ii) **If** b_k es contenida en $C_{k-1}(\phi)$ **then** actualizar $C_{k-1}(\phi) := C_k(b_k)$ en tiempo $O(1)$. *Else* "b_k es contenida en $C_{k-1}(b_j)$, $j < k$" proceder como sigue:
         A) En tiempo $O(n)$ hacer lo siguiente:
            1) Usar el arreglo A para obtener el orden rotacional horario de los puntos rojos con respecto al punto b_k (b_k es una línea en el arreglo dual A; atravesamos esta línea, en el orden dado por A, descubriendo el orden en que línea dual roja lo cruza, de este modo obteniendo el orden rotacional horario de los puntos rojos con respecto a b_k.),
            2) Análogamente, usar el arreglo A para obtener el orden rotacional en sentido horario de los puntos rojos con respecto al punto b_j,

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

3) Mezclar ambos órdenes rotacionales para calcular el orden rotacional en sentido horario de los puntos rojos con respecto al segmento azul $\overline{b_k b_j}$.

B) Usando el Lema 3, en tiempo $O(n)$ determine si existe un segmento rojo s_{jk} intersectando el segmento azul $\overline{b_j b_k}$.

C) **If** la respuesta es afirmativa **then** dividir la región convexa $C_{k-1}(b_j)$ usando el procedimiento de división de $CH(R)$ en regiones convexas, detallado en la sección 7.4. **Else** "$b_k \sim b_j''$ **then** actualizar $C_{k-1}(b_j) := C_k(b_j)$ en tiempo $O(1)$.

iii) En tiempo $O(1)$ añadir b_k al conjunto de puntos azules en su región convexa definitiva.

c) Calcular $E(\overline{Gi})$ desde los subconjuntos disjuntos de puntos azules contenidos en regiones convexas no vacías.

4. En tiempo $O(n^2)$ calcular el conjunto $Sb = E(\overline{Ge}) \cup E(\overline{Gi}) \cup \{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$ y el valor sb.

5. Proceder análogamente para calcular el conjunto Sr y el valor sr.

Algoritmo 4.0.1: Procedimiento propuesto para Reportar teóricamente las Intersecciones Bicromáticas de Segmento.

Este Algoritmo tiene un tiempo $O(mn + mlogn + nlogn) = O(n^2)$, ya que el paso 3.b alcanza dicha complejidad. Esto da paso a que los autores del procedimiento establezcan un teorema que afirma que la solución de este problema puede ser computadas en tiempo y espacio $O(n^2)$.

4.0.2. Explicando el Algoritmo BR

El algoritmo plantea dividir el problema en tres casos de intersección, ya que la solución se expresa de esta forma:

$$Sb = E(\overline{Ge}) \cup E(\overline{Gi}) \cup \{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$$

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Ecuación 4.0.2: Conjunto Solución del Algoritmo BR

La Ecuación 4.0.2 representa la unión de tres conjuntos distintos, los cuales se obtendrán dentro del algoritmo. Cada caso de intersección bicromática puede o no resultar vacío, ya que estos casos pueden no existir en un ejemplo dado.

Para entender de mejor manera los conjuntos solución, se definen tres conceptos:

DEFINICIÓN 4: *"Se llamará un segmento interno a una arista que ambos vértices (del mismo color) que la componen están dentro de un cierre convexo determinado".*

DEFINICIÓN 5: *"Se llamará un segmento externo a una arista que ambos vértices (del mismo color) que la componen están fuera de un cierre convexo determinado".*

DEFINICIÓN 6: *"Se llamará un segmento mixto a una arista (del mismo color) que posee un vértice dentro y otro fuera de un cierre convexo determinado".*

De las definiciones anteriores es posible complementar la ecuación 4.0.2. El primer caso es el conjunto $E(\overline{Ge})$ de los segmentos azules (de un color) que intersectan al cierre convexo $CH(R)$ (de otro color) de un lado al otro, es decir, los segmentos externos azule sque intersectan de lado a lado a $CH(R)$, esto puede apreciarse en la figura 4.2a. El segundo conjunto es $E(\overline{Gi})$ de los segmentos formados por puntos azules internos que intersectan por lo menos a un segmento rojo, es decir, los segmentos internos a $CH(R)$ que intersectan a los segmentos internos de otro color, la figura 4.2b muestra a este conjunto solución. Por último el conjunto $\{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$ que corresponde a los segmentos formados por un punto azul interior y otro punto azul exterior al $CH(R)$ que intersectan por lo menos un segmento rojo, es decir, los segmentos mixtos que se intesectan con otro segmento del otro color, esta solución puede verse en la figura 4.2c.

La ejecución del algoritmo consta de varias partes y, como ya se ha mencionado, cada una de ellas resuelve parte del problema ocupando métodos y técnicas específicas. Para lograr la solución correctamente, es necesario explicar cada una de estas partes. Es posible encontrar la explicación original del algoritmo en (Cortés et al., 2012). En adelante la explicación en detalle de cada paso.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

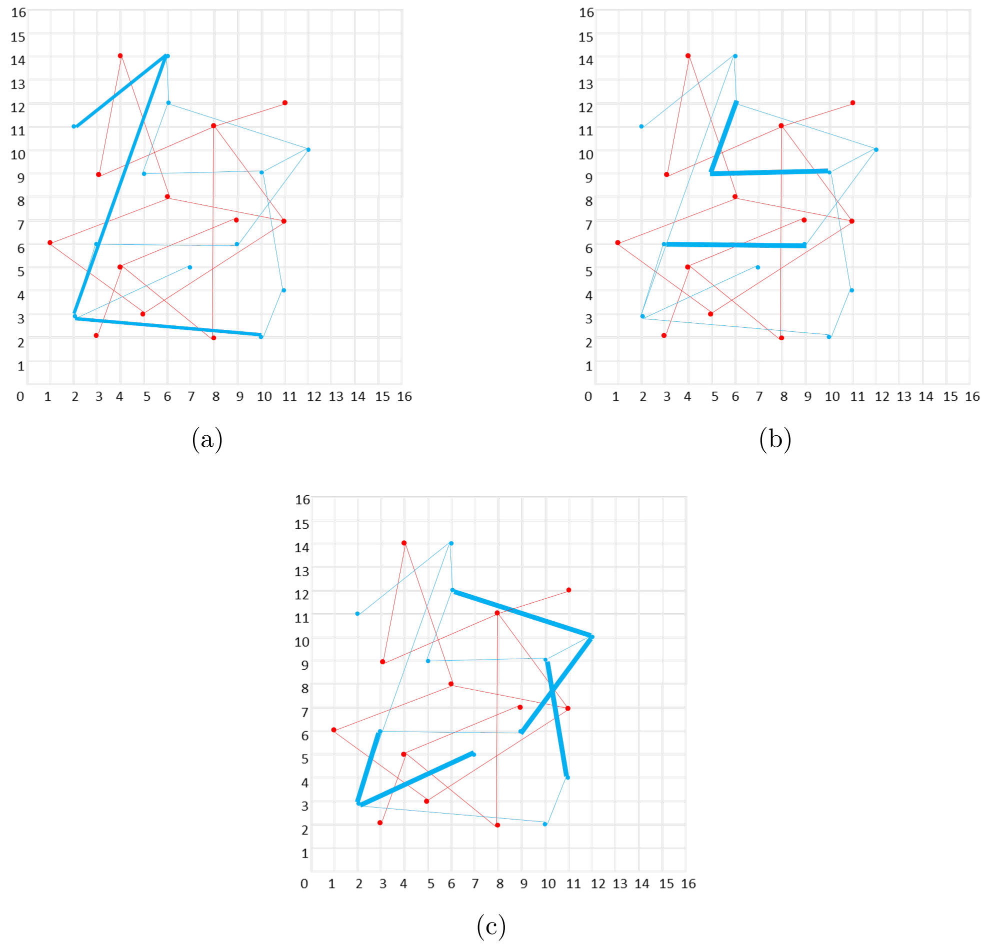


Figura 4.2: 4.2a es $E(\overline{Ge})$, 4.2b es $E(\overline{Gi})$ y 4.2c es $\{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$

Es muy importante recalcar que el algoritmo inicia el procesamiento de las intersecciones bicromáticas de segmentos del grafo azul (B) con respecto al grafo rojo (R), para luego aplicar nuevamente el procedimiento, pero procesando al conjunto rojo (R) con respecto al conjunto azul (B).

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

4.0.3. Parte 1: Cierre convexo $CH(R)$ y conjuntos $V(Ge)$ y $V(Gi)$

Al recibir los conjuntos de datos de entrada, es necesario identificar y separar los subconjuntos de vértices azules externos e internos. Para ello se definen los límites que separarán estos subconjuntos ubicados en los vértices extremos del grafo R, es decir, se obtendrá el cierre convexo del grafo R ($CH(R)$).

Este procedimiento debe ejecutarse en tiempo $O(nlogn)$, por lo que se debe utilizar un algoritmo que satisfaga esta restricción. Se ha escogido el algoritmo de Graham Scan (detallado en Sección 2.3, versión 2). De esta forma, se logra el cierre convexo en dicho orden (Ver Figura 4.3) y es posible identificar los vértices externos e internos a $CH(R)$.

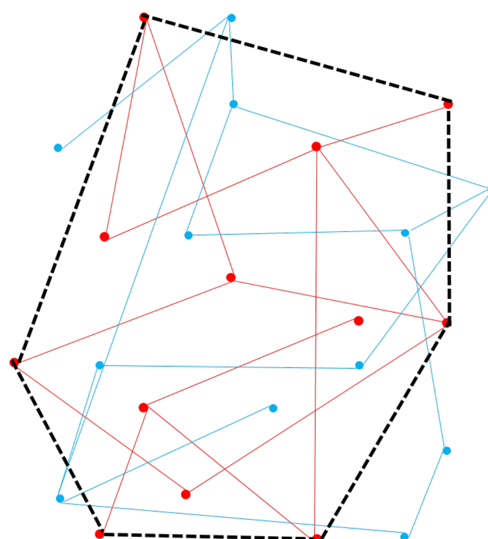


Figura 4.3: Cierre Convexo $CH(R)$

Para determinar los dos conjuntos de vértices, se ocupa el cierre convexo ya calculado. Este es un problema resuelto por Haines en (Haines, 1989), el cual se calcula, en un tiempo de $O(n)$, la ubicación del vértice con respecto al Polígono.

En el Algoritmo BR, para cada vértice del Grafo B se aplica el algoritmo desarrollado por Haines, comparándolo con el $CH(R)$. Una vez consultada y encontrada la posición del vértice, se añade a la lista de los vértices externos o internos, según corresponda.

Los resultados de este procedimiento son las dos conjuntos de datos externos al Polígono (Be) e internos al Polígono (Bi). En la Figura 4.4, los conjuntos de datos resultantes serían $Be = \{1, 2, 3, 9, 10, 12\}$ y $Bi = \{4, 5, 6, 7, 8, 11\}$.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

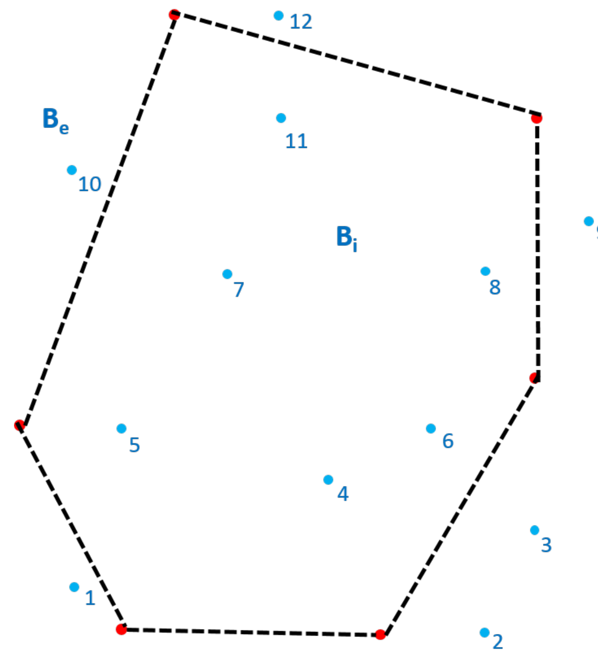


Figura 4.4: Puntos externos e internos al Polígono

Con esto se obtienen los resultados esperados para el paso 1, dando la base para que los conjuntos de datos puedan ser procesados en las siguientes etapas.

4.0.4. Parte 2: Obtener $E(\overline{Ge})$

Esta parte se centra en obtener el conjunto $E(\overline{Ge})$ (ya explicado anteriormente). Para lograr esta solución, es necesario utilizar una técnica que permita encontrar vértices que están en lados opuestos del polígono; lo que produce que se realice un número restringido de comparaciones, disminuyendo la complejidad del procedimiento al solo incluir en el análisis a ciertos puntos externos.

Esta técnica es el Cáliper Rotatorio (detallado en la sección 2.4) que logra resultados en buenos órdenes de tiempos de ejecución. El procedimiento pretende que este Cáliper realice un giro completo al polígono convexo $CH(R)$, analizando cada punto azul externo (ya calculado) que se encuentre entre las dos líneas de soporte del cáliper, consultando si intersectan completamente al polígono.

Sin embargo, antes de realizar el procedimiento, es necesario determinar el orden en el

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

cual el cáliper rotará, ya que es más sencillo guiar la rotación del Cáliper con respecto a los puntos azules externos que seguir la rotación de los ángulos del polígono convexo (o rotación por defecto), lo que afectará en la ejecución del procedimiento.

Es necesario obtener el ordenamiento rotacional de los puntos $V(Ge)$ con respecto a $CH(R)$ (detallado en la sección 7.2). Con la utilización de un ordenamiento rotacional, el giro del Cáliper Rotatorio es controlado y el número de comparaciones en la obtención de la solución es más limitado en cada rotación, ya que discrimina a ciertos puntos.

Una vez obtenido el ordenamiento Rotacional, se construye un cáliper sobre $CH(R)$, ubicando el par antipodal del cáliper en los vértices con menor y mayor coordenada Y, por lo que el Cáliper será paralelo a la horizontal, como punto de inicio. Luego, se ocupará el ordenamiento rotacional para dar la dirección al Cáliper Rotatorio en cada giro. En cada giro se consultará por cada uno de los vértices de $V(Ge)$ que estén entre las líneas de soporte paralelo.

En cada rotación se añade al vértice de $V(Ge)$ correspondiente a una lista que contenga los vértices de $V(Ge)$ que se encuentran dentro de las líneas paralelas del cáliper. Así, con este elemento se realiza una consulta con cada vértice dentro del cáliper, estas consultas son las siguientes:

1. ¿El elemento aún está dentro del cáliper después de la rotación?,
2. ¿Los dos vértices consultados forman una arista registrada en el grafo?, y
3. ¿Existe alguna intersección entre la arista de los vértices dentro del cáliper y la arista formada por el par antipodal del cáliper?.

Este procedimiento se basa en el Lema 2 propuesto por (Cortés et al., 2012):

"Sea Q un conjunto de puntos externos a un polígono convexo P, y p, $q \in Q$. El segmento $\overline{pq}$ intersectan a P si y solo si existe un cáliper de P a través de dos vértices u, v de P que contienen los puntos p y q, y $\overline{uv}$ y $\overline{pq}$ se intersectan".

Esto queda más claro en la Figura 4.5, donde se muestra un cáliper y dos vértices de $V(Ge)$ p y q que forman una arista que intersecta al par antipodal $\overline{uv}$.

Con esto se logra consultar las intersecciones bicromáticas de segmentos que se encuentran atravesando todo el grafo R, y lo más importante, es que se logra evitar las intersecciones monocromáticas y se minimizan las consultas a vértices que se encuentran en lados opuesto del polígono convexo $CH(R)$.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

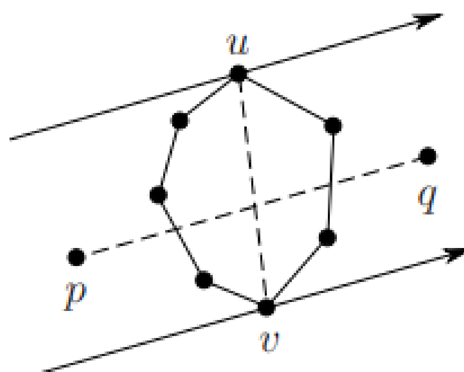


Figura 4.5: Lema 2, Intersección de un Polígono(Cortés et al., 2012)

4.0.5. Parte 3: Obtener $E(\overline{Gi})$

Este punto se centra en obtener el conjunto $E(\overline{Gi})$ explicado anteriormente. Este proceso se lleva a cabo mediante la división del polígono convexo formado por $CH(R)$, con el fin de separar el grafo azul en subgrafos azules completos que no tengan intersección alguna con segmentos rojos.

Tomando como entrada a $CH(R)$ y la lista de puntos azules pertenecientes a $V(G_i)$,el proceso consiste en ir agregando, uno a uno, los puntos azules al interior de $CH(R)$. En la primera iteración, el cierre convexo pasa a ser la única región convexa existente, la cual contendrá a un primer punto azul b_j: el punto b_1. En las siguientes iteraciones, se irán agregando los demás puntos, buscando en qué región convexa de las existentes se encuentra cada uno, de acuerdo a sus coordenadas. Si la región convexa no contiene puntos azules, este nuevo punto b_j simplemente se agrega a la región; en caso contrario, se traza un segmento entre b_j y un punto b_k, que ya estaba en la región convexa a trabajar, y se verifica si este segmento intersecta a algún segmento rojo.

Para encontrar dicha intersección, se procede aplicando un cáliper rotatorio (Ver sección 2.4) sobre el segmento $\overline{b_j b_k}$ y se aplica el lema 2 propuesto en (Cortés et al., 2012).

Para mejorar la eficiencia del giro del cáliper, al igual que en la parte 2 del algoritmo BR, es necesario conocer el orden rotacional de todos los puntos rojos con respecto a cada uno de los dos puntos azules que forman el segmento $\overline{b_j b_k}$. El orden rotacional mencionado

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

es obtenido a través de un arreglo dual de líneas; este proceso se detalla en el punto 3.a.ii.A del algoritmo BR descrito en la subsección 4.0.1.

Lo anterior se basa en una de las propiedades de la dualidad, la cual indica que el punto p se encuentra en la línea L sí y solo si el punto $\mathcal{D}(L)$ se encuentra en la línea $\mathcal{D}(p)$. De esto se desprende otra propiedad de la dualidad: *Las líneas L_1 y L_2 se intersectan en un punto p sí y solo si la línea $D(p)$ pasa a través de los dos puntos $\mathcal{D}(L_1)$ y $\mathcal{D}(L_2)$*, es decir, si varios puntos $p_1 ... p_n$ se encuentran en la línea L, todas las líneas $\mathcal{D}(p_1) ... \mathcal{D}(p_n)$ se encuentran en el punto $\mathcal{D}(L)$; en otras palabras, se intersectan todas en un único punto, formando un dibujo similar a un abanico.

Así, para encontrar el orden rotacional, solo basta con saber el orden en que todas las rectas duales rojas (obtenidas de cada punto rojo) intersectan a cada recta dual azul (primero la recta dual de b_j y luego la recta dual de $\overline{b_k}$). Para mayores detalles sobre arreglos duales de líneas, ver sección 2.5.

Para llevar a cabo esto en la implementación, se utilizó el mapeo $L : y = mx + b \Leftrightarrow p : (m, b)$, ya que es más simple para la aplicación que se necesita.

Llevando a cabo experimentos prácticos de orden visual, se determinó que el modo de ordenar las rectas duales para obtener el orden rotacional de los puntos de un color con respecto a cada punto del otro color es el siguiente:

- Encontrar los puntos de intersección de las rectas duales de un color respecto a cada recta dual del otro color.
- Ordenar con respecto a la abscisa los puntos de intersección encontrados, siguiendo el siguiente orden:
  - Primero se ordenan los puntos cuya abscisa es positiva, desde el 0 al $+\infty$.
  - Luego se añaden los puntos cuya abscisa es $+\infty$, seguidos de aquellos que tienen abscisa $-\infty$.
  - Finalmente, se ordenan los puntos cuya abscisa es negativa, desde el $-\infty$ hasta el 0.

Luego de verificar la existencia de intersección del segmento $\overline{b_j b_k}$ con algún segmento rojo, en caso de existir tal intersección, se procede a dividir la región convexa en dos, tres o cuatro nuevas regiones convexas siguiendo el procedimiento descrito en la sección 7.4. En caso de no existir alguna intersección entre el segmento $\overline{b_j b_k}$ y un segmento rojo, se dice que b_j y b_k están relacionados, por lo que se ingresa el nuevo punto azul a la región

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

convexa sin mayor problema.

Una vez ingresados todos los puntos azules a sus respectivas regiones convexas dentro de la partición de $CH(R)$, se procede a obtener el conjunto $E(\overline{Gi})$ a partir de los subconjuntos disjuntos de puntos azules contenidos en las regiones convexas no vacías. En (Cortés et al., 2012) se indica que toda arista azul que tenga sus vértices en regiones convexas distintas, es parte de $E(\overline{Gi})$ (Ver figura 4.7), por lo que, ya conocidas las regiones convexas, basta con buscar entre las aristas azules, las que cumplan con esta propiedad.

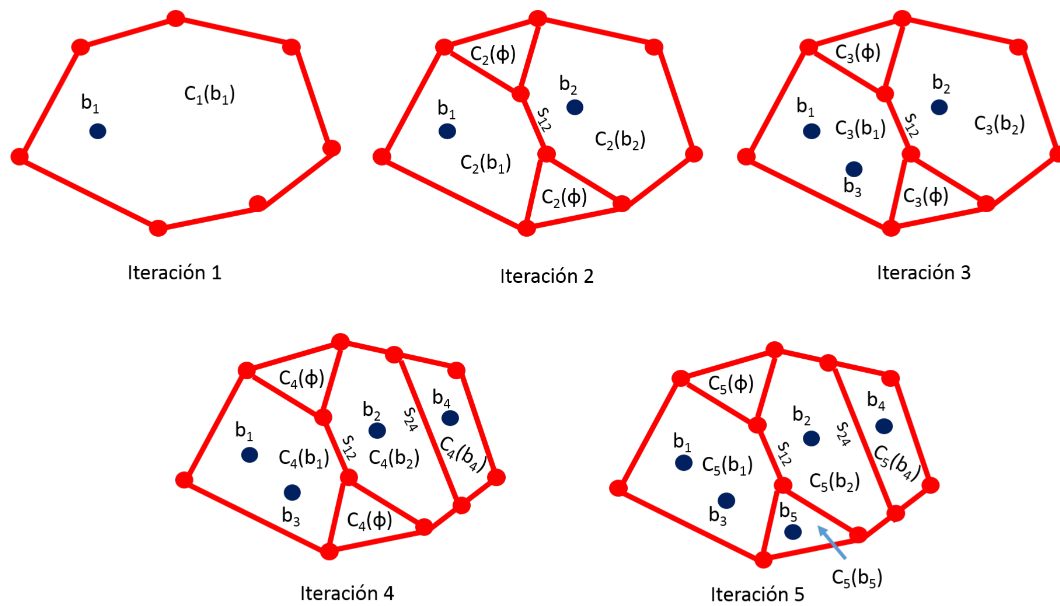


Figura 4.6: Instancia de las regiones convexas creadas hasta la iteración 5

En la figura 4.6, los puntos azules son representados por b_j, con $j = \{1, 2, \ldots, k-1\}$, donde j indica en qué iteración el punto fue incluido dentro de alguna partición; las regiones están representadas por $C_k(X)$, donde k indica el número de iteración y X indica el contenido de la partición, el cual puede ser vacío ($\emptyset$) o un punto azul b_j (solo se considera el primer punto azul ingresado a la respectiva región, el que pasa a ser una especie de "representante" de dicha región; los puntos que después se van agregando, se incluyen sin cambiar el representante).

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

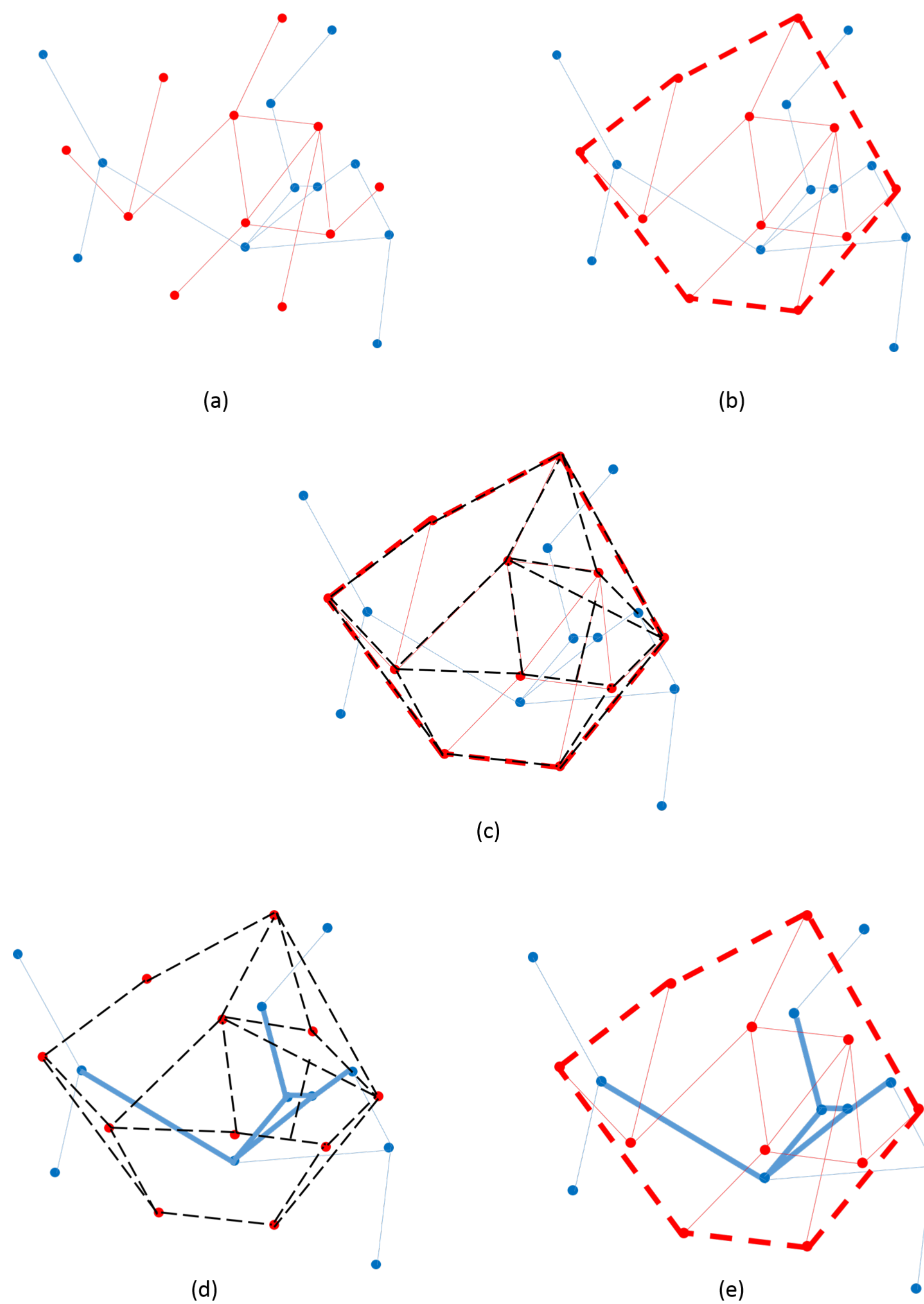


Figura 4.7: Ejemplo de búsqueda de $E(\overline{Gi})$. En (a) se puede ver dos grafos (rojo y azul). En (b) se observa $CH(R)$. En (c) ya se realizaron las particiones correspondientes. En (d) se resalta los segmentos azules que cruzan de una región a otra (se eliminó segmentos rojos para mejorar visibilidad). En (e) se resalta $E(\overline{Gi})$ dentro de $CH(R)$

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Adicionalmente, cada segmento rojo que intersecta a un segmento formado por dos puntos azules es representado con s_{jk}, donde j y k corresponden al número j de cada punto que forma el segmento azul intersectado (si el segmento intersecta al segmento $\overline{b_1 b_2}$, entonces el segmento se llamará s_{12}).

En la instancia mostrada en la misma figura, se puede ver cómo, en la primera iteración, la única región existente es la que corresponde al cierre convexo de los puntos rojos, y que además contiene al primer punto de la lista de puntos azules que pertenecen a $V(G_i)$.

En la segunda iteración, al existir un segmento rojo s_{12} que intersecta al segmento $\overline{b_1 b_2}$, se realiza una partición del polígono en cuatro partes (Ver procedimiento en la sección 7.4), quedando vacías dos de éstas. En la tercera iteración se incluye un tercer punto azul, el que es ubicado en la misma región que b_1 sin existencia de segmento rojo que intersecte al segmento $\overline{b_1 b_3}$, por lo que el punto azul se añade sin problemas, manteniendo a b_1 como representante de la región. En la iteración 4, se produce una nueva partición al agregar el punto b_4, ya que existe el segmento rojo s_{24} intersectando al segmento azul $\overline{b_2 b_4}$ (Notar que la región fue dividida solo en dos partes, ver procedimiento en sección 7.4).

Finalmente, en la quinta iteración, el punto azul b_5 fue ubicado sin problemas en una de las regiones vacías existentes.

4.0.6. Parte 4: Obtener $\{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$

Una vez obtenidos los dos conjuntos de segmentos de intersecciones bicromáticas dadas en el exterior del Polígono y las dadas en el interior de este, es posible determinar los segmentos que participan en una interseccion bicromática con un elemento dentro y otro fuera del Polígono, al utilizar las estructuras y técnicas ya desarrolladas.

Según los Autores en (Cortés et al., 2012), este procedimiento en el algoritmo BR tiene una complejidad de tiempo $O(n^2)$, sin embargo no se describe de forma clara las instrucciones para desarrollarla. Aún así, dadas las caracteristicas del problema, se puede deducir la forma de obtener la solución requerida.

El algoritmo BR ya en esta etapa ha construido y desarrollado varias estructuras que permiten sacar provecho de ellas. Así, la parte 4 se desarrollará de una manera simple , pero efectiva en el rendimiento. El procedimiento tendrá dos subprocedimientos, que ayudarán en la discriminación de aristas y vértices que no son necesarios consultar.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

El primero de ellos es el preprocesamiento de los puntos azules; el objetivo de ello es encontrar los pares de vértices azules (uno perteneciente a B_e y el otro a B_i) que forman una arista en el grafo B. Para este propósito, se consulta, para cada vértice de Be, todos los vértices que pertenecen a Bi y, en caso de formar una arista, se almacenará en una lista de Aristas uv (por el conjunto $\overline{uv}$); este procedimiento omite las aristas de vértices de los mismos conjuntos, ya que no son consultados. Este paso se consigue en un tiempo $O(n^2)$, pues se recorre los dos conjuntos Be y Bi completamente (ver Figura 4.8).

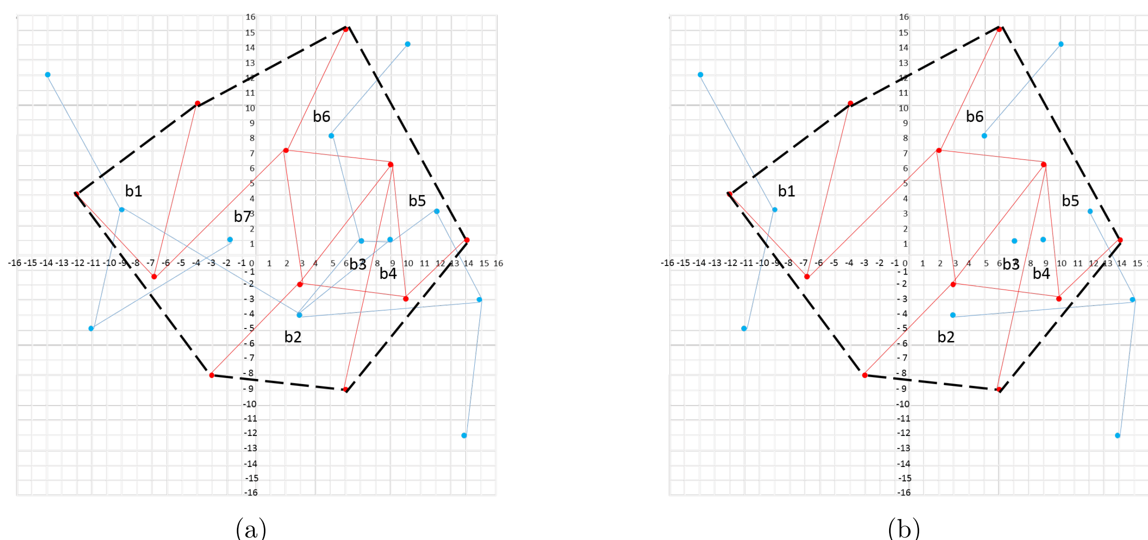


Figura 4.8: Selección de aristas $\overline{uv}$

En el ejemplo, las aristas escogidas serían $\{([(-9,3)-(-14,12)]\ , [(-11,-5)-(-9,3)]\ , [(3,-4)-(15,-3)], [(12,3)-(15,-3)]\ , [(5,8)-(10,14)])\}$, dejando a las otras aristas fuera del siguiente procedimiento.

El segundo preprocesamiento consta de una búsqueda similar, pero de las aristas pertenecientes al grafo rojo. Este conjunto de Aristas tendrá todas las aristas pertenecientes al grafo rojo, es decir, todas las aristas del grafo rojo. Este procedimiento tiene una complejidad de $O(n^2)$.

Una vez obtenidos ambos preprocesamientos, es posible realizar una búsqueda entre las aristas del Grafo Be hacia Bi, y las aristas del grafo rojo. Los elementos se compararán uno a uno, y se consultará si estos segmentos están intesecados, de ser así, el segmento azul (o rojo en caso inverso) será incluida en la parte de la solución de $\{\overline{uv} : u \in V(Gi), v \in$

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

$V(Ge)\}$. La complejidad de este proceso es de $O(n * m) = O(n^2)$.

Finalmente, el procedimiento presente cumple con lo expuesto en (Cortés et al., 2012), ya que tiene una complejidad de tiempo $O(n^2)$ en total, pues cada preprocesamiento y la operación de búsqueda las intersecciones dadas entre segmentos con vértice dentro y fuera del CH(R), son de complejidad $O(n^2)$.

4.0.7. Parte 5: Obtener Sr

En este punto, el algoritmo ya ha logrado obtener la mitad de la solución, es decir, el conjunto de los segmentos azules que intersectan por lo menos una vez segmentos rojos.

Para obtener las intersecciones bicromáticas de los segmentos rojos sobre los segmentos azules, se debe utilizar el mismo procedimiento explicado anteriormente; análogamente, se aplican los pasos 1, 2, 3 y 4, cambiando el orden de los grafos a usar, es decir, en un principio el algoritmo se aplicó (R, B), ahora se aplicará (B, R).

De esta forma, el algoritmo calculará la siguiente expresión:

$$Sr = E(\overline{Ge}) \cup E(\overline{Gi}) \cup \{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$$

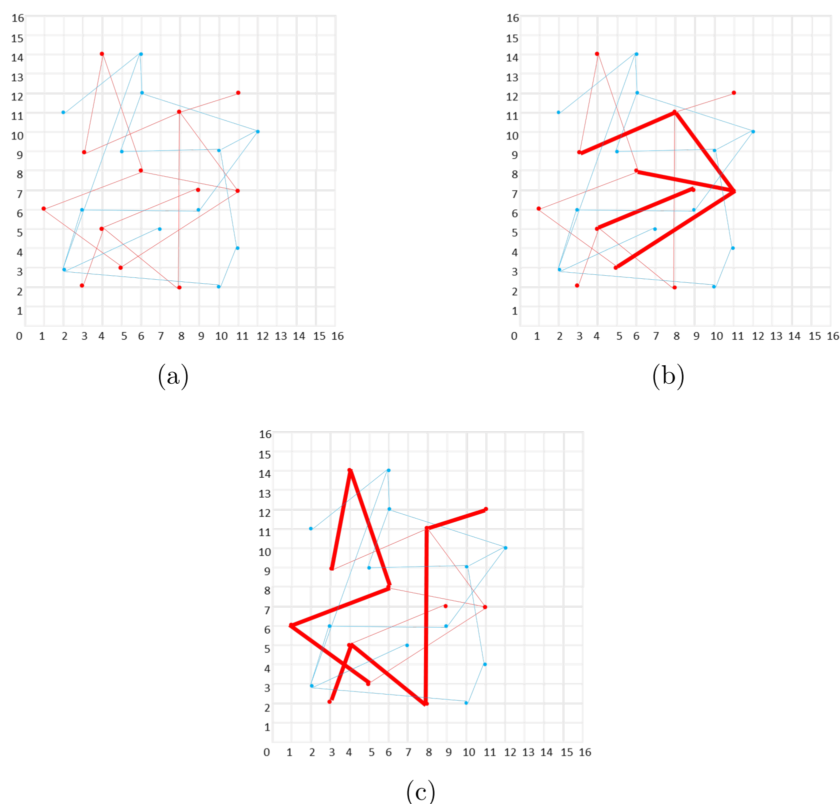
Ecuación 4.0.7: El conjunto solución de segmentos rojos que intersectan por lo menos una vez segmentos azules.

Esta expresión es idéntica a la Ecuación 4.0.2, donde se presenta la solución para el conjunto Sb, por lo que el procedimiento funciona de igual forma para encontrar este conjunto Sr. Es posible ver la solución del conjunto Sr en las siguientes Figuraes.

En la Figura 4.9 es posible ver la distribución del contenido de la solución. En este caso, el conjunto $E(\overline{Ge})$ es vacío, pues no existen segmentos que atraviesen completamente al otro grafo. Mientras que en el caso de $E(\overline{Gi})$ y en $\{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$ se encuentran los segmentos que generan intersecciones bicromáticas.

Este proceso no afecta a la complejidad total del algoritmo, ya que se realiza el procesamiento dos veces, lo que no aumenta sustancialmente en tiempo de ejecución. Esto se explicará más en profundidad en el capítulo de Complejidad.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile



(a) (b) (c)

Figura 4.9: 4.9a es $E(\overline{Ge})$, 4.9b es $E(\overline{Gi})$ y 4.9c es $\{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$

4.1. Conclusión del Algoritmo BR

El algoritmo BR plantea una búsqueda estratégica sobre los conjuntos de puntos, disminuyendo la complejidad y evitando la consulta de intersecciones monocromáticas. El algoritmo planteado tiene una complejidad cuadrática, lo que satisface la problemática de complejidad 3-Sum Hard, por lo que se considera un algoritmo óptimo.

Además, el procedimiento logra dividir los conjuntos de datos y los procesos en casos especiales, lo que otorga escalabilidad al algoritmo, entregando también partes específicas de la solución que pueden tener más importancia en otras aplicaciones.

Es por ello que el algoritmo tiene nuevas propiedades con un rendimiento óptimo, demostrado en (Cortés et al., 2012). Sin embargo, este nuevo algoritmo no tiene registros

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

de resultados de experimentación en una implementación en software con volúmenes de datos variados.

El presente proyecto intentará proveer dichos registros y acotar conclusiones sobre los resultados obtenidos en la experimentación.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Capítulo 5

Especificación de Requerimientos de Software

5.1. Alcances

El software proporciona información sobre las intersecciones bicromáticas de segmentos encontrados en dos conjuntos distintos (llamados Rojo y Azul, respectivamente), es decir, entrega un listado de los segmentos involucrados de un conjunto sobre el otro (segmentos azules que intersectan segmentos rojos y viceversa); entrega la cardinalidad del listado producido; permite ingresar como entrada los archivos que contiene la información de los grafos rojo y azul (sus vértices y aristas); y, opcionalmente, permite la creación aleatoria de un grafo (que puede recuperarse en un archivo). Adicionalmente, se implementa una interfaz gráfica que permita dibujar grafos rojo y azul para observar/explicar el comportamiento del algoritmo de una forma visual, sin embargo, estos grafos están limitados por el número de vértices y aristas posibles de dibujar sobre el área de pantalla destinada a estos efectos, para facilitar la visibilidad de la solución.

El desarrollo de este software tiene por fin experimentar con la aplicación de un algoritmo formulado y publicado por un grupo de investigación de matemáticas aplicadas; el propósito es lograr la implementación correcta de este algoritmo en el lenguaje de programación Java. El uso de esta herramienta permite el análisis específico de las intersecciones bicromáticas de segmentos sobre grafos.

Finalmente, esta herramienta se crea bajo la necesidad de implementar un nuevo algo-

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

ritmo propuesto desde la teoría de la geometría computacional en un enfoque matemático, inspirando el desarrollo del software para la experimentación de su desempeño a través de volúmenes de datos variados.

5.2. Objetivo del software

5.2.1. Objetivo General

Reportar las Intersecciones Bicromáticas de Segmentos a través de la implementación del Algoritmo BR, que permita la experimentación de distintos conjuntos de datos para la evaluación de su comportamiento.

5.2.2. Objetivos Específicos

El sistema permite:

- Gestionar los conjuntos de datos de entrada a analizar, ya sean obtenidos desde archivo generados aleatoriamente o de forma manual.

- Implementar el Algoritmo BR en todas sus etapas, entregando información de la ejecución de cada una de la etapas intermedias.

- Reportar la información sobre las intersecciones bicromáticas de segmentos presentes en dos conjuntos de datos (grafos R y B) distintos.

- Gestionar la Información de Salida y los resultados intermedios de la ejecución del algoritmo.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

5.3. Requerimientos Específicos

5.3.1. Requerimientos Funcionales del sistema

Requerimientos Funcionales del Sistema		
id	**Nombre**	**Descripción**
R01	Ingresar datos manualmente	El sistema debe contar con el ingreso de datos manualmente a través de la interfaz para probar posibilidades en la experimentación.
R02	Gestionar archivos Entrada	El sistema debe ser capaz de leer un determinado tipo de archivo para recoger volúmenes de datos más extensos. Además el sistema debe implementar la creación de grafos con datos aleatorios con un límite de datos especificado; debe existir la posibilidad de escribir en archivos los datos creados aleatoriamente para un uso posterior.
R03	Reportar Intersecciones Bicromáticas de segmento	El sistema debe implementar el Algoritmo BR y responder con la solución esperada. Además, debe utilizar los procedimientos establecidos e implementados de las complejidades del algoritmo en cuestión.
R04	Generar informe de Resultados de la solución	El sistema debe mostrar los resultados obtenidos de la ejecución del Algoritmo BR, tanto parcial como general de una manera comprensible y ordenada
R05	Generar informe de Resultados de la Ejecución	El sistema debe mostrar los tiempos de ejecución obtenidos del Algoritmo BR, tanto de forma parcial como general
R06	Gestionar archivos Salida	El sistema debe ser capaz de escribir un archivo donde almacene los resultados generales o parciales del análisis del Algoritmo BR.
R07	Guía de ayuda de usuario	El sistema debe contar con elementos de ayuda al usuario para guiar en el buen desarrollo de su ejecución.

Tabla 5.1: Requierimientos Funcionales del Sistema

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

5.3.2. Interfaces externas de entrada

El sistema permite la lectura de archivos de datos para permitir la ejecución del algoritmo.

Interfaces Externas de Entrada		
identificador	**Nombre del ítem**	**Detalle de Datos contenidos en ítem**
01	Datos del Grafo	Vértices, Aristas (los archivos serán nombrados: $< color >_< dimensión > .gr$)

Tabla 5.2: Interfaces Externas de Entrada

5.3.3. Interfaces externas de Salida

El sistema es capaz de guardar los resultados del análisis sobre los grafos.

Interfaces Externas de Salida			
id	**Nombre del ítem**	**Detalle de Datos contenidos en ítem**	**Medio Salida**
01	Informe de los Resultados	(Sb , Sr) con sb y sr correspondientes. Aristas de $E(\overline{Ge}) \cup E(\overline{Gi}) \cup \{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$	Archivo .dat (pantalla)
02	Informe de la Ejecución	Tiempos de la Ejecución Parcial de cada Paso y del Algoritmo Completo	Archivo .dat (pantalla)
03	Grafos Generados Manual o Aleatoriamente	Información de los grafos dibujados manualmente o generados aleatoriamente (Grafo Rojo y Azul)	Archivo .gr

Tabla 5.3: Interfaces Externas de Salida

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

5.3.4. Atributos del producto

El sistema cumple con todos estos atributos, que lo diferencian y caracterizan. Cada atributo especifica el comportamiento que tendrá el software y detalla las contingencias sobre los casos, donde estos atributos sean probados.

- FUNCIONALIDAD-CUMPLIMIENTO DE LA FUNCIONALIDAD
  El sistema debe ser capaz de cumplir con especificaciones descritas anteriormente. La principal preocupación es que responda tal y como se especifica en el algoritmo.

- FUNCIONALIDAD-PRESICIÓN
  El sistema debe corresponder a los resultados esperados y atender a los casos de excepciones en donde no se encuentre solución.

- USABILIDAD-OPERATIVIDAD
  El sistema debe contar con las contingencias para atender todas las excepciones que se produzcan, además debe contar con mensajes de error totalmente entendibles para el usuario, especificando el error que se presenta.

- EFICIENCIA-UTILIZACIÓN DE RECURSOS
  El sistema debe contar con las contingencias para el tratamiento de recursos esperado. Es decir, el sistema debe cumplir con el tratamiento realizado a los conjuntos de datos por separado, manteniendo una utilización de recursos eficiente.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Capítulo 6

Diseño

6.1. Diagrama de Clases

Para obtener el éxito en el desarrollo de este proyecto es necesario usar una representación de lo que se espera construir de forma entendible, esto se realiza para evaluar anticipadamente la forma de abordar la solución.

El proyecto presenta una problemática que requiere un diseño de diagramas de clases consistente y simple en la cantidad de clases ocupadas. Esto, debido a que el sistema debe ser construido para una funcionalidad específica y para algunas características simples.

Es por ello, también, que el diseño del diagrama de clases abarca el desarrollo del algoritmo presentado a partir de un enfoque Orientado a Objetos. A partir de esto, se especifica un análisis de criterio de diseño basado en la ejecución del Algoritmo.

6.1.1. Análisis de Diseño

El algoritmo es un procedimiento que posee elementos propios y métodos para subprocedimientos o auxiliares. Esto permite abarcar todos estos elementos en una clase, la que soporta elementos atributos y métodos. Esto, además, provee de una presencia y un comportamiento al algoritmo, lo que no limita a esta clase de crecer o ser utilizada en otro contexto distinto.

Para un problema que utilice grafos, es muy necesario contar con tres clases base. La primera, es el elemento mínimo una clase Vértice, que contenga las coordenadas de un

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

punto en un plano cartesiano. La segunda, es la clase arista, que es un segmento que relaciona a dos vértices, que contenga las referencias a sus dos vértices relacionados. Por último, la clase Grafo, que almacena un listado de los vértices y de las aristas pertenecientes al grafo. Junto con la forma de comportamiento de ésta.

La clase Grafo contiene métodos de análisis específicos para un grafo, tales como el algoritmo de Graham Scan para el Cierre convexo, y otros métodos que ayudan al algoritmo anterior, en búsqueda, ordenamientos y consultas sobre atributos y estados del grafo.

El algoritmo BR requiere de clases auxiliares para el tratamiento de partes del código, de forma de eliminar complejidad en la lectura del código y aumentar escalabilidad al procedimiento. Es por esta razón que existen las clases Cáliper Rotatorio (caliperRotatorio), Tangente (tangente) y Región Convexa (regionConvexa), recta Dual (rectaDual) y dual Rojo (dualRojo), las que a su vez soportan atributos y métodos propios. La clase de algoritmo BR (algoritmoBR) contiene el procedimiento principal de este proyecto, llamado método BR(), que recibe dos grafos (B, R). Además, contiene los algoritmos para obtener la ubicación de un punto respecto de un polígono, métodos de ordenamiento merge sort, algoritmo de localización de un punto respecto a una recta, método de obtención de rectas tangentes, obtención del Orden Rotacional, Creación de un Arreglo de Líneas. La clase contiene los atributos para los conjuntos solución de cada paso por separado y el conjunto total, y las variables de tiempos de ejecución de cada uno de ellos.

Como ya se ha mencionado, existen clases que ayudan a obtener los resultados de algunos de los pasos del algoritmo BR, estos tienen un comportamiento propio y atributos propios, por ello se incluye como clases. A continuación se mencionan:

La clase Tangente es parte importante para desarrollar el orden rotacional en la Clase Algoritmo BR, en ella se encuentran los atributos para los vértices que forman la recta tangente al polígono convexo para cada punto externo a él, además, de sus respectivas pendientes. Sus métodos contemplan el cálculo y acceso a sus atributos.

En el Paso 2, parte (b) se utiliza un caliper rotatorio para realizar una búsqueda optimizada sobre elementos alrededor de un polígono convexo. Sus atributos principales son los vértices antipodales, este par de vértices son los que mantienen al caliper sobre el polígono convexo; existe una lista de vértices que almacena los vertices que estan dentro de las líneas paralelas de soporte del caliper rotatorio, este listado va actualizandose en cada giro. Esta clase tiene su propio comportamiento sobre los conjuntos de datos, los métodos

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

de esta clase describen la forma de manipular estos datos, a través de los métodos de movimientos y acceso a los datos, y en métodos de búsqueda de intersecciones.

En el paso 3 del algoritmo BR, es necesario calcular el Arreglo Dual de líneas, para ello se cuenta con dos clases. La clase dual Rojo, que es una recta formada por vértices de un vértice azul y un vértice rojo, esto representa a la recta dual obtenida de cada punto rojo, se considera una recta roja; además, contiene un atributo que indica la posición en el orden rotacional del Arreglo Dual, útil para obtener las regiones convexas del polígono convexo. Mientras que la clase recta Dual contiene al vértice azul asociado y una lista de las rectas desde el vertice azul a cada vertice rojo, luego esto es utilizado para determinar un orden rotacional en el que se obtendrán las particiones del Cierre convexo.

La clase de región Convexa (regionConvexa) es también parte importante del paso 3 del Algoritmo BR, pues las intersecciones bicromáticas de segmentos son descubiertas luego de que un polígono formado por el cierre convexo se va dividiendo en regiones convexas. Esta clase almacena una lista de vértices, que representa los límites de la región convexa, y otra lista de los vértices azules que estan dentro de la región. Los métodos de esta clase corresponden al acceso a los datos y a consultas sobre el estado de la misma.

Para desarrollar el paso 4 del algoritmo BR no es necesario crear una nueva clase, ya que las clases existentes son capaces de dar soporte a la búsqueda de la solución del algoritmo. El paso 5 no requiere implementación en el diagrama, ya que el paso consta en realizar la operación del Algoritmo BR con los conjuntos de datos en sentido opuesto, es decir, que el conjunto R sobre el conjunto B.

El Software requiere diseñar una forma de comunicación e interfaz para el sistema. Para ello se construye esta clase, llamada interfaz de usuario, la que contendrá todos los elementos visuales para la comprensión del usuario.

El diagrama de clases está enfocado en resaltar los métodos usados en la investigación del proyecto, cada clase es, como ya se ha mencionado, parte de los algoritmos y procedimientos para obtener la solución esperada para el Algoritmo BR. Por ello, el diagrama se limita a la implementación del algoritmo BR.

Según lo detallado anteriormente, se presenta el siguiente diagrama de clases (Ver imagen 6.1. Para visualizar mejor cada detalle del diagrama se presenta en el anexo **??** cada clase por separado).

El diseño de clases presentado muestra no tan solo la composición de cada clase y

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

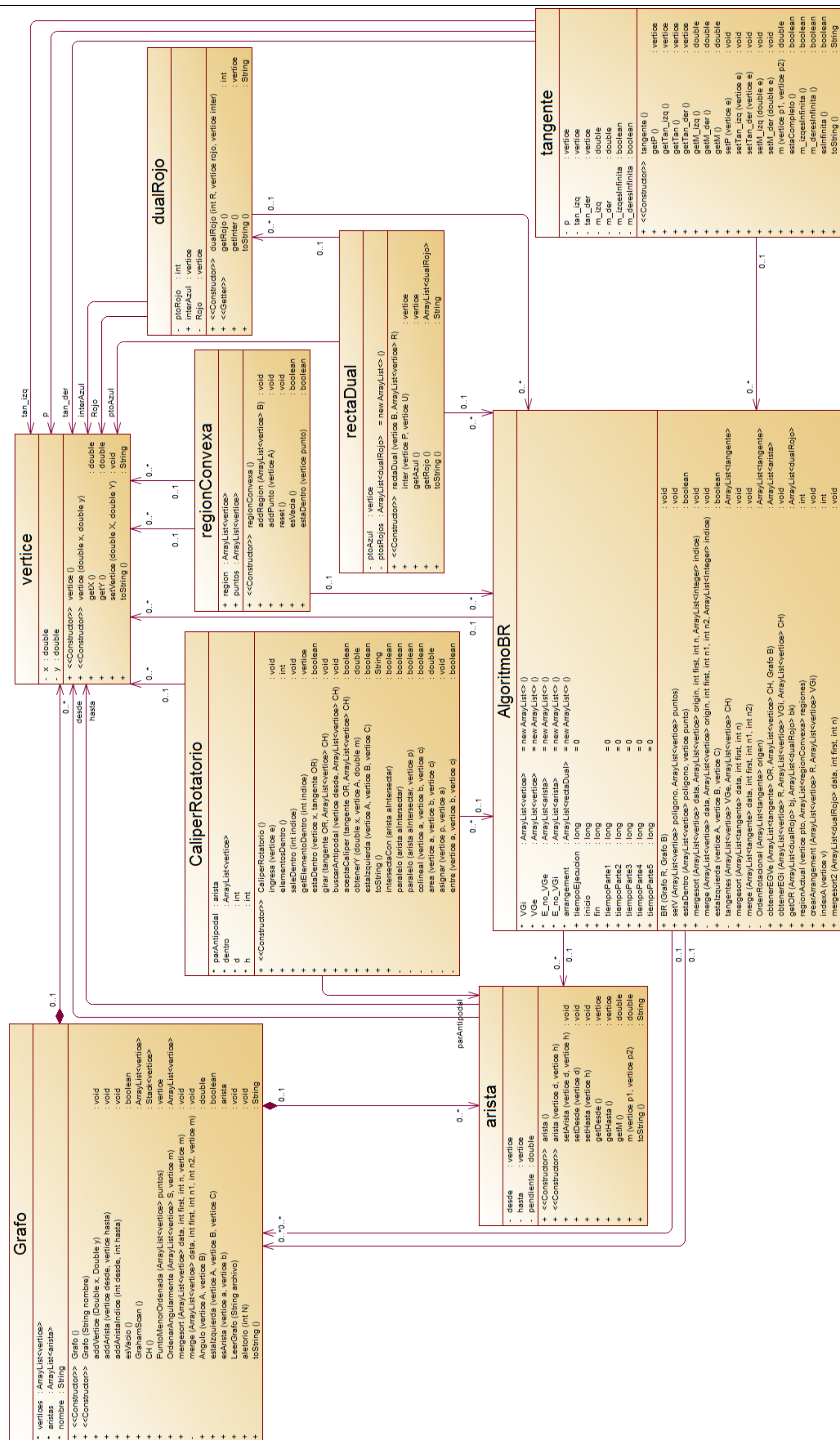


Figura 6.1: Diagrama de Clases

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

su comportamiento, si no que también con las relaciones que tiene con las demás clases existentes en el paquetes correspondiente al algoritmo BR.

Una de las relaciones más comunes dentro del diagrama son la composición de los vértices y de las aristas para con el Grafo, esto ya que tanto las instancias de vértices y de aristas no pueden existir si no existe una instancia de la clase Grafo para contenerlas.

Las relaciones existentes entre las demás clases son asociaciones, ya que no existe alguna dependencia entre ellas. Todas las clases se asocian a la clase vértice y la mayoría a la clase arista y solo la clase Algoritmo BR se asocia con la clase Grafo. Esto es porque la mayor parte de los procesos se realizan sobre las instancias de vértice o de aristas, y solamente la clase algoritmo BR utiliza las intancias de estos dos conjuntos para ordenar los procedimientos para uno o para otro.

De la misma forma, es la clase algoritmo BR la que tiene asociaciones con las demás clases (tangente, recta Dual, dual Rojo, Caliper Rotatorio y región Convexa), ya que es ésta quien las utiliza para obtener la solución del algoritmo. Es importante mencionar que las asociaciones tienen cardinalidades de 0 a muchos (0..*) en dirección a la clase Algoritmo BR, mientras en las direcciones opuestas las cardinalidades son de 0 o 1 (0..1), esto es porque la clase Algoritmo BR es quien dirige el procedimiento.

Por otro lado, la relación entre la clase dual Rojo y la clase recta dual se debe a que esta última contiene una lista de instancias de la clase dual Rojo. Esto es útil en la búsqueda del OR a partir del Arreglo Dual que es parte del paso 3 del Algoritmo BR.

Así, las relaciones en el diagrama de clases representa el control de las interacciones de las clases sobre las demás, regulando el acceso a los datos, los privilegios sobre ellos y la existencia de las instancias.

6.2. Diseño de Archivos de Entrada

El sistema requiere de un tipo de archivo que logre representar un grafo con todos sus datos. Para ello es necesario organizar un documento y especificar qué tipo de dato es necesario, el formato y la extensión del archivo.

La importancia radica en el objetivo del proyecto, ya que se espera esperimentar con datos tanto de casos específicos y, además, con grandes volúmenes de datos. El poder representar todos estos casos de grafos en archivos de entrada significa una ayuda en el

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

proceso de experimentación.

Para lograr un buen resultado se expresa las siguientes consideraciones para confeccionar el estándar de archivo que manejará el sistema:

6.2.1. Estructura del Archivo

La estructura del archivo contará con dos partes. La primera, los Vértices presentados con sus coordenadas (X, Y) una seguida de la otra, separadas por un salto de línea; a partir del primer par se numerarán los vértices secuencialmente desde 0. En la segunda parte, las Aristas son representadas con los índices de los vértices presentes en el archivo, ya que al ingresar cada vértice se crea un índice para acceder a él, es decir, que las aristas son números enteros que representan la posición de un vértice en la lista de vértices del grafo correspondiente, por lo que se indica la posición del vértice de inicio y el de fin para formar una arista.

Arraylist<> vertices

(4,3)	(2.3 , 5)	(9 , 5.2)	(10,10)

Arraylist<> aristas

(4,3)-(2.3 , 5)	(4,3)-(10,10)	(2.3 , 5)-(9, 5.2)	(9, 5.2)-(10,10)

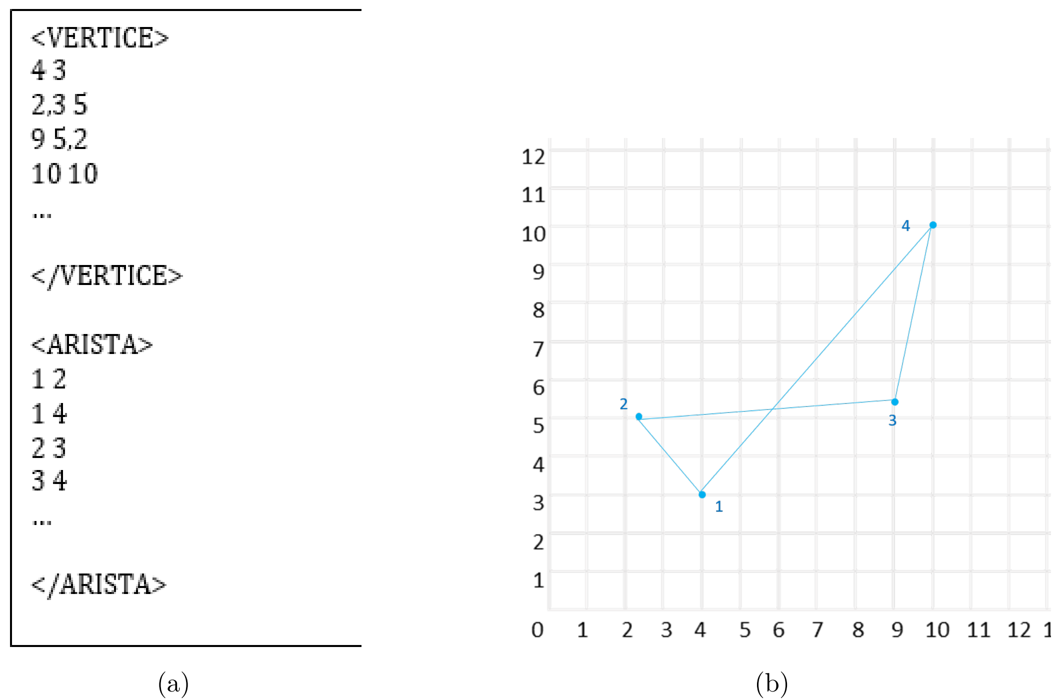
Figura 6.2: Representación de las Listas de vértices y aristas

Para separar estas secciones dentro del documento se utilizarán textos marcados con una sintáxis similar a la de los lenguajes de texto marcado HTML y XML. Estos textos solo tendrán este objetivo y no representan mayor funcionalidad. Es importante separar los conjuntos de datos de vértices y de aristas para no incurrir en errores en la ejecución del Software.

Los textos serán “VERTICE” y “ARISTA” que estarán escritas entre los símbolos “< xxxx >”. Tal como se muestra en el ejemplo de la figura 6.3.

De esta forma, la información de los vértices y de las aristas queda protegida, logrando que sea entendible y estándar para todos los archivos de carga.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile



(a) (b)

Figura 6.3: Estructura de Archivo de Entrada de Datos

6.2.2. Tipos de Datos y Formato

La forma en como se presenta la información es importante para no presentar problemas en la carga de los datos. Es por ello que es necesario detallar la forma correcta de presentar esta información en los archivos de entrada.

El tipo de dato de los vértices es de punto flotante, lo que en lenguaje de programación es un tipo de dato Double. Esto permite almacenar las coordenadas X e Y con números decimales sin problemas. En cambio, el tipo de dato presente en Arista es vértice, sin embargo para lograr representar correctamente la información de los vértices involucrados se usarán los índices de los vértices relacionados, es decir, el número que indica la posición dentro del listado de vértices del grafo, lo que permite guardar la referencia del vértice que se utilizará.

El formato para presentar a los vértices puede apreciarse en la Ilustración 6.3, donde cada vértice se escribe "coordenada Xcoordenada Y", es decir, el número de la coordenada X, espacio, y el número de la coordenada Y. Luego, se escribe un salto de línea y siga la

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

lectura de otro vértice. Es importante mencionar que las coordenadas aceptan número con punto flotante ().

El formato para presentar a las aristas puede apreciarse en la Ilustración 6.3, donde cada arista se escribe "<índice vértice inicio> <índice vértice fin>", es decir, el número correspondiente al índice del vértice de inicio, y el número del vértice de fin. Luego, se escribe un salto de línea y se sigue la lectura de otra arista. Es importante mencionar que los índices solamente aceptan número enteros positivos.

De esta forma se especifica claramente el formato y el tipo de dato que se debe utilizar en este archivo.

6.2.3. Extensión del Archivo

Para diferenciar el archivo con otros ficheros de datos, se utilizará la extensión gr (abreviatura de "grafo"). Esto significa que un archivo que contenga información de un grafo se llamará "<nombre del archivo>.gr" de esta forma se diferenciarán de los archivos con información de resultado del sistema.

6.2.4. Usos Y Beneficios

Estos tipos de archivos se usarán exclusivamente para la carga de datos al sistema, y por lo demás, facilitar la carga de volúmenes masivos de datos.

Utilizar este formato de archivo logrará evitar problemas de carga de datos y reduce el tiempo de ingreso manual.

6.3. Diseño interfaz y navegación

El sistema contará con una única interfaz simple, ya que su funcionalidad es específica y con un único manejo de archivos. Las características más importantes son:

- El manejo de ingreso y archivos de entrada-salida sencillo y reconocible con un ícono representativo.
- Debe existir un cuadro de dibujo incorporado a la interfaz, para el ingreso de los vértices y aristas de un grafo de ejemplo (debe ser acotado a un límite de entradas por tema de visibilidad), además deben existir elementos de dibujo para confeccionar

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

el ingreso de datos a través de íconos representativos (y diferenciados para ambos colores).

- Se debe incorporar un cuadro de resultados. En él se mostrará la información de las intersecciones bicromáticas de segmento de forma ordenada y entendible. Además debe existir la posibilidad de almacenar o guardar los resultados en archivos.
- Debe contar con información que sirva de guía para el usuario en cada ícono y elemento de la intefaz.

Es necesario destacar que la aplicación es de índole experimental, y su uso está pensado en la experimentación, documentación y rendimiento del algoritmo propuesto para la problemática.

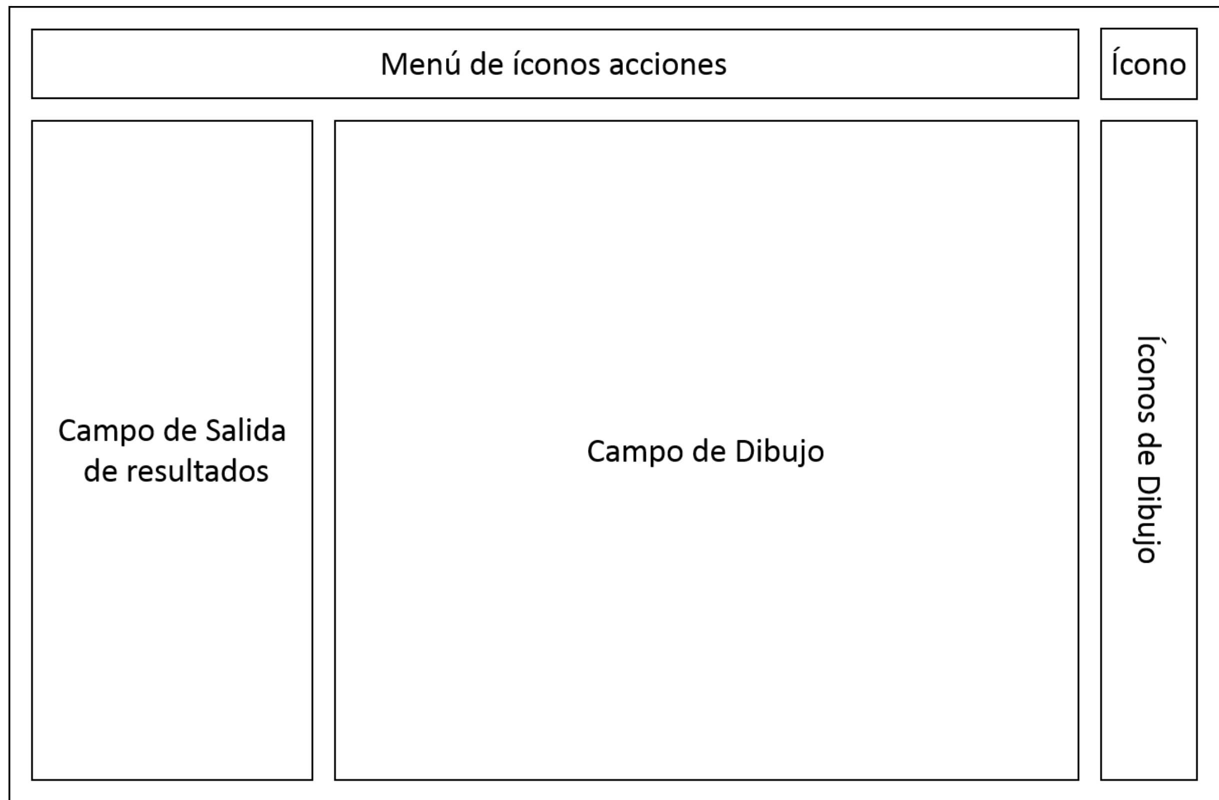


Figura 6.4: Interfaz de Usuario - Estructura

El sistema seguirá un diseño para la interfaz de usuario. El diseño considera la organización de los elementos dentro de la pantalla y el aspecto estético de la interfaz de presentación. Este diseño considera además los elementos gráficos a usar, imágenes e íco-

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

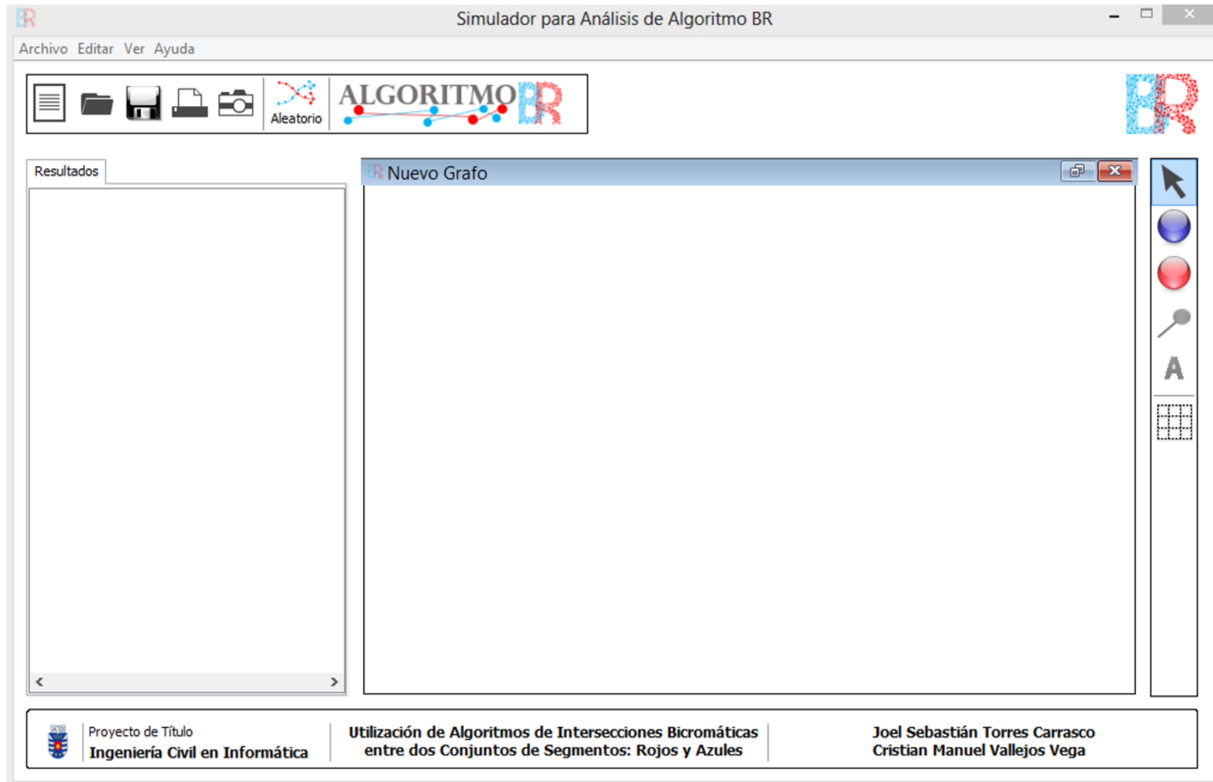


Figura 6.5: Interfaz de Usuario - Versión Final

nos. Se presenta este diseño en la ilustración 6.4, que muestra de forma esquemática la distribución de cada elemento de la interfaz. Se puede observar el resultado final de la interfaz para el usuario en la ilustragión 6.5.

Como puede apreciarse en la Ilustración 6.4, existe un diseño del esquema estructural de la interfaz. Cada campo tiene su funcionalidad y razones para su ubicación. Estas asignaciones se detallan en adelante:

- Menú de Íconos de acciones
  Este espacio de la interfaz tendrá íconos que permitirán realizar las acciones más importantes dentro del sistema. Estas acciones son "Abrir Archivo" (para grafos de ambos colores), "Generar Grafo Aleatorio" (para grafos de ambos colores), "Generar Grafo Manual", "Guardar Resultado" y "Generar Resultado" (algoritmo BR). Cada funcionalidad será representada por un ícono característico.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

- Ícono

  En este espacio de la interfaz se ubica el logo que identificará a este software, a continuación se muestra esta imagen.

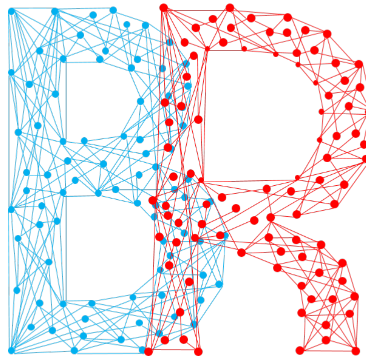


Figura 6.6: Ícono del Software

- Campo de Salida de resultados

  En este campo se mostrarán los resultados obtenidos de aplicar el algoritmo BR, es decir, se mostrarán por pantalla las intersecciones bicromáticas de segmentos de los conjuntos (Grafos) puestos. Los resultados serán mostrados en orden para lograr ser completamente entendibles en el espacio asignado.

- Campo de Dibujo

  El campo de dibujo está destinado a ser el espacio en donde los usuarios puedan crear sus propios ejemplos para experimentar casos específicos de grafos. Sin embargo, por temas de visibilidad de los grafos, se limitará la creación de cada grafo a diez vértices (10 vértices en total, considerando el grafo azul y rojo).

- Íconos de Dibujo

  En este espacio se ubicarán los íconos que tengan acciones sobre la creación de grafos manualmente, es decir, las acciones como “Añadir vértice” (para grafos de ambos colores) y “Añadir Arista” (para grafos de ambos colores). Se diferenciarán los íconos por color y existirán restricciones para relacionar vértices de un color con otro.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

El sistema tiene un menú con acciones muy independientes, por lo que no es una jerarquía compleja, esto se produce porque la funcionalidad del software es muy específica en sus requerimientos funcionales (ver Diagrama de la imagen 6.7).

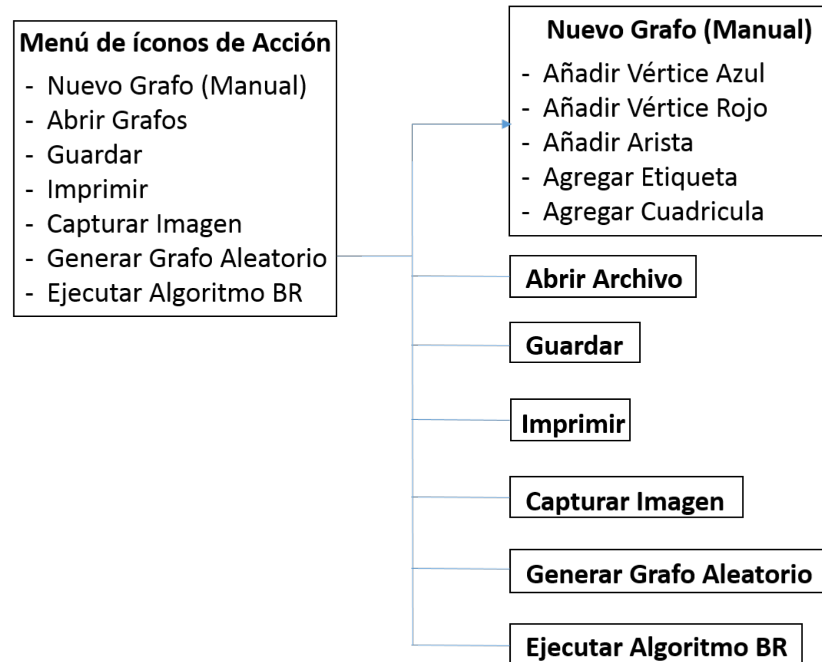


Figura 6.7: Diagrama de Jerarquía

En este diagrama de jerarquía de menú, se puede ver la anidación de las acciones existentes en el software. Esta anidación se hace debido a la restricción de crear un grafo con interacciones del usuario con el ejemplo del conjunto de datos usados en el software.

Puede verse que el comportamiento de la interfaz es estática y no abre paso a otras pantallas o interfaces, por ello no es necesario diseñar un mapa de navegación.

El software es una herramienta específica, y por esta razón, la interfaz diseñada es específica, con pocos elementos y totalmente concebida para uso experimental.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Capítulo 7

Construcción de BR-Software

7.1. Introducción

Para el desarrollo del proyecto es necesario contar con algunas exigencias técnicas. Estas exigencias técnicas deben evaluarse con respecto a la realidad del proyecto.

Para lograr el desarrollo es necesario un equipo de características estándar normales, ya que el desarrollo del software se realizará en el lenguaje de programación Java, el cual no necesita requerimientos elevados de hardware y su licencia es gratuita (es necesario instalar el framework de java jdk).

La programación en java requiere de un software editor, para ello se usará Netbeans IDE, software libre y de licencia de desarrollo gratuita, con un uso intuitivo y simple. Para enfrentar el proyecto, el equipo de desarrollo cuenta con experiencia en programación en lenguaje Java, conocimiento de estructuras de datos avanzada, programación orientada a objetos y conocimiento en la teoría de Geometría computacional.

El desarrollo del algoritmo BR implica también la construcción de algoritmos de aplicación específica en temas de geometría computacional, que se describen en las siguientes secciones de este punto.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

7.2. Orden Rotacional

Un Orden Rotacional es una lista de tangentes, obtenidas en la sección **??**, que contiene el orden por el cual se puede girar sobre un polígono convexo. Este ordenamiento puede basarse en distintos criterios; el criterio utilizado para la implementación del cáliper rotatorio actual, está basado en pendientes, obtenidas con los vértices que forman un conjunto Q externo y los vértices del polígono convexo P.

Por consiguiente, se necesita un algoritmo que pueda entregar esta solución de los puntos tangentes; el algoritmo que se utilizará será el presentado en (Devadoss y O'Rourke, 2011) originalmente propuesto para solucionar en tiempo $O(n^2)$ para un vértice distinto al del polígono convexo (ver Figura 7.1).

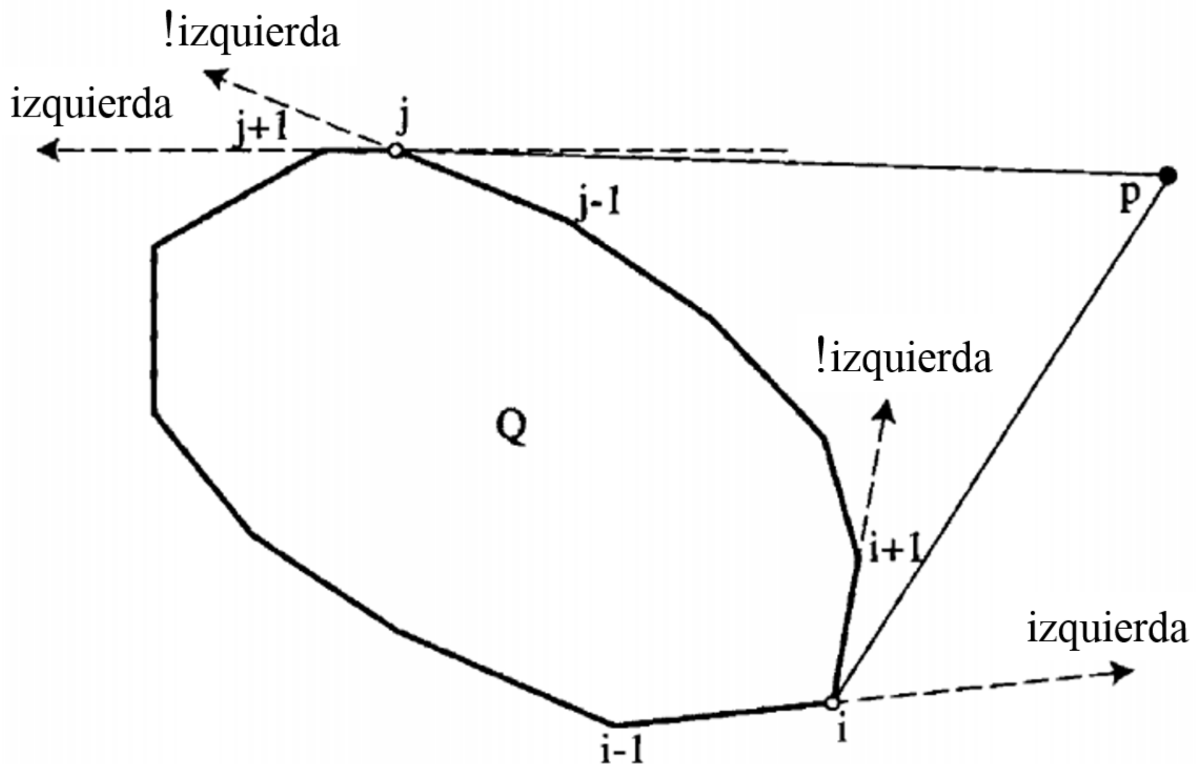


Figura 7.1: Vértices de Tangencia de p sobre P (Devadoss y O'Rourke, 2011)

Sin embargo, O'Rourke hace referencia a que el tiempo de ejecución puede ser reducido

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Algoritmo 3 Vértices Tangentes (Ver (Devadoss y O'Rourke, 2011))

Entrada: El Polígono Convexo $P = \{p_0, p_2, \ldots, p_{n-1}\}$ y un punto p externo a P.
Salida: Los Dos Puntos Tangentes de p sobre el Polígono convexo P.

1: **para** $i = 0$ hasta $n - 1$ **hacer**
2:    **si** $Xor(p$ está en la Izquierda o en (p_{i-1}, p_i), p está en la izquierda o en $(p_i, p_{i+1}))$ **entonces**
3:       p_i es vértice de Tangencia
4:    **fin si**
5: **fin para**

a un tiempo $O(nlogn)$ si se cumple que los vértices a evaluar sean extrictamente externos al polígono convexo, es decir, que es posible implementar una solución más efectiva considerando que únicamente los vértices son externos y conocemos su ubicación, por lo que se necesitaría comparar su ubicación con la ubicación de los vértices del polígono P.

Es por ello, que es necesario ordenar los vértices del polígono convexo ascendentemente por la coordenada X previa a la aplicación del algoritmo sin perder el orden original en el que se encuentran en el polígono (ya que el algoritmo depende de la posición de los vértices en el polígono), este procedimiento de preprocesado tiene una complejidad de $O(nlogn)$. De esta forma, a través del promedio de las coordenadas X de los vértices, se obtiene los vértices que se encuentran en la mitad del polígono, así, es posible ver si el vértice externo a consultar comenzará desde el inicio de la lista de vértices ordenados o, en su defecto por el final y retrocediendo.

Esta pequeña modificación mejora la búsqueda de los vértices de tangencia (ver Algoritmo **??**), ya que evita la consulta de vértices más alejados al punto externo p en tiempo $O(nlogn)$. Sin embargo, en el peor de los casos, es posible que la búsqueda se extienda por todos los vértices del polígono convexo P y con una complejidad de $O(n^2)$.

De esta forma, el algoritmo cambia a esta estructura, mejorando la complejidad. El nuevo algoritmo se puede ver en el algoritmo 4.

Esto entrega los vértices tangentes izquierdos y derechos (horarios o antihorarios) de cada vértice externo al polígono convexo y, además, sus respectivas pendientes.

Iterando este procedimiento por el total de puntos externos es posible obtener el listado de los vértices de tangencia izquierdos y derechos (Ver Tabla 7.1). Una vez obtenidas estas listas por separado, es necesario mezclarlas y formar un listado total de los puntos externos

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Algoritmo 4 Vértices Tangentes Modificado Con vértices de P Ordenados por el eje X

Entrada: El Polígono Convexo $P = \{p_0, p_2, \ldots, p_{n-1}\}$, el polígono P ordenado por las coordenadas X dadas por sus vértices y un punto p externo a P.

Salida: Los Dos Puntos Tangentes de p sobre el Polígono convexo P.

```
    si p es mayor a la mitad entonces
2:    para Desde i = 0 hasta n − 1 hacer
        si Xor(p está en la Izquierda o en (p_{i−1}, p_i), p está en la izquierda o en (p_i, p_{i+1}))
        entonces
4:        p_i es vértice de Tangencia
        fin si
6:    fin para
    si no
8:    para Desde i = n − 1 hasta 0 hacer
        si Xor(p está en la Izquierda o en (p_{i−1}, p_i), p está en la izquierda o en (p_i, p_{i+1}))
        entonces
10:       p_i es vértice de Tangencia
        fin si
12:   fin para
    fin si
```

con su vértice de tangencia izquierdo o derecho, y su respectiva pendiente; es necesario realizar esta mezcla, ya que se debe considerar que en el giro del Cáliper Rotatorio, todos los puntos externos deben ser alcanzados, sea por la línea de soporte del Cáliper derecha o la línea de soporte izquierda (ver Figuras 7.5 y 7.6). La pendiente es el dato de relevancia en este punto, ya que en ella se basa el criterio para confeccionar el orden rotacional.

Ya obtenida la lista de tangentes, se debe realizar un ordenamiento de sus respectivas pendientes de la recta, formada por el vértice externo al vértice de tangencia del polígono convexo, de forma ascendente, pero con una variación muy importante. Para que un Orden Rotacional sea útil en el giro de un Cáliper Rotatorio debe partir desde la pendiente horizontal y seguir la trayectoria de giro hasta el inicio, es decir, el orden rotacional debe partir desde la pendiente cero ascendentemente hasta el infinito, luego ascendentemente desde el infinito negativo hasta nuevamente la pendiente más cercana al cero. De este modo, el orden rotacional sigue paulatinamente un giro completo sobre el polígono convexo y es capaz de alcanzar cada vértice externo al polígono.

Es importante mencionar que este orden rotacional permite completar una media vuelta

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

al polígono (es decir, a 180°), por lo que es necesario recorrer dos veces este orden para completar un giro completo sobre el polígono convexo (ver Tabla 7.2).

Finalmente, el ordenamiento rotacional es obtenido en una complejidad de $O(logn)$, lo que aporta a que el cáliper rotatorio sea capaz de seguirlo y alcanzar todos los puntos externos al polígono, realizando esto en una complejidad de tiempo $O(nlogn)$ para el Punto 2 del Algoritmo BR (ver sección **??**).

7.2.1. Caso de Estudio

El siguiente caso de estudio, se presenta para demostrar gráficamente el funcionamiento de este procedimiento.

Sea $P = \{(8,2),(9,7),(8,11),(3,9),(1,6),(3,2)\}$ un polígono convexo en el plano (representado en color rojo) y $Q = \{(2,11),(10,9),(2,4),(11,4),(1,2),(10,2)\}$ el conjunto de puntos externos al polígono convexo (en color azul) representados en la Figura 7.2.

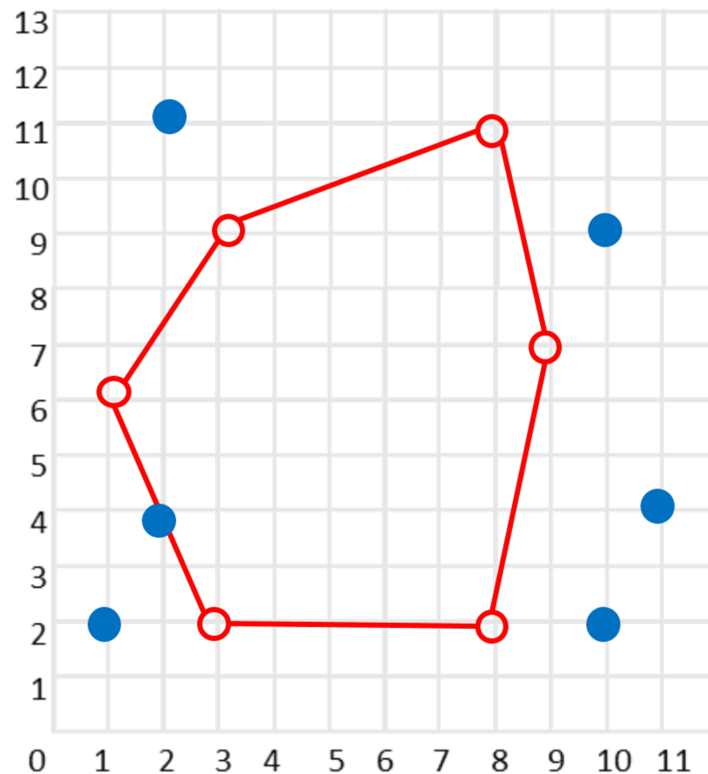


Figura 7.2: Polígono Convexo P y Conjunto de Puntos Externos Q

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Aplicando el algoritmo modificado para las tangentes, es posible obtener todas las tangentes de los puntos externos al Polígono Convexo P (tal como se muestra en la Figura 7.3).

De este procedimiento, contamos con el listado de las tangentes izquierdas y derechas (Ver Tabla 7.1).

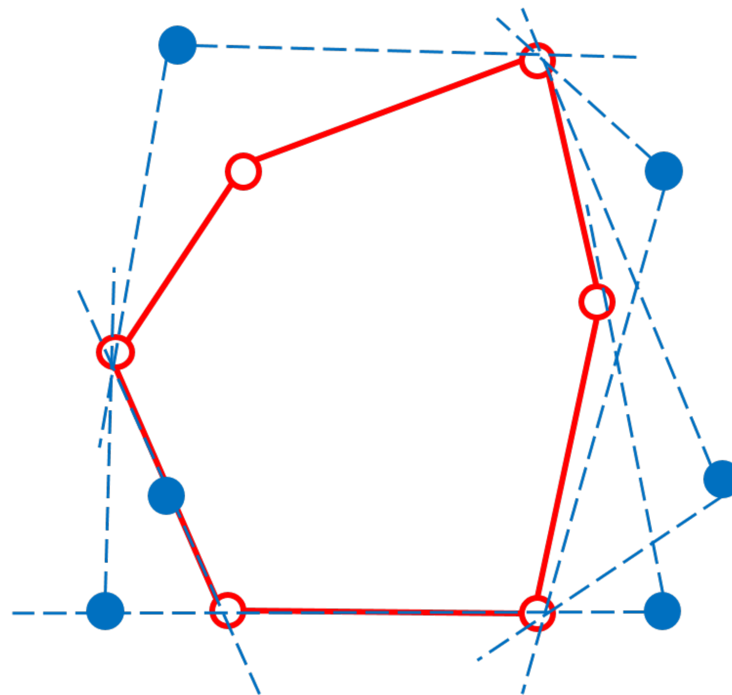


Figura 7.3: Conjuntos de tangentes desde cada punto externo hasta el polígono convexo.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Tangentes Izquierdas			**Tangentes Derechas**		
p	**Tangente izquierda**	**Pendiente** m	p	**Tangente derecha**	**Pendiente** m
(2,11)	(8,11)	0	(2,11)	(1,6)	5
(10,9)	(8,2)	3,5	(10,9)	(8,11)	-1
(11,4)	(8,2)	0,67	(11,4)	(8,11)	-2,33
(10,2)	(3,2)	0	(10,2)	(9,7)	-5
(1,2)	(1,6)	∞	(1,2)	(3,2)	0
(2,4)	(1,6)	-2	(2,4)	(3,2)	-2

Tabla 7.1: Listado de Tangentes Izquierdas y Tangentes Derechos

Como ya se cuenta con ambos listados de tangentes, es posible realizar la mezcla y obtener un listado único, que incluye cada punto externo con su tangente (sea ésta izquierda o derecha), y su respectiva pendiente.

En este paso es necesario realizar el ordenamiento presentado en la sección anterior, iniciando desde la pendiente 0 (cero) hasta ∞ , y luego desde -∞ (infinito negativo) hasta la pendiente más cercana a 0. De esta forma, se obtiene listado del Orden Rotacional (Ver Tabla 7.2).

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Orden Rotacional				
Orden Rotacional	p	**Tangente izquierda**	**Tangente derecha**	**Pendiente** m
1	(2,11)	(8,11)		0
2	(10,2)	(3,2)		0
3	(1,2)		(3,2)	0
4	(11,4)	(8,2)		0,67
5	(10,9)	(8,2)		3,5
6	(2,11)		(1,6)	5
7	(1,2)	(1,6)		∞
8	(10,2)		(9,7)	-5
9	(11,4)		(8,11)	-2,33
10	(2,4)		(3,2)	-2
11	(2,4)	(1,6)		-2
12	(10,9)		(8,11)	-1

Tabla 7.2: Orden Rotacional

Estos resultados pueden ser expresados de forma gráfica (Ver Figura 7.4), cabe destacar que cada pendiente muestra la dirección del punto externo alcanzado y que el ordenamiento es efectuado en sentido antihorario.

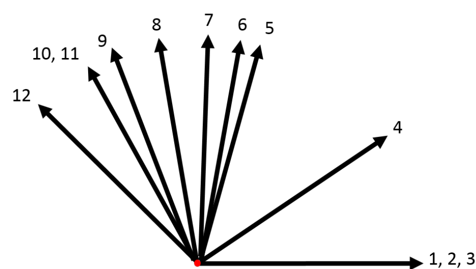


Figura 7.4: Representación del Orden Rotacional en la Tabla 7.2

Al unir la dirección de todas las tangentes de forma ordenada se obtiene una figura en forma de abanico, esta representación está ordenado en sentido antihorario y aclara la forma en que el cáliper rotatorio girará (Ver Figura 7.5). Como se puede apreciar, el orden

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

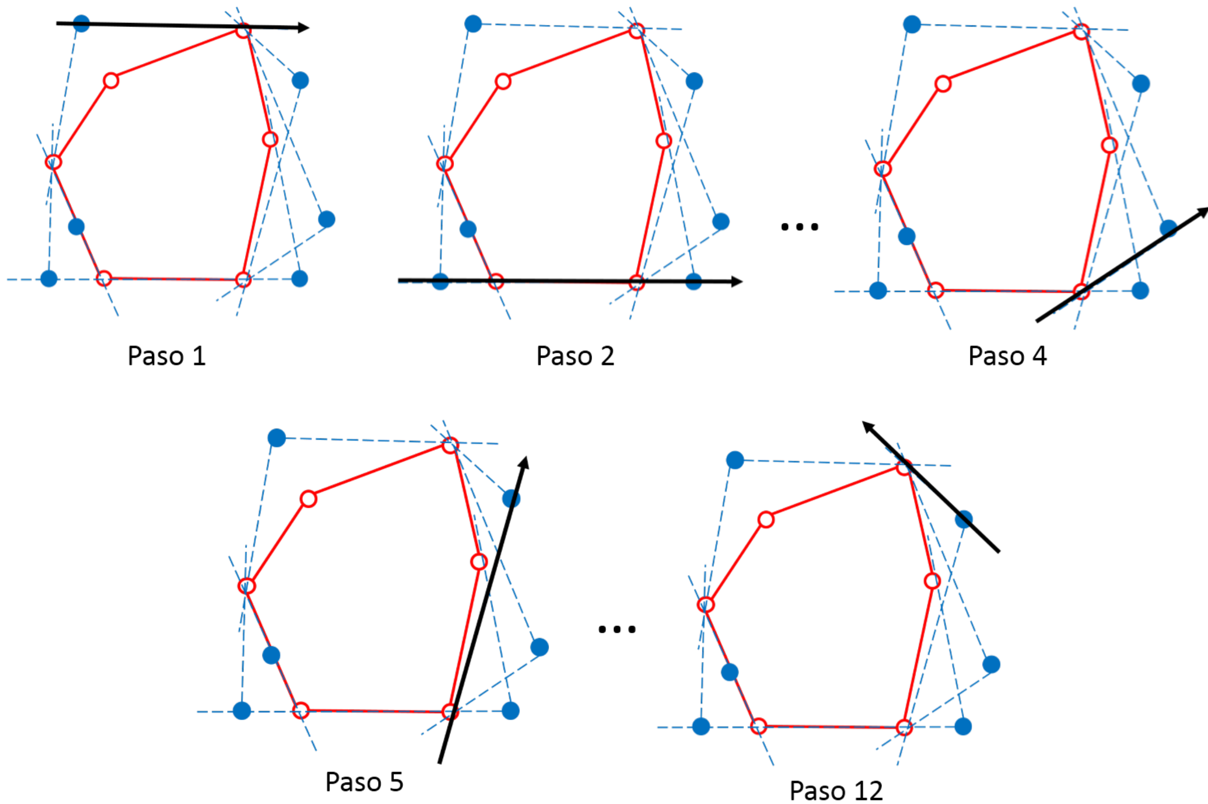


Figura 7.5: Seguimiento de orden rotacional

rotacional puede mostrar la dirección del orden rotacional hasta 180°, lo que permite que dos líneas de soporte paralelas puedan abarcar todo el contorno de un polígono convexo. En la Figura 7.5, se muestra el movimiento de una línea de soporte en cada dirección dada en el Orden Rotacional, cada vértice externo es alcanzado, iniciando desde la pendiente horizontal y girando en sentido antihorario, hasta completar el Orden Rotacional.

Otra cosa destacable, es que los pasos del Orden Rotacional son exactamente el doble de la cantidad de puntos externos presentes. Así, el número de pendientes distintas es menor o igual al número de pasos presentes en el orden rotacional.

Finalmente, es posible aplicar el cáliper rotatorio al polígono convexo P y es posible que este cáliper pueda seguir el orden rotacional obtenido anteriormente. Tal como se muestra en la Figura 7.6, el cáliper puede girar sobre el polígono convexo P alcanzando a cada punto externo ajustándose al polígono en sentido antihorario siguiendo el orden dado

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

por las pendientes de cada punto externo al polígono, sin problemas de pérdida del par antipodal, ya que es parte del movimiento natural del cáliper rotatorio. Además, se puede apreciar la pendiente que ocupan en cada paso (Ver Figura 7.4 y comparar con Figura 7.6).

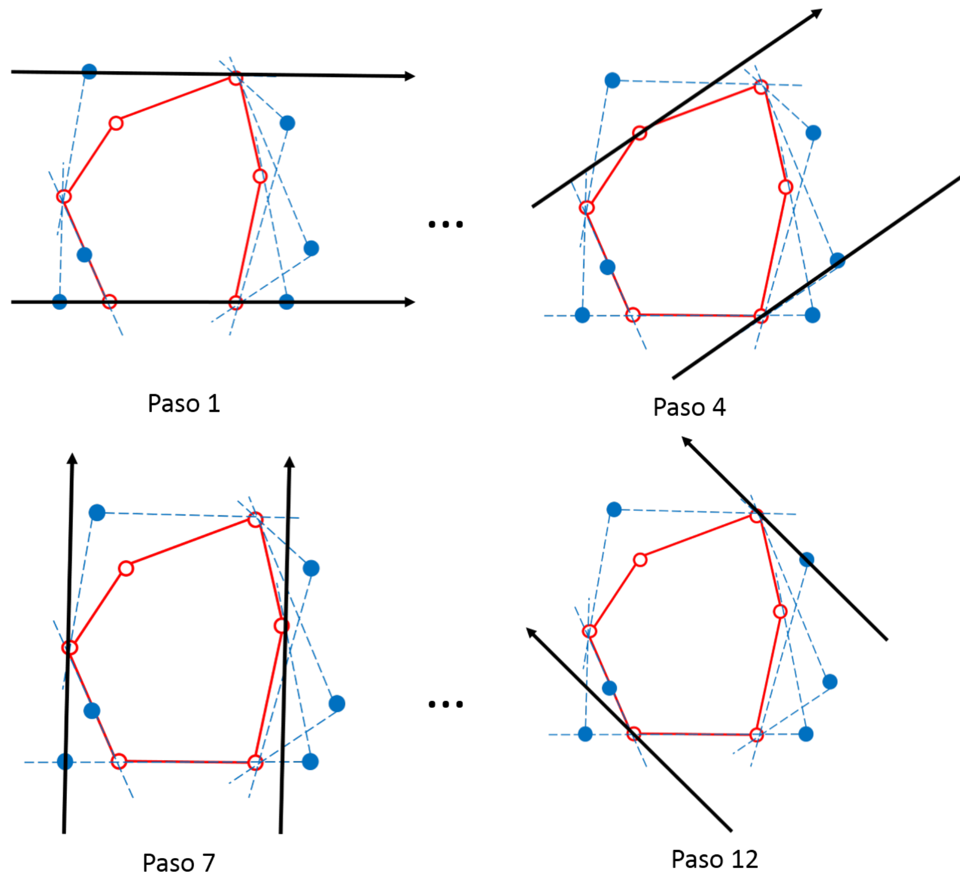


Figura 7.6: Seguimiento de Cáliper Rotatorio

De esta forma, se comprueba que es posible implementar un Cáliper Rotatorio que esté condicionado a seguir un orden rotacional dado por factores externos y que esté basado en las pendientes formadas por líneas tangentes al polígono. La mayor ventaja de efectuar un orden rotacional sobre los puntos externos al conjunto *Be* (Conjunto de los vértices azules externos) y luego aplicar el caliper rotatorio sobre ellos es el significativo ahorro de tiempo, pues se logra alcanzar a cada punto de la lista en un solo paso.

El caliper rotatorio se utiliza en la implementación del algoritmo BR, específicamente

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

en el paso 2, en conjunto con otras operaciones que se describen a continuación. Este procedimiento ofrece un rendimiento de complejidad de tiempo $O(|Q| + nlogn)$, el mayor de ellos es $O(nlogn)$, ya que $|Q|$ es en el peor de los casos de complejidad $O(n)$, donde todo los vértices del Grafo son externos al $CH(R)$, siendo Q el conjunto de los puntos externos; su implementación se probó con un conjunto de datos descritos en el anexo C "Datos de Experimentación", funcionando correctamente para todos ellos.

7.3. Intersecciones Segmento-Segmento

Las intersecciones son problemas muy complicados y costosos para grandes volúmenes de datos, es por ello que el desarrollo de algoritmos que traten estos problemas deben ser muy eficientes en complejidad de tiempo y espacio. Por otro lado, existen diferentes formas de visualizar las intersecciones, pero por temas de simplicidad se basará el desarrollo del trabajo en un problema específico que aporta a la solución: las intersecciones segmento-segmento.

Las intersecciones segmento-segmento representan el problema mínimo en donde dos aristas o segmentos se intersectan, los datos son conocidos (es decir, los vértices que componen a estas aristas son conocidas) y el procedimiento es de $O(1)$.

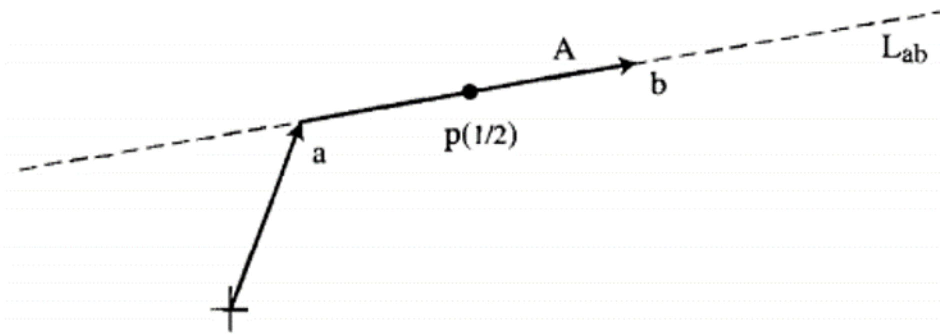


Figura 7.7: Se presenta $p(s) = a + A$; $p(\frac{1}{2}) = a + \frac{1}{2}A$

Según (Devadoss y O'Rourke, 2011) el algoritmo de intersecciones más completo se basa en las propiedades de las matrices que representa el sistema de ecuaciones formado por las aristas a evaluar. El razonamiento se basa en lo siguiente:

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

"Sean dos aristas L_{ab} (con vértices a y b) y L_{cd} (con vértices c y d); y sean $A = b - a$ y $C = d - c$ puntos de vectores a lo largo del segmento. Cualquier punto en la línea L_{ab} pueden ser representados como la suma $p(s) = a + sA$, que toma un punto en L_{ab}, y entonces mover alguna distancia a lo largo de la línea aumentando A por s. La variable s se llama el parámetro de esta ecuación. Se consideran los valores obtenidos por $s = 0$, $s = 1$, y $s = 1/2$; $p(0) = a$, $p(1) = a + A = a + b - a = b$, y $p(1/2) = (a + b)/2$. Esto demuestra que $p(s)$ para $s \in [0, 1]$ representa todos los puntos en el segmento $\overline{ab}$, con el valor de s representando una fracción de la distancia entre vértices; específicamente, los extremos de s producen los vértices."

Similarmente se representa los puntos en el segundo segmento por $q(t) = c + tC$, con $t \in [0, 1]$. Un punto de intersección entre los segmentos se identifica por los valores de s y t que logran $p(s)$ igual a $q(t) : a + sA = c + tC$. Así, se logra la convención de que desde 0 hasta 1 indican las coordenadas x e y, su solución es:

$$s = [a_0(d_1 - c_1) + c_0(a_1 - d_1) + d_0(c_1 - a_1)]/D, \quad (7.1)$$

$$t = [a_0(c_1 - b_1) + b_0(a_1 - c_1) + c_0(b_1 - a_1)]/D, \quad (7.2)$$

$$D = a_0(d_1 - c_1) + b_0(c_1 - d_1) + d_0(b_1 - a_1) + c_0(a_1 - b_1) \quad (7.3)$$

Ecuación 7.3: Sistema de ecuaciones

La división por cero es una posibilidad en estas ecuaciones. El denominador D pasa a ser cero si y solo si dos líneas son paralelas. Algunos segmentos paralelos involucran intersección, y algunas no. Para ello se consultan casos específicos, como el solapamiento colineal que se refiere a que un segmento solapa a otro si y sólo sí un vértice de una de ellas está entre los vértices de la otra. Con esto se verifica si c no es colineal con L_{ab}, entonces los segmentos paralelos L_{ab} y L_{ac} no se intersectan.

Con todos estos criterios y casos especiales se puede determinar si se intersectan dos segmentos, además de obtener el punto de intersección sin problemas.

El algoritmo de detección de intersecciones en base a dos pares de vértices es utilizado en casi todas las secciones del Algoritmo BR, por ello es importante para la obtención de la solución. El procedimiento es usado en el paso 2 y se aplica en cada giro del caliper rotatorio, ya que debe consultar si un segmento formado por dos vértices azules intersectan

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

al segmento formado por el par antipodal (ver Sección **??**). Luego, también es usado en el paso 3, donde nuevamente es utilizado para consultar si un segmento formado por vértices rojos intersecta al par antipodal del caliper sobre un segmento azul (ver sección 4.0.5). Y Finalmente, tambien es usado en el paso 4, donde es utilizado para consultar si un segmento azul intersecta alguna arista de algun región convexa (ver sección 4.0.6).

7.4. Partición de Polígonos Convexos

Uno de los pasos importantes para lograr completar el algoritmo BR, que encuentra las intersecciones bicromáticas de segmentos, es lograr la partición del polígono convexo formado al obtener el cierre convexo del conjunto de puntos rojos o azules, según sea el caso. Este procedimiento corresponde al aplicado en el punto 3.b.ii.C (Ver algoritmo 4.0.1) en el caso de encontrarse un segmento rojo que intersecte al segmento azul b_jb_k (Ver subsección 4.0.5).

Sea $V(G_i) = \{b_1, ..., b_m\}$ el conjunto de puntos azules interiores a $CH(R)$. Considerar la relación de equivalencia $b_j \sim b_k$ sí y sólo sí el segmento azul b_jb_k no cruza segmento rojo alguno. En este caso, se dice que b_j y b_k están *relacionados*. En la etapa inicial, se toma el punto azul b_1, haciendo $C_1(b_1) := CH(R)$. En la etapa k, $k \leq m$, se introduce el punto azul b_k. Ahora se considera la región convexa actual $k-1$, con $1 \leq k-1 \leq m$. Las regiones convexas pueden ser de dos tipos (Cortés et al., 2012):

(1) Regiones convexas que contienen todos los puntos azules b_s de $V(G_i)$, con $1 \leq s \leq k-1$, tal que todos están relacionados entre ellos. Cada región es asociada a uno de sus puntos azules, llamado b_j, como su representativo, y se denota por el tipo $C_{k-1}(b_j)$.

(2) Regiones convexas que no contienen puntos azules b_s de $V(G_i)$ con $1 \leq s \leq k-1$, denotado por el tipo $C_{k-1}(\phi)$.

En la etapa k, el siguiente punto azul b_k es ubicado en una región convexa de cualquiera de los dos tipos descritos anteriormente, la cual es dividida en, a lo más, cuatro nuevas regiones convexas como sigue, distinguiéndose tres casos (Cortés et al., 2012):

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Caso 1: $b_k \in C_{k-1}(\phi) \rightarrow C_{k-1}(\phi) := C_k(b_k)$
Caso 2: $b_k \in C_{k-1}(b_j)$ y $b_k \sim b_j \rightarrow C_{k-1}(b_j) := C_k(b_j)$
Caso 3: $b_k \in C_{k-1}(b_j)$ y $b_k \nsim b_j$, seguir como se explica a continuación:

- Considerar un segmento rojo S_{jk} intersectando a $b_j b_k$ y la línea ℓ_{jk} conteniendo a S_{jk}
- Asumir que S_{jk} está estrictamente contenido en $C_{k-1}(b_j)$ (sus dos vértices rojos están en el interior de $C_{k-1}(b_j)$). (Ver Figura 7.8)
- ℓ_{jk} intersecta a $C_{k-1}(b_j)$ en dos aristas, ya que los puntos están en posición general.
- Unir los vértices de S_{jk} con los vértices de estas aristas.
- Por lo tanto, $C_{k-1}(b_j)$ es dividida en cuatro regiones convexas:
    (a) $C_k(b_k)$, que podría contener puntos azules b_s con $s \in \{k+1, ..., m\}$ pero no puntos azules b_s con $s \in \{1, ..., k-1\}$.
    (b) $C_k(b_j)$, conteniendo los puntos azules b_s con $s \in \{1, ..., k-1\}$, de modo que $b_s \sim b_j$.
    (c) Dos regiones del tipo $C_k(\phi)$.

Las regiones convexas restantes son actualizadas obteniendo más regiones convexas de tipo $C_k(\phi) = C_{k-1}(\phi)$, y regiones convexas de tipo $C_{k-1}(b_s)$ son actualizadas como $C_k(b_j)$ con $s \in \{1, ..., k-1\}$ y $b_k \nsim b_j$. Notar que en este caso los puntos azules b_s, b_k y b_j son ubicados en regiones convexas diferentes, ya que los segmentos azules dados por estos puntos son intersectados por segmentos rojos.

Suponer ahora que el segmento S_{jk} está contenido en $C_{k-1}(b_j)$ y uno o sus dos vértices pertenecen a $C_{k-1}(b_j)$ (Están en uno de los bordes de la región). Se procede como anteriormente, dividiendo $C_{k-1}(b_j)$ en dos o tres regiones convexas respectivamente, faltando a lo más las dos regiones de tipo $C_k(\phi)$ (Ver figuras 7.9 y 7.10).

Finalmente, asumir que el segmento S_{jk} no está contenido en $C_{k-1}(b_j)$, es decir, al menos un vértice de S_{jk} está ubicado en una región convexa diferente. Ya que $C_{k-1}(b_j)$ es la región convexa a dividir, se procede análogamente pero considerando, mediante el uso de un punto artificial, la parte del segmento contenida en la región convexa a dividir (Ver Figura 7.11). Dicho punto no será tomado en cuenta en las siguientes etapas y no provocará cambios en la complejidad de tiempo del algoritmo.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

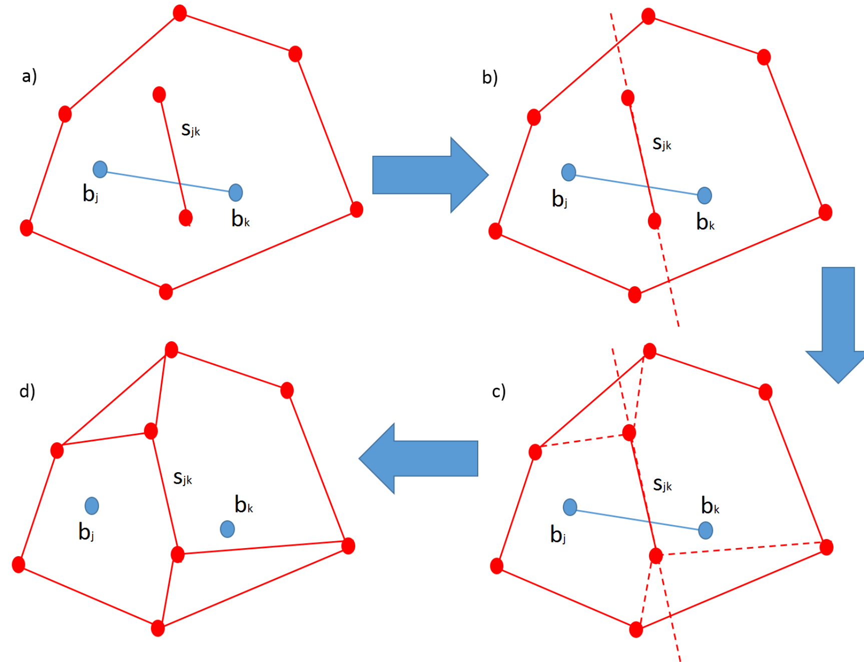


Figura 7.8: S_{jk} está estricatamente contenido en $C_{k-1}(b_j)$

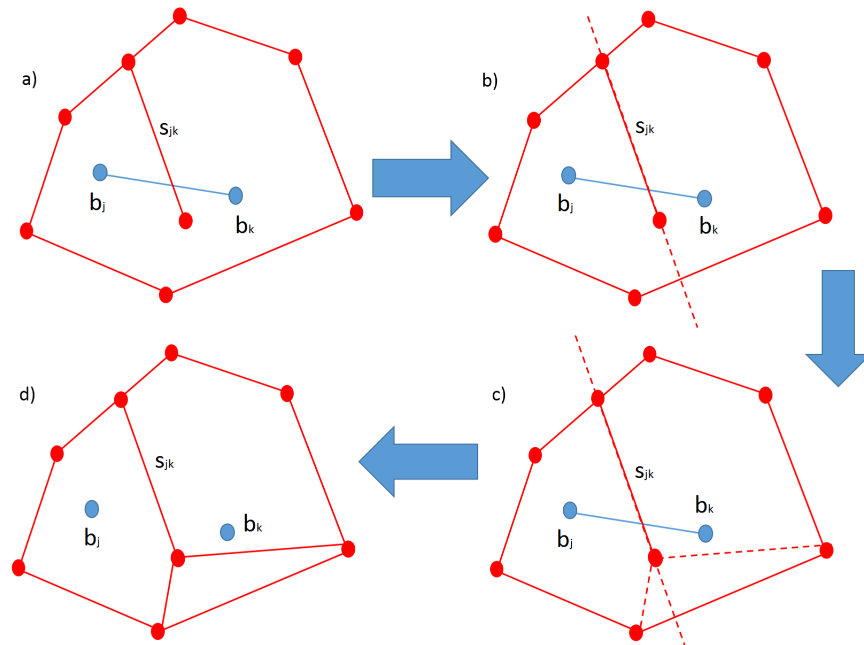


Figura 7.9: S_{jk} está contenido en $C_{k-1}(b_j)$ y uno de sus vértices pertenece a $C_{k-1}(b_j)$.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

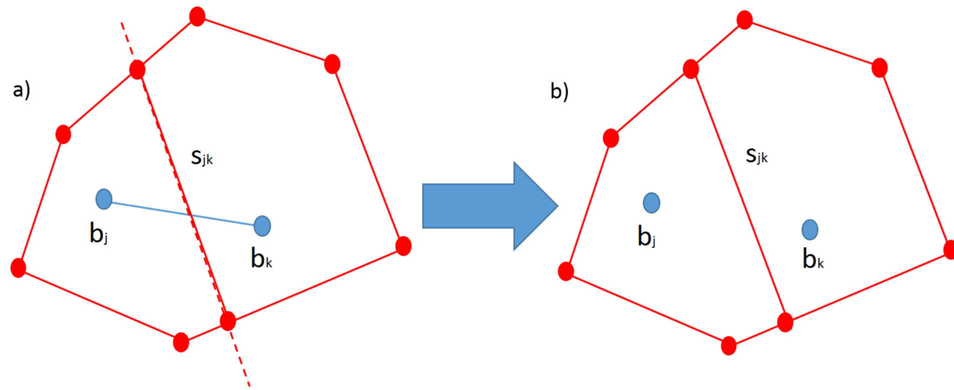


Figura 7.10: S_{jk} está contenido en $C_{k-1}(b_j)$ y sus dos vértices pertenecen a $C_{k-1}(b_j)$.

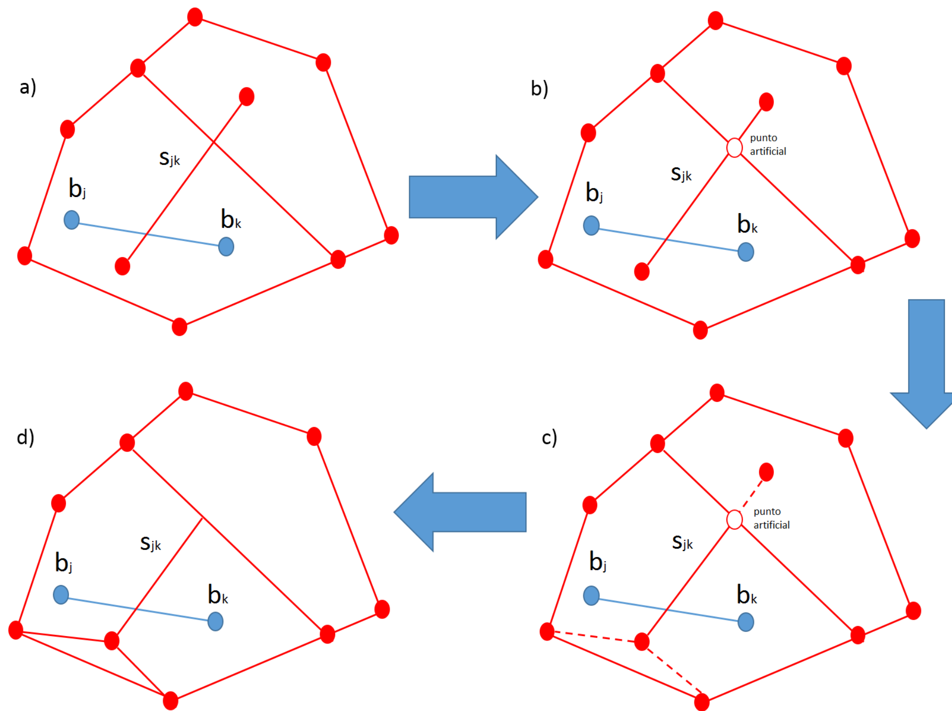


Figura 7.11: S_{jk} no está contenido en $C_{k-1}(b_j)$

Este paso es de suma importancia para hallar, en primera instancia, qué segmentos azules internos al cierre convexo rojo ($CH(R)$) se intersectan con algún segmento rojo y, luego, de manera análoga, qué segmentos rojos internos al cierre convexo azul ($CH(B)$) se intersectan con algún segmento azul.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

7.5. Ubicación de un Punto en un Polígono

Este problema es considerado el caso básico de la Ubicación planar de un punto en un espacio. Se considera un poligono (convexo o no Convexo) y un vértice en el mismo espacio planar, se espera conocer si el vértice está dentro del polígono o fuera de éste.

Un algoritmo capaz de decidir si un vértice dado está en el interior o en el exterior de un polígono es el algoritmo propuesto por Eric Haines en (Haines, 1989), este presenta una solución natural e intuitiva al problema y está basada en la curva de Jordan.

La solución que plantea este algoritmo se obtiene extendiendo una recta imaginaria hacia uno de los extremos del eje x (es decir, al infinito positivo o negativo). Con esta recta se realiza un cálculo que consulta por todas las aristas del polígono si existe o no una intersección y se va contando la cantidad de veces que se intersecta la recta imaginaria con las aristas del polígono.

Si la cantidad de intersecciones es par, entonces el vértice se encuentra fuera del poligono; por otro lado, si la cantidad de intersecciones es impar entonces el vértice se encuentra dentro del polígono (ver Figura 7.12).

Este procedimiento no es exclusivo de un polígono convexo, es decir, es aplicable a cualquier tipo de polígono, y el tiempo de ejecución es igual para todos ellos, con una complejidad de $O(n)$.

Algoritmo 5 Algoritmo de Localización de un punto Interno o Externo a un Polígono

Entrada: Polígono P, un vértice v
Salida: Verdadero si esta dentro o Falso si es que está fuera del P.

1: Contador=0
2: Suponer una Recta desde v hasta el ∞
3: **para** Cada Arista del Polígono **hacer**
4:     **si** la arista de P intersecta a la recta creada **entonces**
5:         Aumentar en 1 el Contador
6:     **fin si**
7: **fin para**
8: **si** Contador es Par **entonces**
9:     Falso
10: **si no**
11:     Verdadero
12: **fin si**

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

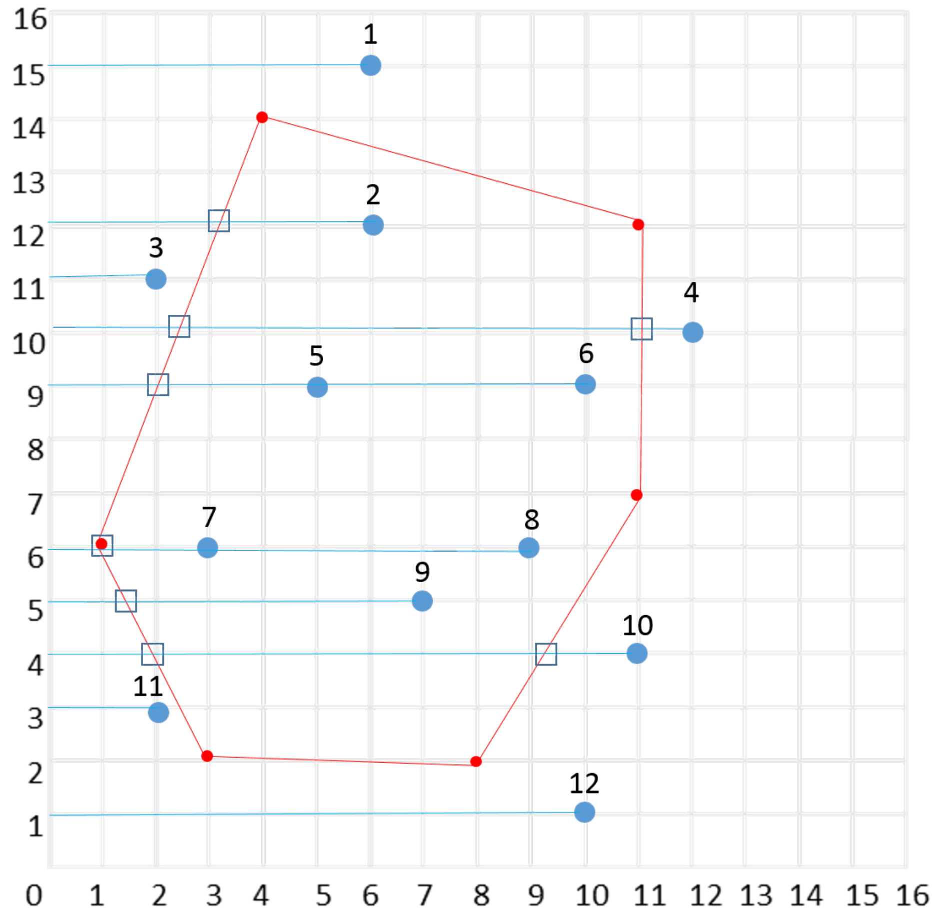


Figura 7.12: Algoritmo de identificación de vértices externos e internos a un polígono (Haines, 1989)

En la imagen 7.12 es posible ver cómo el vértice azul (1) tiene cero intersecciones (que se considera par), por lo que se encuentra fuera del polígono; en cambio, el vértice (2) tiene una intersección (una intersección es impar), por lo que se considera dentro del polígono. Siguiendo la lógica, el vértices (3) se encuentra fuera, el vértice (4) está fuera, el vértice (5) está dentro y el vértice (6) se encuentra dentro del polígono también.

Este procedimiento va consultando hasta la totalidad de los vértices azules. El algoritmo 5 detalla el procedimiento descrito en la Figura 7.12.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

7.6. Localización de puntos en un polígono convexo dividido en regiones convexas

Para poder llevar a cabo las particiones de la parte 3 del algoritmo BR, es necesario ubicar en cuál de todas las regiones existentes en CH(R) se encuentra el punto azul a ingresar (Ver subsección 7.4). El algoritmo BR (Algoritmo 4.0.1) en el Capítulo 4, en su tercera parte indica que para encontrar las intersecciones bicromáticas interiores a $CH(R)$ se debe realizar un particionamiento de este polígono en regiones convexas que contengan subgrafos completos sin intersecciones con segmentos rojos. Es así como en primer lugar se necesita crear una primera región convexa, correspondiente a $CH(R)$, a partir de la cual se comienza la partición, mediante la inclusión de los puntos azules uno a uno (uno por iteración), dejando solo el primer punto b_1 dentro de la primera región convexa.

En los pasos siguientes, para cada punto azul perteneciente a $V(G_i)$ (que está dentro de $CH(R)$), es necesario descubrir en cuál de las regiones convexas existentes (en un principio la única que existe es $CH(R)$) se ubicaría el nuevo punto azul (de acuerdo a sus coordenadas), para luego realizar la partición descrita en la subseción 4.0.5 y la sección 7.4, .

Teniendo una clase *regionConvexa* representando a cada polígono convexo resultante de la partición, la idea es que *dado un punto b_j, se recorran las particiones consultando si el punto ingresado pertenece ahí.* Así, para cada región, se consultará si el punto está contenido en ella.

En la implementación del algoritmo BR se utiliza un método que nace de manera natural y que se detalla en el algoritmo 6.

Algoritmo 6 Algoritmo de Localización de un punto en un polígono particionado

Entrada: Listado de regiones convexas, un vértice b_j
Salida: Región convexa que contiene a b_j

1: **para** Cada Región Convexa del Listado **hacer**
2:     **si** b_j está contenido en la región actual **entonces**
3:         **devolver** Región Convexa
4:     **fin si**
5: **fin para**

El algoritmo 6 es de complejidad $O(n^2)$, ya que el recorrido por el listado de regiones

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

convexas es de orden lineal $O(n)$ y por cada iteración se aplica el algoritmo 5, que también es $O(n)$. El algoritmo 6 puede ser mejorado en complejidad si se utiliza un método más eficiente para saber si un punto está contenido en un polígono determinado.

En (Cortés et al., 2012), se sugiere realizar esta parte del algoritmo de una manera diferente. Al notar que la partición del polígono convexo $CH(R)$ es una subdivión planar de complejidad lineal, donde las celdas son regiones convexas, y según la necesidad de actualizar la subdivisión a medida que se va dividiendo el poligono, ellos sugieren utilizar una estructura de datos lineal-espacial como la propuesta por (Arge et al., 2006).

El problema de la subdivisión planar y de ubicación de un punto en el plano es fundamental en la geometría computacional. Puede definirse como el preprocesamiento de una subdivisión poligonal del plano en alguna estructura de datos, de tal manera que sea posible consultar de manera eficiente en qué polígono del plano cae el punto de consulta (Arya et al., 2000).

Cada etapa de particionado planar produce una subdivisión del plano que tiene un reparto regular y acelera así las consultas. Por ejemplo, una estructura de datos muy simple, y rápida, podría ser una que permita subdividir el plano en una cuadrícula, de tal manera que las celdas son lo suficientemente pequeñas para representar todo el detalle inherente en el mapa poligonal. Se trata básicamente de cómo se almacenan los *rastermaps* y las consultas de ubicación de puntos se pueden realizar en un tiempo constante; sin embargo, los requisitos de almacenamiento son tan altos que hacen que este enfoque no sea factible. La idea de partición plana es crear una estructura de datos que proporcione los tiempos de consulta rápida mientras que no aumenta los requisitos de almacenamiento para los datos poligonales.

(de Berg et al., 1997) han indicado que el almacenamiento y el tiempo de búsqueda óptimos para el problema es $O(n)$ almacenamiento y $O(logn)$ tiempo para la búsqueda.

Para intentar cumplir a cabalidad lo indicado por (Cortés et al., 2012) al proponer el algoritmo BR, se procedió a revisar (Arge et al., 2006) y, en el transcurso de la revisión, se hizo necesario recurrir a ocho (8) de sus referencias (Baumgarten et al., 1992; Cheng y Janardan, 1992; Driscoll et al., 1986; Imai y Asano, 1987; Mehlhorn, 1984; Mortensen et al., 2005; Sarnak y Tarjan, 1986; Willard, 1992) para intentar entender del todo cómo funciona el algoritmo propuesto, el cual trabaja con estructuras de datos avanzadas como *segment trees* y algoritmos de *fractional cascading*, para entregar mejoras a trabajos anteriores.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Otros métodos estudiados describen enfoques diferentes que han sido capaces de lograr con éxito el óptimo indicado por (de Berg et al., 1997): el método de triangulación de (Kirkpatrick, 1981) y el método de construcción aleatoria de mapas trapezoidales basado en el trabajo de (Mulmuley, 1988; Seidel, 1991). En (O'Rourke, 1998) también se expone uno de estos métodos.

La idea básica de todos estos métodos es lograr un preprocesamiento que entregue una estructura de tipo árbol, con información de coordenadas de vértices y segmentos (Ver Figura 7.14), que permita realizar una búsqueda binaria y cumplir con el $O(logn)$ tiempo.

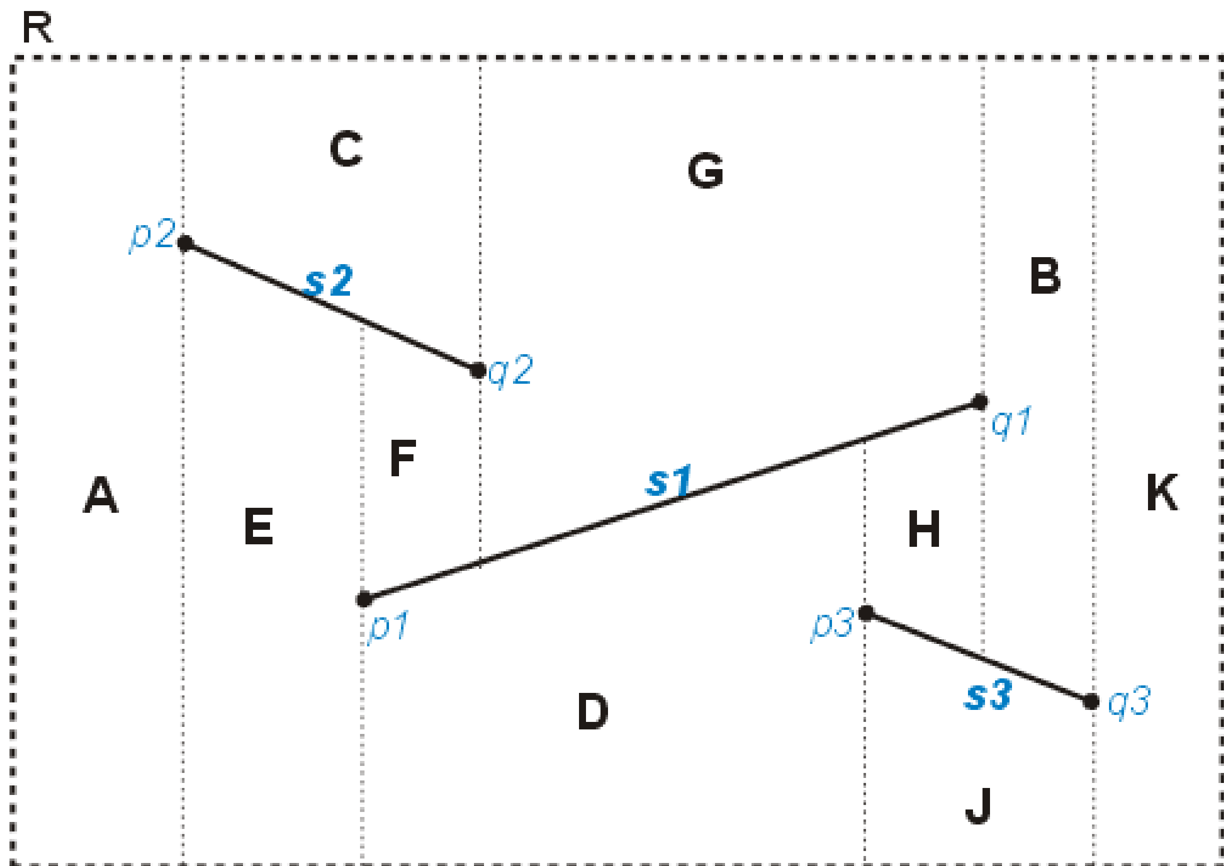


Figura 7.13: Mapa trapezoidal generado por tres segmentos s_1, s_2 y s_3, basado en (de Berg et al., 1997)

En la Figura 7.14 se observa el árbol generado a partir del mapa trapezoidal de la Figura 7.13. Las hojas del árbol corresponden a los trapezoides del mapa.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

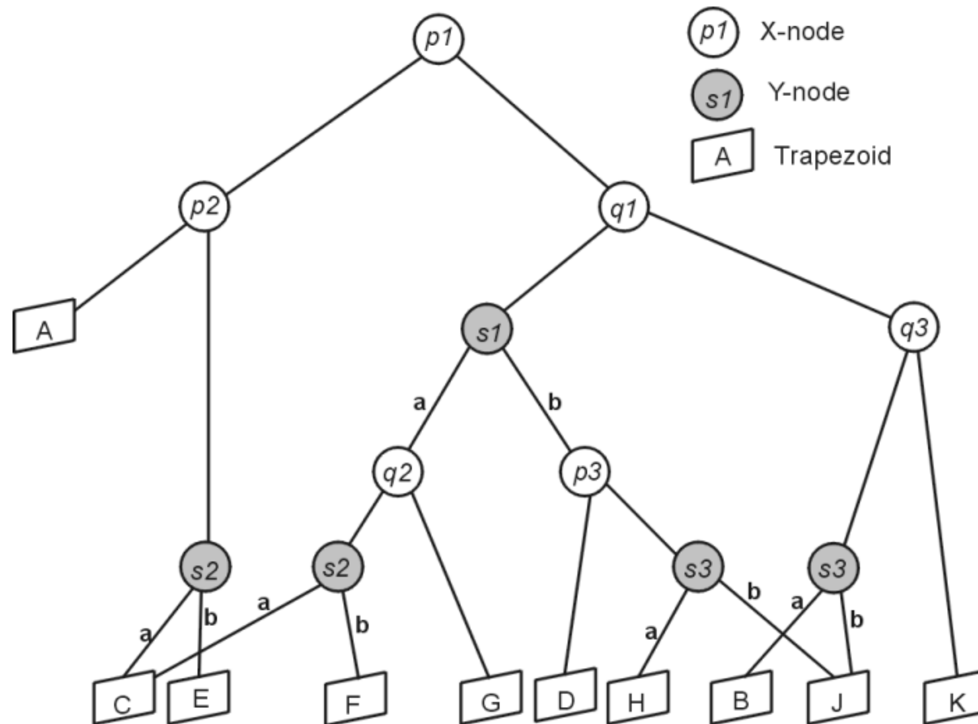


Figura 7.14: Ejemplo de árbol de búsqueda para ubicación de un punto en una subdivisión planar, basado en (de Berg et al., 1997)

Lamentablemente, y de manera inesperada, todo el tiempo invertido en el estudio y resivión de material bibliográfico del área no lograron dar frutos y no se llegó a comprender completamente, dentro de los plazos aceptables para la realización de un Proyecto de Título, los procedimientos a realizar para llegar a construir el árbol de búsqueda que permitiría encontrar la región convexa a la cual pertenece el punto b_j que se está incluyendo en la partición, requiriéndose mucho más tiempo de estudio y experimentación para poder llegar a implementar alguno de los métodos de *Dynamic Planar Point Location* mencionados anteriormente.

7.7. Desarrollo del Algoritmo BR

Al unir todas las partes explicadas del algoritmo BR, solo resta realizar un análisis de la complejidad que resulta de la implementación.

La parte 1 consiste en encontrar, en primer lugar, el cierre convexo de los puntos rojos,

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

lo cual se realiza con el algoritmo de Graham (versión 2) en $O(nlogn)$, según lo indicado en la sección 2.3. Luego, para encontrar $V(G_i)$ y $V(G_e)$, se utiliza el algoritmo descrito en la sección 7.5, el cual es $O(n)$. Considerando que se tienen m puntos azules y que por cada uno de ellos se debe consultar si está dentro o fuera de $CH(R)$, se tiene que el hacer esta separación de puntos es de orden cuadrático $O(nm) = O(n^2)$. Así, al sumar las complejidades de ambas etapas, que además son secuenciales, se tiene $O(nlogn)+O(n^2) = O(n^2)$.

La parte 2 comprende dos subprocesos; el primero de ellos, es la obtención del ordenamiento rotacional de los puntos azules externos a $CH(R)$ con respecto a este mismo, y luego, la aplicación de un cáliper rotatorio sobre $CH(R)$, todo esto con el objetivo de obtener la solución $E(\overline{Ge})$.

El orden rotacional está implementado en base al algoritmo de O'rourke (Devadoss y O'Rourke, 2011), modificado en la sección 2.4, que tiene una complejidad de $O(logn)$. Este algoritmo se ejecuta por cada vértice en Be, por lo que la complejidad total del primer punto es de $O(nlogn)$. Luego, el cáliper rotatorio realiza un giro completo sobre $CH(R)$, realizando la búsqueda de los vértices en Be que correspondan al lema 2 de (Cortés et al., 2012) (ver Figura 4.5), esto quiere decir que el procedimiento recorre dos veces el orden rotacional para lograr un giro de 360°, lo que significa que la complejidad de este paso es de $O(2n) = O(n)$; en cada giro, el procedimiento consulta por cada vértice de Be que está dentro de los límites del Cáliper, decidiendo si continúa dentro o fuera de él. Esta consulta nunca llega a ser por la totalidad de los vértices, por lo que es un orden $O(nlogn)$. Sin embargo, la complejidad depende de la cantidad de segmentos en la solución, de tal forma que la complejidad real es $O(|E(\overline{Ge})|+nlogn)$, lo que en un caso extremo, se podría llegar a tener una complejidad $O(n^2)$.

La tercera parte del algoritmo BR pide, en un principio, construir un arreglo dual de líneas, que posteriormente será útil para obtener el orden rotacional de todos los puntos rojos con respecto a cada punto azul perteneciente al segmento que se está verificando en la iteración actual. En este procedimiento basta con encontrar dicho orden rotacional, para lo cual se crea una lista de rectas duales azules (que en realidad son puntos azules) y para cada una de ellas, se procede a ordenar todas las rectas duales rojas de acuerdo al punto en que se intersectan con la recta azul. Lo anterior es posible tras valerse de las propiedades de la dualidad entre puntos y rectas. Este procedimiento es realizado en orden

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

cuadrático, ya que si se tienen m puntos azules interiores a $CH(R)$ y n puntos rojos en total, para cada punto azul interior a $CH(R)$ ($O(m)$) se recorren todos los puntos rojos ($O(n)$); por lo que en total se tiene $O(mn) = O(n^2)$. En $O(n)$, se contruye la primera región convexa, correspondiente a $CH(R)$, y se le ingresa el primer punto azul de la lista de puntos pertenecientes a $V(G_i)$. Luego, para cada uno de los m puntos azules interiores a $CH(R)$ se realizan las siguientes operaciones:

- Encontrar la región convexa a la que pertenece el punto, lo que de acuerdo a la sección 7.6, es realizado en $O(n^2)$.

- Si la región encontrada es vacía, ingresar el punto y actualizar región. Esto se realiza en $O(1)$. En caso contrario, se genera un segmento azul entre el punto entrante y el punto ”representante” que ya se encontraba en la región y se realiza lo siguiente:

    - Obtener el orden rotacional de cada uno de los dos puntos azules, desde las rectas duales. Cada punto azul es encontrado directamente y en $O(1)$. Cada orden rotacional se agrega a una lista en $O(n)$ y en la misma complejidad se mezclan ambos ordenes rotacionales.

    - Se aplica un cáliper rotatorio sobre el segmento formado por los dos puntos azules, para determinar si existe algún segmento rojo que lo intersecte. Esto se realiza en $O(n)$ (Ver sección 2.4).

    - Si se encuentra intersección, tomar el segmento rojo que intersecta al azul, nombrarlo s_{jk} y realizar la partición de la región, según procedimiento indicado en la sección 7.4, lo cual es realizado en $O(n)$ según lo indicado en la misma sección. En caso de no haber intersección, se agrega el punto azul a la región, sin cambiar el representante. Esto último se realiza en $O(1)$. Así, en el peor de los casos, este item tiene una complejidad total de $O(n)$.

- Se actualiza la región a la que pertenecen los puntos azules afectados. Esto se realiza en orden constante $O(1)$

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Una vez dividida la región convexa, y ya con todos los puntos azules dentro de su respectiva región, además de una lista de aristas azules que no intersectan aristas rojas, se procede a revisar todas las aristas azules internas para obtener el complemento de $E(G_i)$, que es la solución buscada para el paso 3. Esto se realiza en $O(n^2)$.

En la cuarta parte del algoritmo, donde se busca $\{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$, se realiza un preprocesamiento de los datos correspondientes a las aristas azules. Se recorren todos los puntos azules externos y, por cada uno de ellos, se consultan todos los vértices azules internos a $CH(R)$ y se verifica que sean aristas de B. Este proceso toma $O(n^2)$ tiempo. Luego se aplica un cáliper rotatorio sobre las aristas azules encontradas, lo que es de complejidad $O(n^2)$ ya que para cada arista $(O(n))$ se aplica un cáliper de $O(n)$. Además, en cada giro del cáliper, se requiere consultar los puntos rojos que estén dentro de las líneas paralelas de soporte, verificar si son aristas y si intersectan al segmento azul que le da soporte al cáliper, lo que, según lo explicado en la subsección 4.0.6, toma $O(logn)$ tiempo. Así, esta parte parte 4 tiene una complejidad total de $O(n^2logn)$.

En la parte final, se aplican los cuatro pasos anteriores, de manera análoga, intercambiando el grafo azul por el grafo rojo.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Cuadro Comparativo		
Etapa (Ver algoritmo 4.0.1)	**Propuesto en (Cortés et al., 2012)**	**Implementación**
(1)	$O(nlogn)$	$O(n^2)$
(2)	$O(n^2)$	$O(n^2)$
(a)	$O(nlogn)$	$O(nlogn)$
(b)	$O(n^2)$	$O(n^2)$
(3)	$O(n^2)$	$O(n^3)$
(a)	$O(n^2)$	$O(n^2)$
(b)	$O(n^2)$	$O(n^3)$
(i)	$O(logn)$	$O(n^2)$
(ii)	$O(n)$	$O(n)$
(iii)	$O(1)$	$O(1)$
(c)	$O(n^2)$	$O(n^2)$
(4)	$O(n^2)$	$O(n^2logn)$
TOTAL	$O(n^2)$	$O(n^3)$
(5)	$O(n^2)$	$O(n^3)$

Tabla 7.3: Cuadro Comparativo que hace un balance entre las complejidades algorítmicas propuestas por (Cortés et al., 2012) y las complejidades algorítmicas logradas en el Proyecto de Título

En la tabla 7.3 se presenta una comparativa entre las complejidades esperadas por cada etapa, las que son propuestas y probadas teóricamente en (Cortés et al., 2012), y las complejidades algorítmicas logradas en la implementación. Viendo el resultado final, se ve una diferencia bastante grande entre el esperado y el logrado, siendo cada uno $O(n^2)$ y $O(n^3)$ respectivamente.

Si se mira el desglose de cada etapa del algoritmo BR, es la parte 3 la que más afecta a la complejidad total, lo que se debe principalmente a la ausencia de la estructura de datos sugerida por los autores del algoritmo, la que permite realizar la ubicación de los puntos dentro de las regiones convexas en tiempo $O(logn)$ (punto 3.b.i). Logrando implementar

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

esa estructura de datos, se reduce considerablemente la complejidad total a un mínimo de $O(n^2logn)$, que es lo determinado por la parte 4, aunque de igual manera se lograría esta complejidad si se logra saber si un punto está dentro o fuera de un polígono en $O(logn)$; además se simplificaría bastante la actualización de particiones del plano en regiones convexas.

Para mejorar la parte 4, se requiere de un poco más de esfuerzo e invertir tiempo en revisión de más biliografía y estudiando diversos teoremas de la geometría computacional para lograr realizar el procedimiento en $O(n^2)$.

La parte 1, si bien no cumple con el esperado, no afecta a que la complejidad total aumente. Puede repararse si se logra saber si un punto está afuera o dentro de un polígono en $O(logn)$. En conclusión, las partes a mejorar para lograr el óptimo en la construcción del algoritmo BR son:

(1) Encontrar la posición de un punto respecto a un polígono (adentro o afuera) en $O(logn)$

(2) Implementar una estructura de datos como la propuesta en (Arge et al., 2006), para realizar la parte 2 en $O(n^2)$.

(3) Mejorar la búsqueda de la parte 4.

Quizás la mejora más importante es el punto (2), que incluso puede ser trabajado como un proyecto aparte. Este proyecto sólo contempla la implementación del algoritmo BR, considerando los algoritmos, métodos y técnicas más alcanzables en un tiempo acotado. El desarrollo de algoritmos más eficientes y complejos escapa completamente del tiempos estipulado. Por lo que, este proyecto, deja la implementación libre para que en un proyecto distinto se desarrollen los algoritmos más complejos utilizando el presente trabajo como base.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Capítulo 8

Pruebas de Software

8.1. Prueba de Software

El sistema desarrollado es sometido a una serie de pruebas para comprobar el comportamiento correcto del mismo. En esta serie de pruebas se buscan errores o falencias de programación que desencadenen en fallas del sistema.

Los módulos o elementos a probar en estas pruebas de software son:

- Validación de Formatos de Archivos
  La Validación de Formato de Archivo corresponde a un procedimiento que comprueba la correcta sintaxis y formato de tipos de datos que sean entregados como archivos de datos al software. Se valida que la comprobación del formato sea realizada correctamente y que no carezca de las restricciones necesarias para la detección de archivos con un mal formato o archivos dañados, lo cual ayuda a que el programa no procese datos erróneos.

- Validación módulo de generación de Grafo Aleatorio
  La validación del módulo de generación de grafo aleatorio corresponde al procedimiento que genera los datos de un grafo a partir de una cantidad de vértices introducida por el usuario. Se validará el tipo de dato introducido por el usuario (debe ser de tipo entero positivo, ya que es una magnitud), lo que permitirá la correcta creación de datos al grafo.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

- Validación módulo de generación de Grafo Manual
  La Validación del módulo de generación de Grafo Manual corresponde al procedimiento que restringe la cantidad de elementos que manualmente se pueden crear para un grafo determinado. Se valida que la restricciones funcione para la creación de ambos colores (rojos y azules), y mezclando la composición de los conjuntos de datos de los grafos. De esta forma, el análisis del algoritmo BR se producirá en las condiciones ideales.

8.2. Resultados de Pruebas de Software

En este apartado se presenta una tabla resumen de los resultados de las pruebas de software realizadas con el fin de evaluar el funcionamiento del software (para mayor detalle de las pruebas ver Anexo B).

Resultados de Pruebas de Software		
Módulos a Probar	**Caso de Prueba**	**Resultado**
Validación de Formatos de Archivos	Archivo Dañado, valores nulos	Fracaso
	Ruta inexistente	Fracaso
	Sintáxis Errónea	Fracaso
	Falta de Archivo	Fracaso
Validación módulo de generación de Grafo Aleatorio	Valores Nulos	Fracaso
	Caracteres o Simbolos	Fracaso
	Números Négativos	Fracaso
	Números Decimales	Fracaso
Validación módulo de generación de grafo manual	elementos fuera del cuadro de dibujo	Fracaso
	vértices o nodos de ambos colores (Rojo y Azul)	Fracaso
	Relacionar entre vértices distintos	Fracaso
	eliminar elementos en cascada	Fracaso

Tabla 8.1: Tabla de Resumen de Resultados de Pruebas de Software

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Observación 8.1 *Para fines de la prueba, ”Éxito” significa que el caso de prueba propuesto logra vulnerar el sistema; mientras que ”Fracaso” representa lo opuesto, que el caso de prueba no logró vulnerar al sistema.*

8.3. Conclusiones de las Pruebas de Software

Las pruebas de software son una herramienta útil en la detección de falencias, vulnerabilidades y peligros en un software. En el caso de este proyecto, las pruebas de software ayudan al desarrollo de una implementación aceptable para un ambiente de experimentación, es decir el registro y análisis de datos.

Así, el software respondió apropiadamente a las pruebas efectuadas, es posible concluir que el software requiere solamente mejoras en la implementación de la generación del grafo aleatorio, mejoras que tienen que ver con la comprensión del usuario al estado del proceso que esta realizando en el software, es decir, mensajes de alertas claros.

Si bien se concluye con un reporte favorable ante las pruebas de software, el software debe procurar mejorar sus rutinas y capturar las excepciones que eventualmente puedan suceder en la interacción con un usuario.

Finalmente, es importante que el software debe mejorar sus falencias, ya que pueden producir algún error; por otro lado, el desarrollo se ha llevado a cabo cuidando la integridad del producto, para evitar probables problemas en su ejecución.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Capítulo 9

Experimentación

9.1. Metodología de Experimentación

La experimentación para este proyecto consiste en documentar el comportamiento del algoritmo BR propuesto en (Cortés et al., 2012) frente a diferentes volúmenes de datos. De esta forma, se puede observar las tendencias del procesamiento de datos contrastando el volumen de datos con el tiempo de ejecución.

Para observar el comportamiento completamente se expondrán casos específicos, tales como conjuntos de puntos disjuntos, es decir, sin contacto, lo que implica que no existen intersecciones bicromáticas, conjuntos de puntos angostos y extendidos, entre otros casos. Estos experimentos comprueban la completitud de la solución del algoritmo, ya que se registra si el algoritmo logra la total cobertura de la solución.

Otro experimento es el comportamiento en volúmenes de datos masivos, es decir, que se expone a grandes volúmenes de datos para observar el comportamiento frente a muchos datos y procesamiento.

De estos experimentos se espera comprobar la correcta funcionalidad del algoritmo frente a casos extremos, y frente a grandes volúmenes de datos. Acotando nuevos registros sobre este algoritmo en un entorno experimental, logrando registrar comportamientos reales y comparables.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

9.2. Detalle de los Experimentos

Para realizar esta serie de experimentos, se planifica el tipo de experimento y el tipo de condiciones a medir. Con el fin de expresar correctamente el objetivo de cada experimento. Es importante diferenciar el tipo de prueba y especificar los datos en experimentación.

Cada caso de experimentación será realizado en dos equipos distintos (ver Tabla 9.1), y se ejecutarán en tres ocasiones por cada equipo para obtener datos veraces. Luego de ello se presentará un análisis de los datos obtenidos.

Equipos de Experimentación		
Características	**Equipo 1**	**Equipo 2**
Modelo	Lenovo G 470	HP Pavilion DV4 2028LA
Procesador	Intel Core i5 - 2450M - 2,50 GHz - x64	Intel Core i5 – m430 – 2,27GHz – x32
RAM	4 Gb - DDR3 - 1333 Mhz	4 GB - DDR3 -1333 Mhz
Sistema Operativo	Windows 8 Pro 64 bit	Windows 7 Ultimate 32 bit

Tabla 9.1: Equipos de experimentación

9.2.1. Desarrollo de la experimentación

Como ya se ha hecho mención, esta serie de experimentos se divide en tres objetivos, verificar cada parte del algoritmo por separado, comprobar el comportamiento frente a casos especiales del algoritmo completo y comprobar el comportamiento frente a grandes volúmenes de datos. Se detallan cada serie de pruebas en adelante:

- Experimentación del Compotamiento Individual de cada Paso del Algoritmo BR frente a casos Especiales

  Esta experimentación busca rescatar el rendimiento por cada sección del algoritmo por separado, para contrastar los resultados con la complejidad en tiempo expuesta en el Algoritmo.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Los casos de prueba se dividirán en cuatro secciones, dejando fuera el paso 5, ya que este consta de llamar a la misma operación involucrando inversamente los conjuntos de datos. Así, las secciones serían: (1) Obtención de Conjuntos de puntos externos e internos al Cierre Convexo, (2) Obtener $E(\overline{Ge})$ (Orden Rotacional y Cáliper Rotatorio), (3) Obtener $E(\overline{Gi})$ (Arreglo Dual, Localización Planar y Cáliper Rotatorio) y (4) Obtener $\{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$ (Cáliper Rotatorio en cada región)..
Los casos a probar son: (1) Conjuntos Disjuntos, (2) Conjuntos solapados pero con ausencia de intersecciones, (3) Conjuntos extendidos en los cuatro Cuadrantes, (4) Conjuntos con Intersecciones en el interior y (5) Conjuntos con Intersecciones en el Exterior. (ver conjuntos de datos utilizados enel anexo C)
Se registrará el tiempo de ejecución y se comprobará el conjunto solución obtenido.

- Experimentación del comportamiento frente a grandes volúmenes de datos

  Esta experimentación intenta comprobar el comportamiento del algoritmo frente a grandes volúmenes de datos, por ello se probarán conjuntos de datos predeterminados con un tamaño especificado. Cada experimento se probará en un volumen de dato y se aumentará gradualmente.
  Los volúmenes de datos a probar serán: (6) conjuntos de 100 vértices, (7) conjuntos de 300 vértices, (8) conjuntos de 500 vértices, (9) conjuntos de 1000 vértices. Se registrará el tiempo de ejecución.

9.2.2. Resultados de la Experimentación

A continuación, se presentan los resultados de los experimentos realizados según lo planificado en la sección anterior.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

- Experimentación del Compotamiento Individual de cada Paso del Algoritmo BR frente a casos Especiales.

Casos	**tiempos (ms)**			**Resultados Esperados**			**Resultados Obtenidos**			**Observaciones**
	RB	**BR**	**total**	**sb**	**sr**	**Total**	**sb**	**sr**	**Total**	
(1)	0,196644	0,092368	0,289012	0	0	0	0	0	0	
(2)	0,196643	0,112074	0,308717	0	0	0	0	0	0	
(3)	0,304201	0,106327	0,410528	0	0	0	0	0	0	
(4)	0,212653	0,097294	0,309947	0	0	0	0	0	0	
(5)	0,196643	0,105916	0,302559	0	0	0	0	0	0	

Tabla 9.2: Experimento 1- Paso 1: Cierre Convexo y Separación de Conjuntos

Casos	**tiempos (ms)**			**Resultados Esperados**			**Resultados Obtenidos**			**Observaciones**
	RB	**BR**	**total**	**sb**	**sr**	**Total**	**sb**	**sr**	**Total**	
(1)	3,232496	1,403594	4,636090	0	0	0	0	0	0	
(2)	2,369156	0,101401	2,470557	0	0	0	0	0	0	
(3)	0,407654	0,407654	0,815308	2	0	2	2	0	2	
(4)	0	0,734434	0,734434	0	7	7	0	7	7	
(5)	2,898738	0	2,898738	1	0	1	1	0	1	

Tabla 9.3: Experimento 2- Paso 2: Obtener $E(\overline{Ge})$

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Casos	tiempos (ms)			Resultados Esperados			Resultados Obtenidos			Observaciones
	RB	BR	total	sb	sr	Total	sb	sr	Total	
(1)	0	0	0	0	0	0	0	0	0	
(2)	5,966203	2,947180	8,913383	0	0	0	0	0	0	
(3)	1,271815	1,271815	2,543630	4	1	5	4	1	5	
(4)	11,021853	0	11,021853	6	0	6	6	0	6	
(5)	3,295718	3,06418	6,359898	1	7	8	1	8	9	

Tabla 9.4: Experimento 3- Paso 3: Obtener $E(\overline{Gi})$

Casos	tiempos (ms)			Resultados Esperados			Resultados Obtenidos			Observaciones
	RB	BR	total	sb	sr	Total	sb	sr	Total	
(1)	0,142453	0,024222	0,166675	0	0	0	0	0	0	
(2)	0,146559	0,045158	0,191717	0	0	0	0	0	0	
(3)	0,084979	0,133422	0,218401	4	7	11	4	7	11	
(4)	0,036126	0,02299	0,059116	0	0	0	0	0	0	
(5)	0,305433	0,022989	0,328422	4	0	4	4	0	4	

Tabla 9.5: Experimento 4- Paso 4: Obtener $\{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Casos	tiempos (ms)			Resultados Esperados			Resultados Obtenidos			Observaciones
	RB	**BR**	**total**	**sb**	**sr**	**Total**	**sb**	**sr**	**Total**	
(1)	3,571593	1,520184	5,091777	0	0	0	0	0	0	
(2)	6,977743	3,205813	10,183556	0	0	0	0	0	0	
(3)	8,724539	1,919218	10,643757	10	8	18	10	8	18	
(4)	11,270632	0,854718	12,125350	6	7	13	6	7	13	
(5)	6,696532	3,193085	9,889617	6	7	13	6	7	13	

Tabla 9.6: Experimento 5- Algoritmo BR Completo

Observación 9.1 *Los datos recopilados, presentados en los resultados, fueron recopilados evitando la sobreescritura de datos, es decir, que para cada caso de experimentación, se reinició el software completo. Esto se debe a que, cada vez que el software carga o sobreescribe los datos para su ejecución, independiente del conjunto de experimentación que sea, los resultados en tiempo de ejecución serán menores al real.*

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

- Experimentación del comportamiento frente a grandes volúmenes de datos

Cantidad de Datos	**Tiempo de Ejecución (milisegundos)**		
	RB	**BR**	**Total**
100	15539,712	9197,074	24736,787
300	8391704,0	162579,8	8554284,0
500	34826492,0	9274724,0	44101220,0
1000	179919940,74	71249665,94875	251169635,943124

Tabla 9.7: Experimento 6- Prueba de Volumen de Datos

Cantidad de Datos	**Tiempo de Ejecución (horas)**		
	RB	**BR**	**Total**
100	0,004	0,003	0,007
300	2,3	0,045	2,4
500	9,7	2,6	12,3
1000	49,98	19,79157	69,769343

Tabla 9.8: Experimento 6- Prueba de Volumen de Datos

Observación 9.2 *La experimentación de volumen está acotada a la barrera de los 1000 vértices por grafo, ya que con cantidades más elevadas, el computador ocupado en la experimentación no logra terminar por falta de memoria.*

9.2.3. Análisis de la Experimentación

De los experimentos presentados, es posible observar que sus tiempos de ejecución son similares y guardan cierta proporción respecto del volúmen de datos utilizados.

Para poder visualizar mejor estas tendencias, se presentan gráficos con la información del tiempo de ejecución de cada uno de los casos. Estos casos son representaciones de casos críticos del Algoritmo BR.

En la Figura 9.1, es posible apreciar que el algoritmo para la obtención del cierre convexo, el algoritmo Graham Scan, logra un rendimiento muy constante en cada caso.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

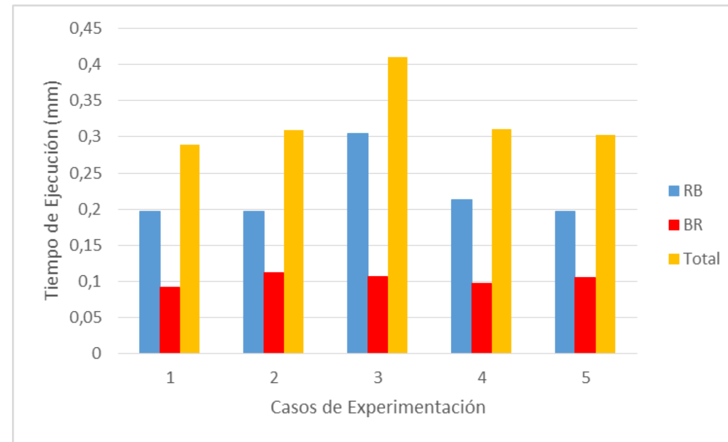


Figura 9.1: Experimento 1 Parte 1 - Cierre Convexo y Separación de Conjuntos

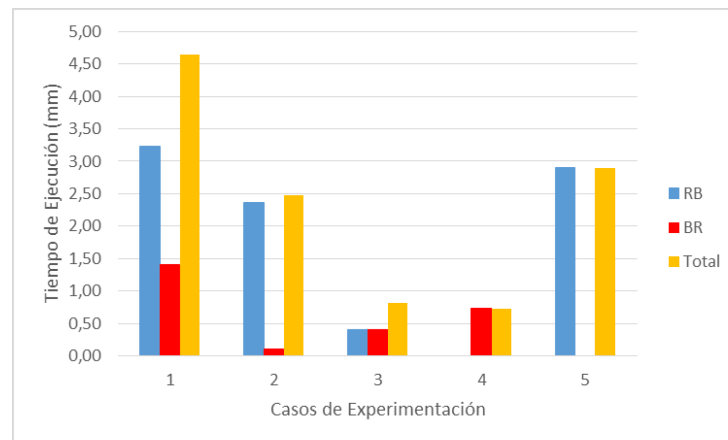


Figura 9.2: Experimento 2 Parte 2 - Obtener $E(\overline{Ge})$

Además, esta parte del Algoritmo consiste en crear una representación de un polígono que sea el límite del grafo, de tal forma de poder separar los vértices del grafo opuesto entre dos regiones, dentro del cierre convexo y fuera de éste. Cabe destacar, que en este paso no se realiza la búsqueda de la solución, pues esta etapa provee los datos necesarios para que las siguientes etapas cuenten con los datos restrigidos para obtener una solución.

Así, con los dos conjuntos de vértices que se obtienen de la parte 1 del Algoritmo BR, es posible tratar con ellos por separado, en procedimientos distintos y aislados (ver Sección 4.0.1). Estos conjuntos serán desarrollados en la parte 2, para obtener la solución de los segmentos que se intersectan atravesando al cierre convexo, y en la parte 3, para obtener

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

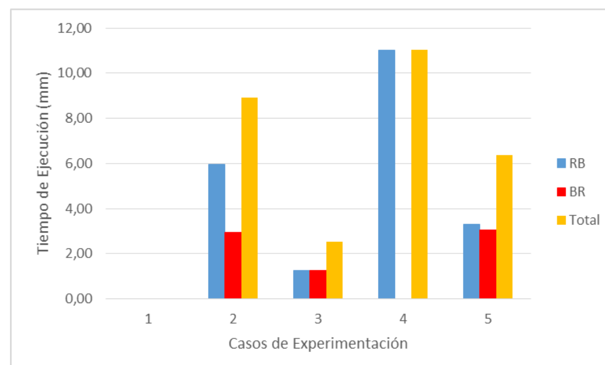


Figura 9.3: Experimento 3 Parte 3 - Obtener $E(\overline{Gi})$

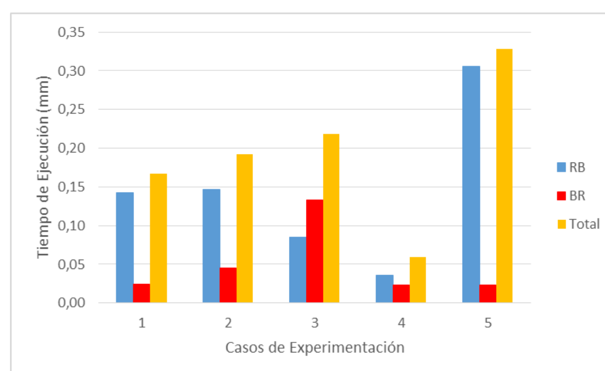


Figura 9.4: Experimento 4- Paso 4: Obtener $\{\overline{uv} : u \in V(Gi), v \in V(Ge)\}$

las intersecciones que se producen en el interior de dicho cierre.

En la Figura 9.2, es notoria la variación de tiempos de ejecución dados, esta gráfica corresponde al paso 2, su comportamiento está directamente relacionado con la cantidad de vértices que componen al conjunto de los puntos azules externos al Cierre Convexo, ya que en este paso construye el el Orden Rotacional y el Cáliper Rotatorio; estos procedimientos estarán completamente condicionados a la cantidad del conjunto de úntos exteriores al CH(R).

Una situación similar se muestra en la Figura 9.3, que corresponde al paso 3 del Algoritmo BR. De la misma forma que el paso 2, este procedimiento depende mucho del conjunto de vértices que se encuentran en el interior del cierre convexo, ya que se debe construir un Arreglo Dual de Líneas basado en ese conjunto, luego utilizar este arreglo para la construcción de las regiones convexas internas del cierre convexo, dejando a gru-

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

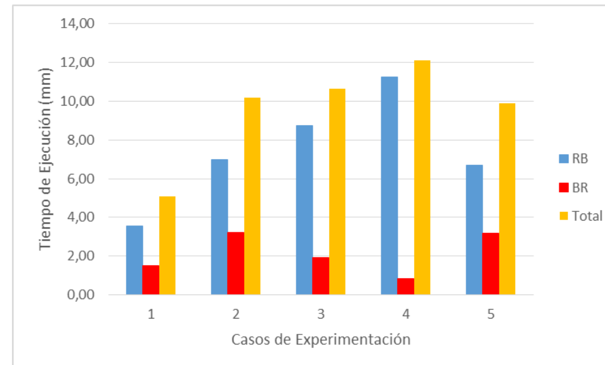


Figura 9.5: Experimento 5- Algoritmo BR Completo

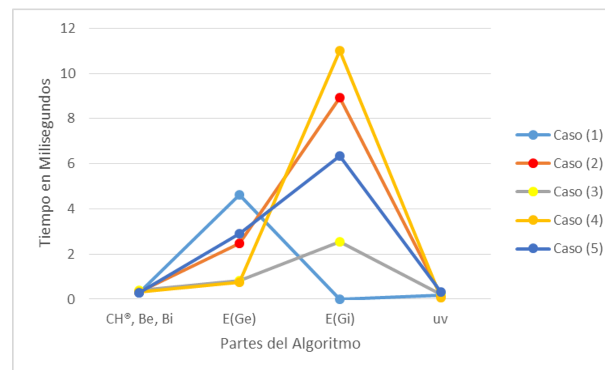


Figura 9.6: Comparativa de Comportamiento por Etapa del Algoritmo BR

pos aislados de vértices azules en ellos, para finalmente realizar el procedimiento para determinar la solución del Algoritmo BR.

Por otra parte, la Figura 9.4, contiene el comportamiento del procedimiento del paso 4, que consiste en encontrar las intersecciones entre vértices que perteneces dentro y fuera del polígono a la vez. A diferencia de los pasos 3 y 4 del Algoritmo BR, el paso 4 no necesita la creación de otro objeto para computar la solución, ya que éste utilizará las mismas estructuras creadas anteriormente en la ejecución. Además, el procedimiento es más básico en comparación con los demás procedimientos, pues se realizan preprocedimientos para obtener onjuntos de datos a analizar más pequños. El rendimiento de este procedimiento marca una tendencia, siempre es menor a los dos resultados anteriores.

Finalmente, se registra el cuadro de resultados generales, esto se puede apreciar en la Figura 9.5. El algoritmo BR es proporciona la solución para el conjunto RB (del Grafo azul

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

sobre el Grafo rojo), y además, para el conjunto BR (del Grafo rojo sobre el Grafo azul). El registro de ambas ejecuciones muestran que claramente que, a pesar de ser la misma forma de procedimiento, la composición de los conjuntos juega un rol importante en la ejecución, pues el rendimientos de los procedimientos son directamente proporcionales al tamaño de los conjuntos.

Existe otra tendencia en los tiempos de ejecucuión presentados, ya que generalmente, la solución *sr* es más rápida que el *sb*, esto se debe a que el algoritmo ya creó varias de las instancias necesarias para la ejecución del algoritmo, por lo que la ejecución es más rápida que la anterior.

También es notoria las diferencias en tiempo entre las Figuras 9.1 y 9.4, con respecto a las Figuras 9.2 y 9.3; ya que las primera Figuras mencionadas no alcanzan ni a medio milisegundo (0,5 ms), mientras que las superan los 4 milisegundos o más, siendo la parte que más tiempo ocupo, la parte 3, encargada de realizar el Arreglo Dual de Líneas, la Construcción de Regiones Convexas y la Búsqueda de la Solución al Algoritmo (ver Figura 9.6).

Cabe notar, que los primeros 2 casos, a pesar de no contener intersecciones bicromáticas de segmento, realiza gran parte de los cálculos de cada paso del Algoritmo BR, a excepción del caso 1 en el paso 3, donde no se ejecuta el procedimiento por restricción del algoritmo.

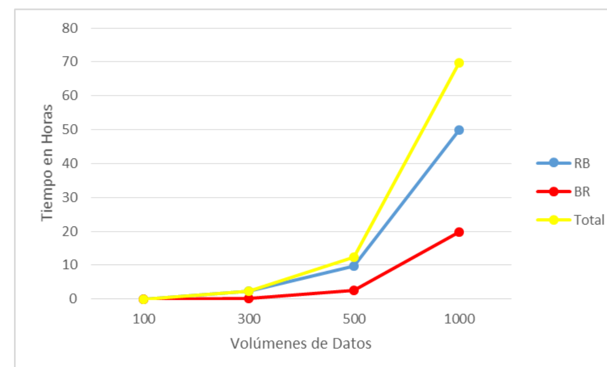


Figura 9.7: Experimento 6- Comportamiento en grandes Volúmenes de Datos

En cuanto los experimentos realizados con grandes volumenes de datos, el comportamiento registrado es el esperado, ya que el Algoritmo BR tiene una solución en tiempo de $O(n^2)$. En cada ejecución, al aumentar la cantidad de datos en los Grafos, el tiempo de ejecución aumentaba cuadráticamente. Tal como se puede apreciar en la Figura 9.7, el

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

tiempo de ejecución está directamente proporcional al volumen de datos usado.

El Algoritmo BR presenta una forma elegante de solución al problema de las Intersecciones Bicromátias de segmento, donde ésta se obtiene clasificadamente. Los procedimientos están diseñados para evadir conjuntos de datos que no representan el interés de la problemática, y se declara que esta solución está optimizada. Sin embargo, el procedimiento completo es sensible al tamaño y a la densidad de los conjuntos de datos a analizar.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Capítulo 10

Líneas de Trabajo Futuro

El problema planteado es una variación del problema original, en donde se incluyen las intersecciones bicromáticas y las intersecciones monocromáticas de segmento, lo que indica que es una línea de investigación muy específica. Además, el algoritmo propuesto por (Cortés et al., 2012) ha demostrado ser 3-Sum, es decir, que esta solución está completamente optimizada.

Por lo que este problema está solucionado para un punto de vista algorítmico, con un enfoque estructurado y considerando a todos los datos en memoria principal.

Sin embargo, es posible que la solución pueda mejorar al buscar formas de implementación sobre un entorno paralelizado, debiendo modificar el algoritmo a un procedimiento que acepte paralelismo, considerando balance de carga en procesamiento y destinar prioridades sobre fragmentos de la solución. Este lineamiento tiene sustento, ya que el algoritmo implementado en este proyecto logra disgregar las tareas realizadas sobre los grafos tanto en procesos y con usos de datos. El algoritmo en su primer paso crea un polígono convexo y separa dos grupos según su posición, con esto realizado, el segundo paso utiliza el grupo de punto externos con respecto al polígono convexo, mientras que en el tercer paso, procesa los datos del conjunto de puntos internos y con todos los datos del otro grafo. De esta forma, cabe la posibilidad de trabajar el algoritmo de forma paralela.

Por otro lado, la solución puede variar si se considera que no todos los datos están en memoria principal, ampliando el problema a la capacidad de memoria disponible y optimización de búsqueda en memoria secundaria. Este lineamiento tiene sustento, ya que el algoritmo pese a que el conjunto de datos total puede ser un volumen enorme, en cada

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

paso del algoritmo se trabaja con una parte fraccionada del total del conjunto. Como ya se explicaba, en el segundo paso del algoritmo, los datos utilizados en el procedimiento son los datos del conjunto de puntos azules externos, descartando los demás conjuntos de datos para otros futuros procesos. Con este argumento, es posible trabajar el algoritmo de forma que almacene en memoria secundaria los conjuntos de datos que no se trabajen en ciertos momentos y alternar los conjuntos en memoria principal.

Siguiendo con las posibilidades que se observa de la solución, es posible aplicarla en situaciones que requieren más investigación en comportamiento, compatibilidad, eficiencia y eficacia. Por ejemplo, asumiendo que existe un conjunto de datos almacenados en un sistema estructurado, es decir, en una base de datos utilizando la estructura de un R-Tree, con el propósito de recuperar y aprovechar las propiedades de esta estructura para obtener el conjunto solución de intersecciones bicromáticas. La complejidad de esta problemática es obtener la compatibilidad espacial al incorporar este análisis geométrico.

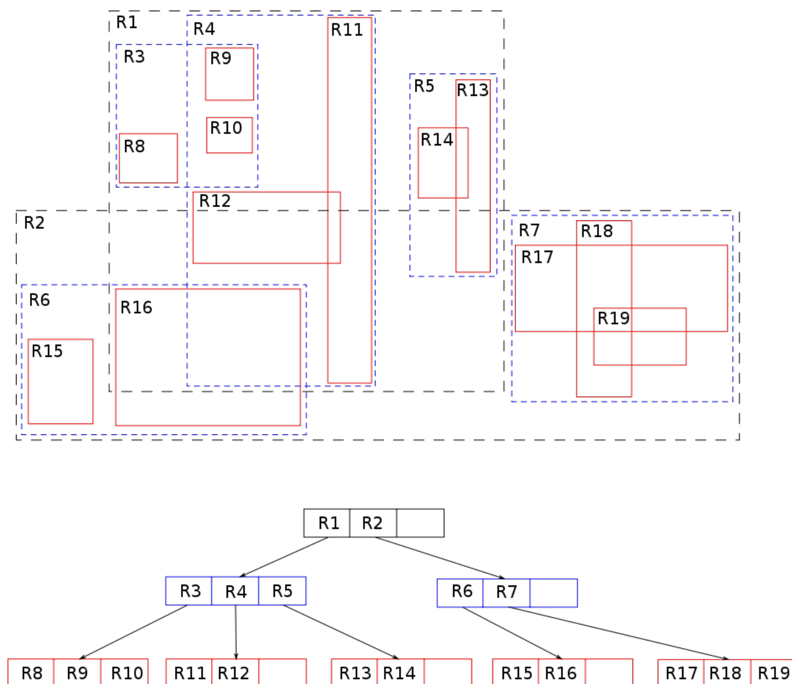


Figura 10.1: Estructura R-tree

Otro ejemplo necesitaría suponer que no existe un conjunto de datos almacenados, ni menos organizados en una estructura, si no que existe un gran volumen de datos no

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

indexados por procesar antes de someterlo a un análisis de intersecciones bicromáticas. El propósito es construir un sistema estructurado que sea capaz de soportar eficientemente un volumen de datos que excederá los límites permitidos en memoria principal, lograr una buena organización y una búsqueda eficiente dentro de la estructura. Existen variadas soluciones aisladas para esta problemática, de los cuales destaca la estructura; sin embargo, se destaca la estructura VA files. Luego, la problemática se transforma nuevamente en la compatibilidad con el análisis de intersección bicromática, de forma de alcarzar cada espacio de memoria que ocupa la gran cantidad de datos almacenados.

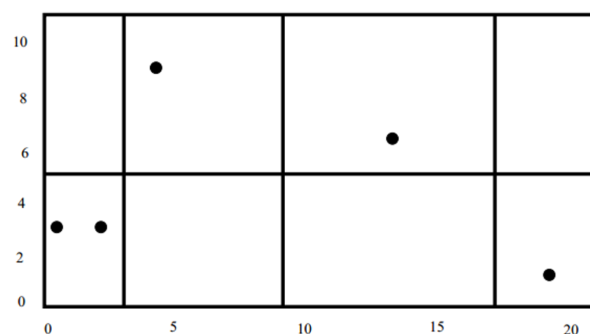


Figura 10.2: Estructura VA-files

Siguiendo otra línea de desarrollo, que se aleja de las mencionadas, este algoritmo puede tomar lineamientos combinándolo con otras áreas de las ciencias de la computación. Solamente por mencionar, dadas las características del análisis realizado por el algoritmo, es posible otorgar un significado diferente al de una intersección bicromática. Esto ocurre al incluir un conjunto de datos que no son geométricos, geográficos o similar, sino que se construye un modelo basado en conjuntos de datos con campos u orígenes distintos, parametrizándolo mediante los valores y relacionando mediante valores comunes, de forma que implementen un grafo. Por ejemplo, al utilizar un grafo basado en la web, el significado de las intersecciones bicromáticas podría ser un indicador relevante, el que debe investigarse. Otro ejemplo, es el del grafo social el cual se basa en el comportamiento de los contactos en redes sociales, de esta forma las intersecciones bicromáticas podrían resultar en un indicador relevante en términos de comportamiento social.

Así, el problema tiene líneas de investigación futura al considerar variables en la forma de implementación, ya que la forma de afrontar una solución satisface a un problema 3-

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

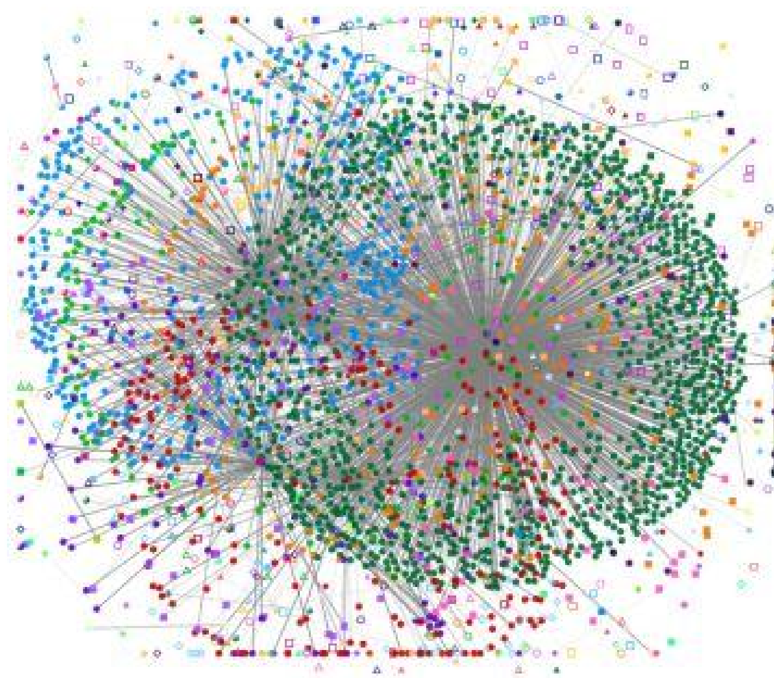


Figura 10.3: Grafo de la Web

Sum, es decir, que solo es posible trabajar la solución considerando aspectos no ideales para su implementación, como ya se ha dicho en cuanto al Paralelismo, el Trabajo en Memoria secundaria, la construcción basada en una estructura R-tree y el desarrollo basado en volumenes gigantes de datos no indexados. Además, la investigación podría ser orientada en la búsqueda de indicadores especializados en otras áreas. De esta forma, la investigación sobre las Intersecciones bicromáticas no está limitada al desarrollo de los conocimientos de la geometría computacional, sino que tiene el potencial de ser parte de nuevas formas de analizar la información.

Finalmente, y como se ha visto en capítulos anteriores, es necesario que la implementación desarrollada en este proyecto de título sea mejorada en términos de complejidad algorítmica, para así lograr igualar lo propuesto por los autores. Por eso es necesario realizar un estudio dedicado a estructuras de datos como las propuestas por (Arge et al., 2006) y por (Kirkpatrick, 1981), lo que significaría un proyecto aparte que complemente al presente.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Capítulo 11

Resumen de Esfuerzo Requerido

El desarrollo de este proyecto significó invertir tiempo en diferentes tareas y actividades para los integrantes del equipo de investigación y desarrollo, ya que cada una de estas tareas es importante para el éxito del proyecto.

Este tiempo es considerado el esfuerzo requerido para el desarrollo del proyecto, expresado en términos de tiempo (en horas hombre ocupadas en el proyecto). Esto es un instrumento de medición que logra identificar las actividades más relevantes del desarrollo, es decir, los puntos críticos del proyecto. Además registrará el comportamiento de este proyecto utilizando el tiempo de desarrollo empleado para futuros desarrollos similares.

En este proyecto, el esfuerzo estuvo centrado en la búsqueda de conocimiento científico, especificamente en Geometría Computacional. Cada investigador abordó una temática en particular, estudiando sus propiedades desde la teoría hasta la aplicación. Cabe destacar que fue una tarea que ocupó gran parte del tiempo aplicado al desarrollo. Sin embargo, este proceso resultó provechoso, ya que este conocimiento fue crucial para aplicar las diferentes técnicas y métodos ajustándolos a la implementación de la solución del Algoritmo BR.

Luego, el desarrollo de la solución estuvo marcada por la constante búsqueda de material teórico adicional para lograr una implementación útil.

A continuación se presenta una tabla 11.1 resumen del esfuerzo requerido distribuido en las actividades del proyecto.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Resumen de Esfuerzo Requerido	
Actividades / Fases	**N^o Horas**
Estudio del Arte	824
Implementación de Algoritmo BR	349
Diseño de Clases	106
Diseño de Interfaz de usuario	25
Implementación de Diseño	218
Experimentación y Evaluación de la Implementación	23
Experimentación de Casos	11
Análisis de los Resultados	12
Documentación del Proyecto	36
TOTAL	1232

Tabla 11.1: Resumen de Esfuerzo Requerido

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Capítulo 12

Conclusiones

Terminado el proyecto, y en contraste a los objetivos de éste, planteados en un principio, se puede concluir lo siguiente:

- Se logró implementar en lenguaje *Java* el algoritmo BR, propuesto en (Cortés et al., 2012), el cual reporta las intersecciones bicromáticas entre dos conjuntos de segmentos, uno de color rojo, y otro de color azul.
- Se estudió y analizó cada parte del algoritmo, adentrándose en cada una de las diversas áreas de la geometría computacional, logrando entender y aplicar la mayoría de los conceptos. La excepción a lo anterior, es la ubicación de puntos en una subdivisión planar, que permite consultas en $O(logn)$ y que puede ser estudiada en (Arge et al., 2006).
- Se diseñó una estructura de clases adecuada al proyecto, con un alto grado de encapsulamiento de métodos. Se aplicó al máximo posible la Orientación a Objetos. Los diagramas quedaron detallados en el capítulo 6 y en el anexo **??**. Además, se logró presentar una interfaz de usuario bastante amigable y clara, que permite realizar las pruebas pertinentes y ejecutar la aplicación, en base al diseño previo.
- Se realizó la correspondiente experimentación, análisis y evaluación de la solución implementada, haciéndose una comparativa entre los resultados propuestos en (Cortés et al., 2012) y los logrados en el proyecto, notándose una importante diferencia, causada por la no comprensión, y por ende no implementación, de la ubicación de puntos señalada en el segundo ítem de estas conclusiones. Se necesita mayor tiempo de estudio para lograr igualar la complejidad algorítmica propuesta.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Además, se logró cumplir con los objetivos del software, permitiendo que el sistema reciba diferentes entrada de datos, tanto desde archivos, como generados de forma aleatoria o manual.

El sistema entrega información parcial de los resultados, mostrando los datos entregados por cada etapa del algoritmo, entregando finalmente la solución requerida.

En términos académicos, este proyecto resultó ser un gran desafío, debido a la novedad del área y la inexperiencia en geometría computacional presentada por las partes involucradas. Sin embargo, y contra lo esperado, se logró sacar adelante el proyecto entendiendo la totalidad del procedimiento. No obstante, y tal como se ha indicado en líneas anteriores, no se logró entender por completo una estructura necesaria para realizar la partición de polígonos convexos.

El proyecto realizado presenta una oportunidad muy grande de desarrollarse en el área, dando lugar a la publicación de trabajos y presentación de ellos en congresos del tema.

Finalmente, y luego de todo el tiempo y esfuerzo requerido para lograr los objetivos, se logra ver la real importancia de conocer a cabalidad los requerimientos de un proyecto informático antes de comenzar su desarrollo, ya que puede presentar un crecimiento importante a medida que avanza y lo que en un principio se parecía pequeño y de fácil realización, puede terminar en un proyecto de gran magnitud, comprometiendo el cumplimiento de los plazos de entrega, sobre todo en un área que es desconocida.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Referencias

Lars Arge, Gerth Stolting Brodal, y Loukas Georgiadis. Improved dynamic planar point location. En *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '06, págs. 305–314. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2720-5. doi:10.1109/FOCS.2006.40. URL http://dx.doi.org/10.1109/FOCS.2006.40.

Sunil Arya, Theocharis Malamatos, y David M. Mount. Nearly optimal expected-case planar point location. 2000.

Hanna Baumgarten, Hermann Jung, y Kurt Mehlhorn. Dynamic point location in general subdivisions. En *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '92, págs. 250–258. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992. ISBN 0-89791-466-X. URL http://dl.acm.org/citation.cfm?id=139404.139458.

Jakob Andreas Bærentzen, Jens Gravesen, François Anton, y Henrik Aanæs. *Guide to Computational Geometry Processing - Foundations, Algorithms, and Methods.* Springer, 2012. ISBN 978-1-4471-4074-0.

Siu-Wing Cheng y Ravi Janardan. New results on dynamic planar point location. *SIAM J. Comput.*, 21(5):972–999, 1992. URL http://dblp.uni-trier.de/db/journals/siamcomp/siamcomp21.html#ChengJ92.

Carmen Cortés, Delia Garijo, Maria Angeles Garrido, Clara I. Grima, Alberto Márquez, Auxiliadora Moreno-González, Jesus Valenzuela, y Maria Trinidad Villar. Reporting bichromatic segment intersections from point sets. *Int. J. Comput. Geometry Appl.*, págs. 421–438, 2012.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Mark de Berg, Marc van Kreveld, y Mark Overmars... [et al.]. *Computational geometry : algorithms and applications*. Springer, Berlin, Heidelberg, New York, 1997. ISBN 3-540-61270-X. URL http://opac.inria.fr/record=b1093337.

Satyan L. Devadoss y Joseph O'Rourke. *Discrete and Computational Geometry*. Princeton University Press, 2011. ISBN 978-0-691-14553-2.

J R Driscoll, N Sarnak, D D Sleator, y R E Tarjan. Making data structures persistent. En *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, págs. 109–121. ACM, New York, NY, USA, 1986. ISBN 0-89791-193-8. doi:10.1145/12130.12142. URL http://doi.acm.org/10.1145/12130.12142.

E. Frankland. *Inorganic Chemistry*. Lea brothers & Company, 1884. URL http://books.google.cl/books?id=WAtDAAAAIAAJ.

Anka Gajentaan y Mark H. Overmars. On a class of o(n2) problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995. doi:10.1016/0925-7721(95)00022-2. URL http://dx.doi.org/10.1016/0925-7721(95)00022-2.

Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1(4):132–133, 1972.

Eric Haines. An introduction to ray tracing. cap. Essential Ray Tracing Algorithms, págs. 33–77. Academic Press Ltd., London, UK, UK, 1989. ISBN 0-12-286160-4. URL http://dl.acm.org/citation.cfm?id=94788.94790.

Martin Held. *On the Computational Geometry of Pocket Machining*, tomo 500 de *Lecture Notes in Computer Science*. Springer, 1991. ISBN 3-540-54103-9.

Hiroshi Imai y Takao Asano. Dynamic orthogonal segment intersection search. *J. Algorithms*, 8(1):1–18, 1987. URL http://dblp.uni-trier.de/db/journals/jal/jal8.html#ImaiA87.

David G. Kirkpatrick. Optimal search in planar subdivisions. Inf. téc., Vancouver, BC, Canada, Canada, 1981.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

D. König. *Theorie der endlichen und unendlichen Graphen: kombinatorische Topologie der Streckenkomplexe*. Chelsea Pub. Co., 1936. URL http://books.google.es/books?id=Afg5AAAAMAAJ.

Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*, tomo 1 de *EATCS Monographs on Theoretical Computer Science*. Springer, 1984. ISBN 978-3-642-69672-5. URL http://dx.doi.org/10.1007/978-3-642-69672-5.

Francisco Rivero Mendoza. *Geometría Computacional (Apunte)*. Universidad de Los Andes, Venezuela, Mérida, Venezuela, 1997. URL http://www.ciens.ula.ve/matematica/publicaciones/libros/por_profesor/lico/geometria_computacional.pdf.

Christian Worm Mortensen, Rasmus Pagh, y Mihai Patrascu. On dynamic range reporting in one dimension. En Harold N. Gabow y Ronald Fagin, eds., *STOC*, págs. 104–111. ACM, 2005. ISBN 1-58113-960-8. URL http://dblp.uni-trier.de/db/conf/stoc/stoc2005.html#MortensenPP05.

Ketan Mulmuley. A fast planar partition algorithm, i (extended abstract). En *FOCS*, págs. 580–589. IEEE Computer Society, 1988. URL http://dblp.uni-trier.de/db/conf/focs/focs88.html#Mulmuley88.

Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, New York, NY, USA, 2nd edón., 1998. ISBN 0521640105.

Franco P. Preparata y Michael I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985. ISBN 0-387-96131-3.

Neil Sarnak y Robert E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29(7):669–679, 1986. ISSN 0001-0782. doi:10.1145/6138.6151. URL http://doi.acm.org/10.1145/6138.6151.

Raimund Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1(1):51–64, 1991. ISSN 0925-7721. doi:10.1016/S0925-7721(99)00042-5. URL http://dx.doi.org/10.1016/S0925-7721(99)00042-5.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Michael Ian Shamos. *Computational Geometry*. Tesis Doctoral, Yale University, 1978.

Godfried Toussaint. Solving geometric problems with the rotating calipers. *In Proc. IEEE MELECON '83*, págs. 10—02, 1983. doi:10.1.1.40.2140. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.2140.

Dan E. Willard. A density control algorithm for doing insertions and deletions in a sequentially ordered file in good worst-case time. *Inf. Comput.*, 97(2):150–204, 1992. URL http://dblp.uni-trier.de/db/journals/iandc/iandc97.html#Willard92.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Apéndice A

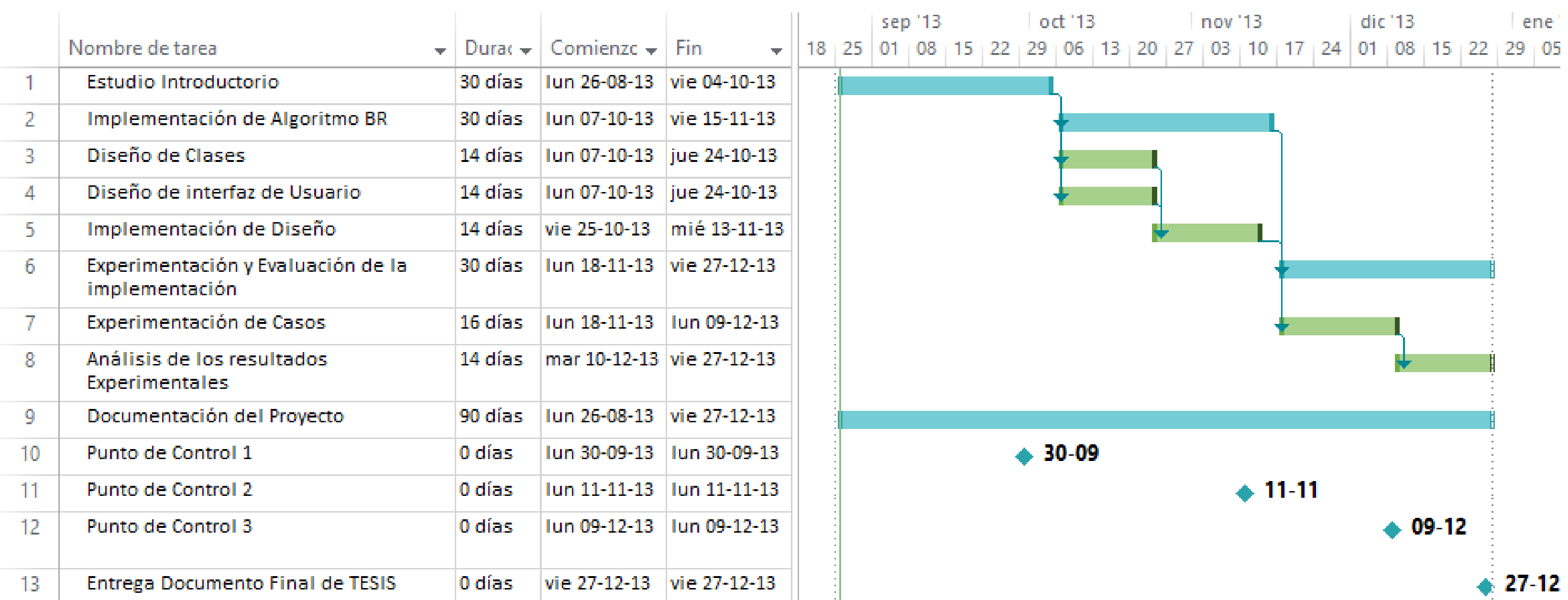
PLANIFICACIÓN INICIAL DEL PROYECTO

El presente proyecto será llevado a cabo según las actividades de: Estudio introductorio, Implementación de Algoritmo BR, Diseño de Clases, Diseño de Interfaz de usuario, Implementación de Diseño, Experimentación y Evaluación de la Implementación, Experimentación de casos, análisis de los resultados, Documentación del Proyecto. Estas actividades estructuran el desarrollo y ayudan al seguimiento del mismo.

A continuación se presenta la Carta Gantt que representa las actividades antes mencionadas y asigna una unidad de tiempo. Esta Carta Gantt está desarrollada en Microsoft Project Pro 2013.

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Carta Gantt
Proyecto de Titulo
Joel Torres Carrasco – Cristian Vallejos Vega
Agosto, 2013



	Nombre de tarea	Durac	Comienzo	Fin
1	Estudio Introductorio	30 días	lun 26-08-13	vie 04-10-13
2	Implementación de Algoritmo BR	30 días	lun 07-10-13	vie 15-11-13
3	Diseño de Clases	14 días	lun 07-10-13	jue 24-10-13
4	Diseño de interfaz de Usuario	14 días	lun 07-10-13	jue 24-10-13
5	Implementación de Diseño	14 días	vie 25-10-13	mié 13-11-13
6	Experimentación y Evaluación de la implementación	30 días	lun 18-11-13	vie 27-12-13
7	Experimentación de Casos	16 días	lun 18-11-13	lun 09-12-13
8	Análisis de los resultados Experimentales	14 días	mar 10-12-13	vie 27-12-13
9	Documentación del Proyecto	90 días	lun 26-08-13	vie 27-12-13
10	Punto de Control 1	0 días	lun 30-09-13	lun 30-09-13
11	Punto de Control 2	0 días	lun 11-11-13	lun 11-11-13
12	Punto de Control 3	0 días	lun 09-12-13	lun 09-12-13
13	Entrega Documento Final de TESIS	0 días	vie 27-12-13	vie 27-12-13

Figura A.1: carta Gantt - Proyecto de Título

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Apéndice B

ESPECIFICACIÓN DE LAS PRUEBAS

B.1. Pruebas de Unidad

1. Validación de Formatos de Archivos

id	Características a Probar	Datos Entrada		Salida Esperada	Salida Obtenida	Éxito / Fracaso	Observaciones
		Ruta Archivo B	Ruta Archivo R				
PS01	Archivo Dañado	dañado	correcto	mensaje de error	mensaje Error	Fracaso	
PS02	Archivo Dañado	correcto	dañado	mensaje de error	Mensaje de Error	Fracaso	
PS03	Ruta inexistente	ruta falsa	ruta real	imposibilitar ruta	ruta imposibilitada	Fracaso	
PS04	Ruta inexistente	ruta real	ruta falsa	imposibilitar ruta	ruta Imposibilitada	Fracaso	
PS05	Sintáxis Errónea	error	correcto	mensaje de error	Mensaje de Error	Fracaso	
PS06	Sintáxis Errónea	correcto	error	mensaje de error	Mensaje de Error	Fracaso	
PS07	Falta de Archivo	nulo	correcto	mensaje de error	Mensaje de Error	Fracaso	
PS08	Falta de archivo	correcto	nulo	Mensaje de error	Mensaje de Error	Fracaso	

Tabla B.1: Prueba de Software - validación de Formatos de archivos

2. Validación módulo de generación de Grafo Aleatorio

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

id	Características a Probar	Datos Entrada		Salida Esperada	Salida Obtenida	Éxito / Fracaso	Observaciones
		Ruta Archivo B	Ruta Archivo R				
PS09	Valores Nulos	Nulo	correcto	mensaje de error	Mensaje de Error	Fracaso	
PS10	Valores Nulos	correcto	Nulo	mensaje de error	Mensaje de Error	Fracaso	
PS11	Caracteres o Simbolos	caracter	número correcto	mensaje de error	Mensaje de Error	Fracaso	
PS12	Caracteres o Simbolos	número correcto	caracter	mensaje de error	Mensaje de Error	Fracaso	
PS13	Números Negativos	Negativo	Positivo	mensaje de error	Mensaje de Error	Fracaso	
PS14	Números Negativos	Positivo	Negativo	mensaje de error	Mensaje de Error	Fracaso	
PS15	Números Decimales	decimal	número correcto	mensaje de error	-	Fracaso	No muestra Mensaje, no ingresa el decimal
PS16	Números Decimales	número correcto	decimal	Mensaje de error	-	Fracaso	No muestra Mensaje, no ingresa el decimal

Tabla B.2: Prueba de Software - validación módulo de generación de Grafo Aleatorio

3. Validación módulo de generación de grafo manual

id	Características a Probar	Datos Entrada MouseAction	Salida Esperada	Salida Obtenida	Éxito / Fracaso	Observaciones
PS17	Elementos fuera del campo de dibujo	Seleccionar un nuevo vértice y arrastarlo fuera de los límites	No permitir	No permite	Fracaso	
PS18	Elementos fuera del campo de dibujo	Seleccionar una nueva Arista y arrastarlo fuera de los límites	No Permitir	NO permite	Fracaso	
PS19	vértices o nodos de ambos colores	Seleccionar un nuevo vértice azul y crear vértices	crear hasta 10 nodos	crear hasta 10 nodos	Frasaco	
PS20	vértices o nodos de ambos colores	Seleccionar un nuevo vértice rojo y crear vértices	crear hasta 10	crear hasta 10 nodos	Fracaso	
PS21	Relacionar entre vértices distintos	Seleccionar una nueva arista y unir un nodo azul a uno rojo	No permitir	nNO permite	Fracaso	
PS22	Relacionar entre vértices distintos	Seleccionar una nueva arista y unir un nodo rojo a uno azul	No permitir	No permite	Fracaso	
PS23	Eliminar elementos en cascada	Seleccionar eliminar vértice y eliminar nodos relacionados	Eliminar correctamente	Elimina correctamente	Fracaso	
PS24	Eliminar elementos en cascada	Seleccionar eliminar arista y eliminar aristas	Eliminar correctamente	Elimina correctamente	Fracaso	

Tabla B.3: Prueba de Software - validación módulo de generación de grafo manual

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Apéndice C

ESPECIFICACIÓN DE LOS EXPERIMENTOS

C.1. Conjuntos de Datos usados en la Experimentación

1. **Conjuntos Disjuntos**

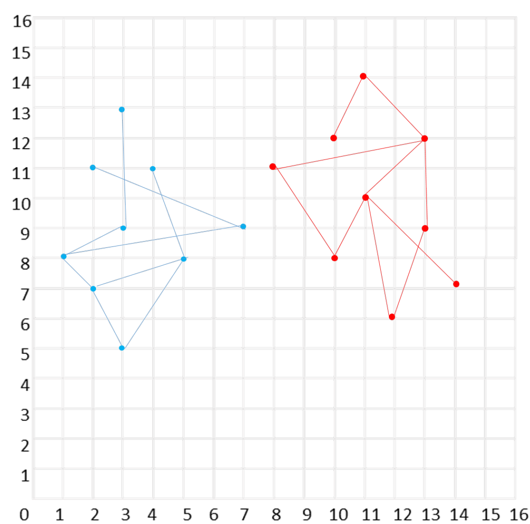


Figura C.1: Caso 1: Conjuntos Disjuntos

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Grafo B:

```
<vertices>
3 13
2 11
4 11
3 9
7 9
1 8
5 8
2 7
3 5
<\vertices>
<aristas>
0 3
1 4
2 6
3 5
4 5
5 7
6 7
6 8
7 8
<\aristas>
```

Grafo R:

```
<vertices>
11 14
10 12
13 12
8 11
11 10
13 9
10 8
14 7
12 6
<\vertices>

<aristas>
0 1
0 2
2 3
2 4
2 5
3 6
4 6
4 7
4 8
5 8
<\aristas>
```

2. **Conjuntos Solapados, pero con ausencia de intersecciones**

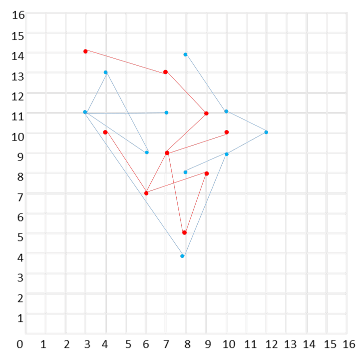


Figura C.2: Caso 2: conjuntos Solapados pero con ausencia de intersecciones

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Grafo B:

```
<vertices>
8 14
4 13
3 11
7 11
10 11
12 10
6 9
10 9
8 8
8 4
<\vertices>

<aristas>
0 4
1 2
1 6
2 3
2 6
4 5
5 7
2 9
7 9
7 8
<\aristas>
```

Grafo R:

```
<vertices>
3 14
7 13
9 11
4 10
10 10
7 9
9 8
6 7
8 5
<\vertices>

<aristas>
0 1
1 2
2 5
4 5
3 7
5 7
5 8
6 7
6 8
<\aristas>
```

---

3. **Conjuntos Extendidos en los cuatro Cuadrantes**
   Grafo B:

```
<vertices>
-4 14
6 7
1 4
-14 1
-2 -2
4 -2
9 -3
4 -6
-5 -12
<\vertices>

<aristas>
0 4
0 8
1 2
1 6
2 5
5 7
4 7
3 4
5 8
3 8
<\aristas>
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Grafo R:

```
<vertices>
-9 12
3 11
2 7
9 4
-12 2
7 -4
-15 -8
-6 -9
2 -12
<\vertices>
<aristas>
0 2
1 2
1 5
2 8
3 5
4 6
4 7
5 6
5 8
7 8
<\aristas>
```

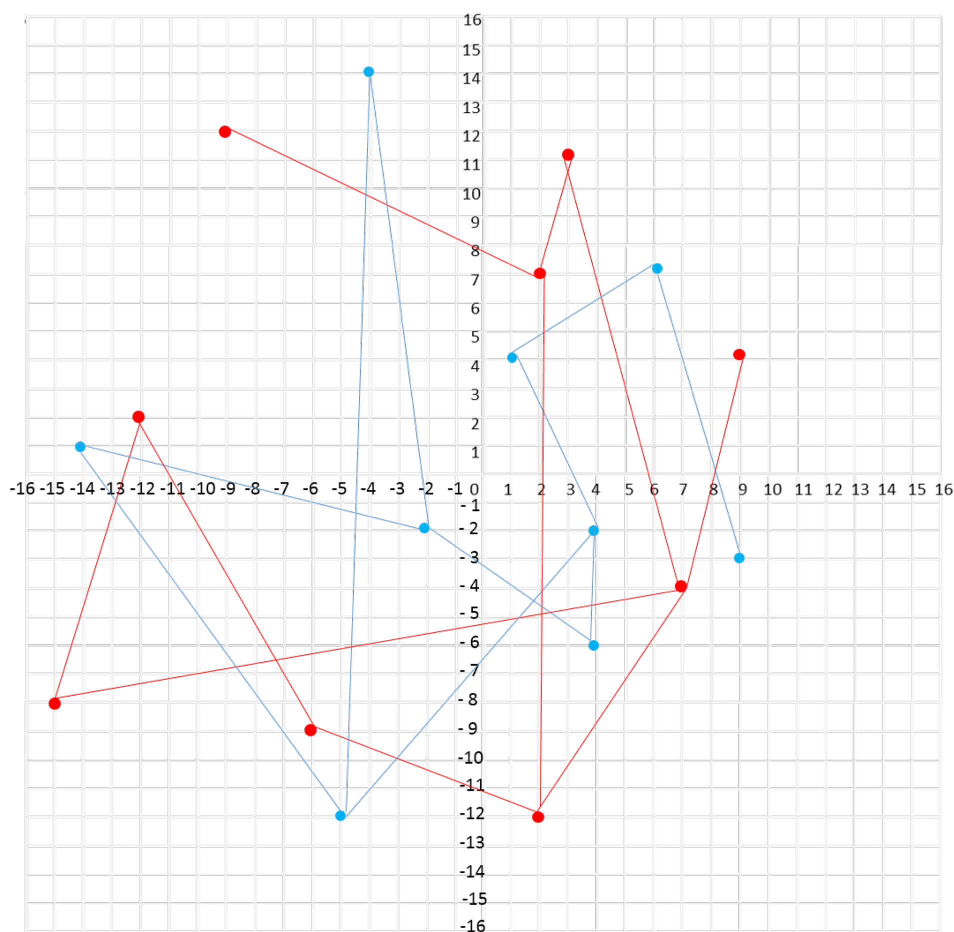


Figura C.3: Caso 3: Conjuntos Extendidos en los Cuatro cuadrantes

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

4. **Conjuntos con Intersecciones en el Exterior**

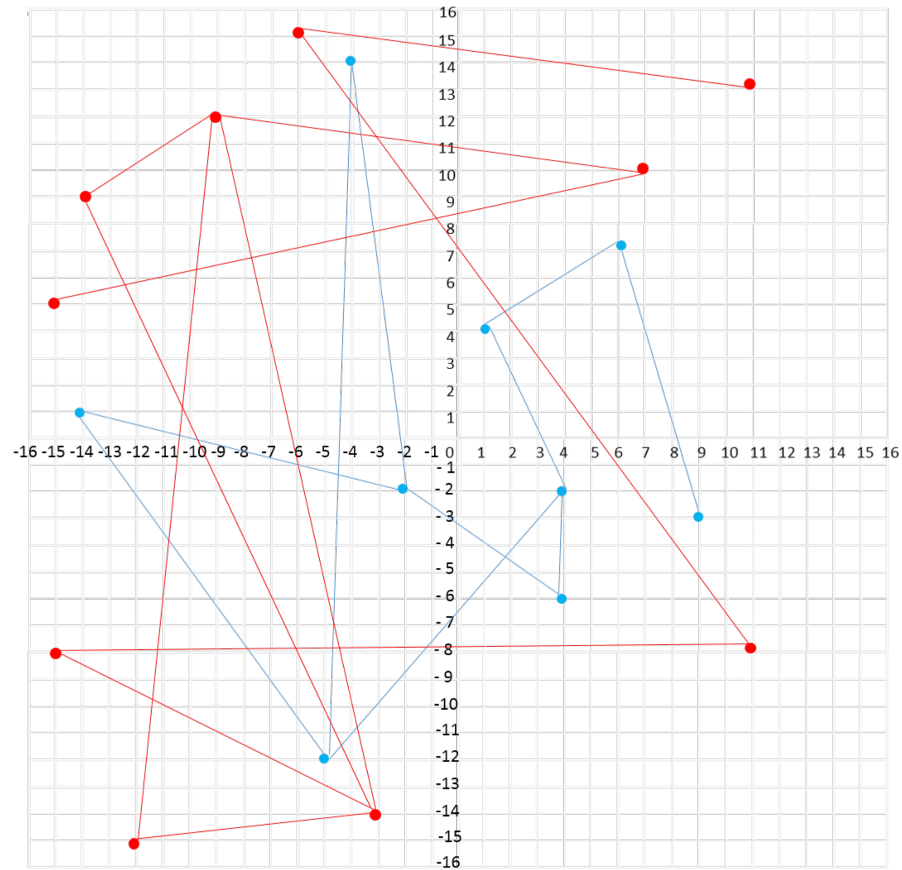


Figura C.4: Caso 4: Conjuntos con Intersecciones en el Exterior

Grafo B:

```
<vertices>
-4 14
6 7
1 4
-14 1
-2 -2
4 -2
9 -3
4 -6
-5 -12
<\vertices>
<aristas>
0 4
0 8
1 2
1 6
2 5
5 7
4 7
3 4
5 8
3 8
<\aristas>
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Grafo R:

```
<vertices>
-6 15
11 13
-9 12
7 10
-14 9
-15 5
-15 -8
11 -8
-3 -14
-12 -15
<\vertices>

<aristas>
0 1
0 7
2 3
2 4
2 8
2 9
3 5
4 8
6 7
6 8
8 9
<\aristas>
```

---

5. **Conjuntos con Intersecciones en el Interior**

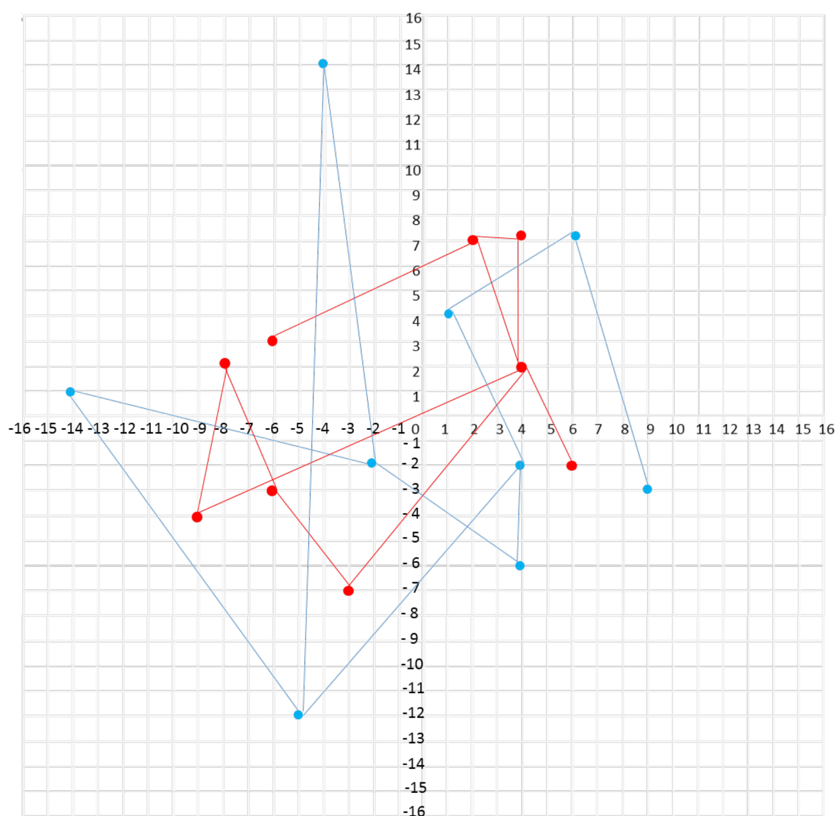


Figura C.5: Caso 5: Conjuntos con Intersecciones en el Interior

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Grafo B:

```
<vertices>
-4 14
6 7
1 4
-14 1
-2 -2
4 -2
9 -3
4 -6
-5 -12
<\vertices>
<aristas>
0 4
0 8
1 2
1 6
2 5
5 7
4 7
3 4
5 8
3 8
<\aristas>
```

Grafo R:

```
<vertices>
2 7
4 7
-6 3
-8 2
4 2
6 -2
-6 -3
-9 -4
-3 -7
<\vertices>
<aristas>
0 1
0 2
0 4
1 4
3 7
3 6
4 5
4 7
4 8
6 8
<\aristas>
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Grafo

	Atributo	Tipo
*	vertices	: ArrayList<vertice>
*	aristas	: ArrayList<arista>
*	nombre	: String

		Método	Retorno
+	<<Constructor>>	Grafo ()	
+	<<Constructor>>	Grafo (String nombre)	
+		addVertice (Double x, Double y)	: void
+		addArista (vertice desde, vertice hasta)	: void
+		addAristaIndice (int desde, int hasta)	: void
+		esVacio ()	: boolean
+		GrahamScan ()	: ArrayList<vertice>
+		CH ()	: Stack<vertice>
+		PuntoMenorOrdenada (ArrayList<vertice> puntos)	: vertice
+		OrdenarAngularmente (ArrayList<vertice> S, vertice m)	: ArrayList<vertice>
+		mergesort (ArrayList<vertice> data, int first, int n, vertice m)	: void
-		merge (ArrayList<vertice> data, int first, int n1, int n2, vertice m)	: void
+		Angulo (vertice A, vertice B)	: double
+		estaIzquierda (vertice A, vertice B, vertice C)	: boolean
+		esArista (vertice a, vertice b)	: arista
+		LeerGrafo (String archivo)	: void
+		aletorio (int N)	: void
+		toString ()	: String

Figura C.6: Clase Grafo

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

AlgoritmoBR

	Atributo	Tipo	Valor inicial
*	VGi	: ArrayList<vertice>	= new ArrayList<> ()
*	VGe	: ArrayList<vertice>	= new ArrayList<> ()
*	E_no_VGe	: ArrayList<arista>	= new ArrayList<> ()
*	E_no_VGi	: ArrayList<arista>	= new ArrayList<> ()
*	arrangement	: ArrayList<rectaDual>	= new ArrayList<> ()
+	tiempoEjecucion	: long	= 0
+	inicio	: long	
+	fin	: long	
+	tiempoParte1	: long	= 0
+	tiempoParte2	: long	= 0
+	tiempoParte3	: long	= 0
+	tiempoParte4	: long	= 0
+	tiempoParte5	: long	= 0

	Método	Retorno
+	BR (Grafo R, Grafo B)	: void
+	setV (ArrayList<vertice> poligono, ArrayList<vertice> puntos)	: void
+	estaDentro (ArrayList<vertice> poligono, vertice punto)	: boolean
+	mergesort (ArrayList<vertice> data, ArrayList<vertice> origin, int first, int n, ArrayList<Integer> indice)	: void
-	merge (ArrayList<vertice> data, ArrayList<vertice> origin, int first, int n1, int n2, ArrayList<Integer> indice)	: void
+	estaIzquierda (vertice A, vertice B, vertice C)	: boolean
-	tangentes (ArrayList<vertice> VGe, ArrayList<vertice> CH)	: ArrayList<tangente>
+	mergesort (ArrayList<tangente> data, int first, int n)	: void
-	merge (ArrayList<tangente> data, int first, int n1, int n2)	: void
-	OrdenRotacional (ArrayList<tangente> origen)	: ArrayList<tangente>
+	obtenerEGVe (ArrayList<tangente> OR, ArrayList<vertice> CH, Grafo B)	: ArrayList<arista>
+	obtenerEGi (ArrayList<vertice> R, ArrayList<vertice> VGi, ArrayList<vertice> CH)	: void
+	getOR (ArrayList<dualRojo> bj, ArrayList<dualRojo> bk)	: ArrayList<dualRojo>
+	regionActual (vertice pto, ArrayList<regionConvexa> regiones)	: int
+	crearArrangement (ArrayList<vertice> R, ArrayList<vertice> VGi)	: void
+	indexA (vertice v)	: int
+	mergesort2 (ArrayList<dualRojo> data, int first, int n)	: void
-	merge2 (ArrayList<dualRojo> data, int first, int n1, int n2)	: void

Figura C.7: Clase AlgoritmoBR

arista

	Atributo	Tipo
-	desde	: vertice
-	hasta	: vertice
-	pendiente	: double

	Estereotipo	Método	Retorno
+	<<Constructor>>	arista ()	
+	<<Constructor>>	arista (vertice d, vertice h)	
+		setArista (vertice d, vertice h)	: void
+		setDesde (vertice d)	: void
+		setHasta (vertice h)	: void
+		getDesde ()	: vertice
+		getHasta ()	: vertice
+		getM ()	: double
+		m (vertice p1, vertice p2)	: double
+		toString ()	: String

Figura C.8: Clase arista

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

vertice

-		x	: double
-		y	: double
+	<<Constructor>>	vertice ()	
+	<<Constructor>>	vertice (double x, double y)	
+		getX ()	: double
+		getY ()	: double
+		setVertice (double X, double Y)	: void
+		toString ()	: String

Figura C.9: Clase vertice

tangente

-		p	: vertice
-		tan_izq	: vertice
-		tan_der	: vertice
-		m_izq	: double
-		m_der	: double
-		m_izqesInfinita	: boolean
-		m_deresInfinita	: boolean
+	<<Constructor>>	tangente ()	
+		getP ()	: vertice
+		getTan_izq ()	: vertice
+		getTan ()	: vertice
+		getTan_der ()	: vertice
+		getM_izq ()	: double
+		getM_der ()	: double
+		getM ()	: double
+		setP (vertice e)	: void
+		setTan_izq (vertice e)	: void
+		setTan_der (vertice e)	: void
+		setM_izq (double e)	: void
+		setM_der (double e)	: void
+		m (vertice p1, vertice p2)	: double
+		estaCompleto ()	: boolean
+		m_izqesInfinita ()	: boolean
+		m_deresInfinita ()	: boolean
+		esInfinita ()	: boolean
+		toString ()	: String
+		toString (String m)	: String

Figura C.10: Clase tangente

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
CaliperRotatorio
* parAntipodal : arista
* dentro       : ArrayList<vertice>
* d            : int
* h            : int
+ <<Constructor>> CaliperRotatorio ()
+ ingresa (vertice e)                                         : void
+ elementosDentro ()                                          : int
+ saleDentro (int indice)                                     : void
+ getElementoDentro (int indice)                              : vertice
+ estaDentro (vertice x, tangente OR)                         : boolean
+ girar (tangente OR, ArrayList<vertice> CH)                  : void
+ buscarAntipodal (vertice desde, ArrayList<vertice> CH)      : void
+ aceptaCaliper (tangente OR, ArrayList<vertice> CH)          : boolean
+ obtenerY (double x, vertice A, double m)                    : double
+ estaIzquierda (vertice A, vertice B, vertice C)             : boolean
+ toString ()                                                 : String
+ intersectaCon (arista aIntersectar)                         : boolean
- paralelo (arista aIntersectar)                              : boolean
- paralelo (arista aIntersectar, vertice p)                   : boolean
- colineal (vertice a, vertice b, vertice c)                  : boolean
- area (vertice a, vertice b, vertice c)                      : double
- asignar (vertice p, vertice a)                              : void
- entre (vertice a, vertice b, vertice c)                     : boolean
```

Figura C.11: Clase Caliper Rotatorio

```
dualRojo
- ptoRojo   : int
+ interAzul : vertice
- Rojo      : vertice
+ <<Constructor>> dualRojo (int R, vertice rojo, vertice inter)
+ <<Getter>>      getRojo ()      : int
+                 getInter ()     : vertice
+                 toString ()     : String
```

Figura C.12: Clase dualRojo

```
rectaDual
- ptoAzul   : vertice
- ptosRojos : ArrayList<dualRojo>   = new ArrayList<> ()
+ <<Constructor>> rectaDual (vertice B, ArrayList<vertice> R)
+                 inter (vertice P, vertice U)       : vertice
+                 getAzul ()                         : vertice
+                 getRojo ()                         : ArrayList<dualRojo>
+                 toString ()                        : String
```

Figura C.13: Clase rectaDual

```
regionConvexa
+ region  : ArrayList<vertice>
+ puntos  : ArrayList<vertice>
+ <<Constructor>> regionConvexa ()
+                 addRegion (ArrayList<vertice> B)   : void
+                 addPunto (vertice A)               : void
+                 reset ()                           : void
+                 esVacia ()                         : boolean
+                 estaDentro (vertice punto)         : boolean
```

Figura C.14: Clase regionConvexa

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Apéndice D

CÓDIGO DE LA IMPLEMENTACIÓN

En este apartado se adjuntará todo el código java resultante de la implementación.

Código D.1: Grafo.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package algoritmobr;

import java.io.*;
import java.util.ArrayList;
import java.util.Stack;
import javax.swing.JOptionPane;

/**
 *
 * @author Joel
 * @author Cristian
 */
public class Grafo {
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
public ArrayList<vertice> vertices;
public ArrayList<arista> aristas;
String nombre;

/**
 * Constructor de un Grafo, da memoria a los arrayList de vertices y ↩
     aristas y un nombre al grafo.
 */
public Grafo() {
    this.vertices = new ArrayList<>();
    this.aristas = new ArrayList<>();
    this.nombre = "AUX";
}

/**
 * Constructor de un Grafo, da memoria a los arrayList de vertices y ↩
     aristas y un dado por el m todo nombre al grafo.
 * @param nombre
 */
public Grafo(String nombre)
{
    this.vertices = new ArrayList<>();
    this.aristas = new ArrayList<>();
    this.nombre = nombre;
}

/**
 * A ade un vertice al grafo
 * @param x - Coordenada X
 * @param y - Coordenada Y
 */
public void addVertice(Double x, Double y)
{
    vertice a = new vertice(x,y);
    vertices.add(a);
}

/**
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
     * A ade una Arista al Grafo a traves de las referencias a de dos ↩
        vertices (arista adireccional)
     * @param desde - primer vertice
     * @param hasta - segundo vertice
     */
    public void addArista(vertice desde, vertice hasta)
    {
        arista a = new arista(desde,hasta);
        aristas.add(a);
    }

    /**
     * A ade una arista al Grafo a traves de indices de los vertices ↩
        involucrados
     * @param desde - Indice del primer vertice
     * @param hasta - Indice del segundo vertice
     */
    public void addAristaIndice(int desde, int hasta)
    {
        arista a = new arista(vertices.get(desde),vertices.get(hasta));
        aristas.add(a);
    }


    public boolean esVacio()
    {
        if(vertices.isEmpty())
            return true;
        else
            return false;
    }

    //m todos
    //Graham Scan O(n log n)
    public ArrayList<vertice> GrahamScan(){
        ArrayList <vertice> Ch = new ArrayList();
        Stack<vertice> P=CH();
        while(!P.empty())
            Ch.add(0,P.pop());
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        return Ch;
    }

    public Stack <vertice> CH(){
        ArrayList<vertice> puntos = new ArrayList<>();
        Stack <vertice> P = new Stack <vertice>();
        vertice min;
        puntos=(ArrayList<vertice>) vertices.clone();
        min = PuntoMenorOrdenada(puntos);
        puntos = OrdenarAngularmente(puntos, min);
        P.push(puntos.get(0));
        P.push(puntos.get(1));
        P.push(puntos.get(2));
        int i = 3;
        while(i<puntos.size()){
            vertice t= new vertice();
            t = P.pop();
            if(estaIzquierda(P.peek(), t, puntos.get(i)))
            {
                P.push(t);
                P.push(puntos.get(i));
                i++;
            }
        }
        return P;
    }
    //O(n)
    public vertice PuntoMenorOrdenada(ArrayList <vertice> puntos){
        vertice menor;
        menor=puntos.get(0);
        for(int i=1; i<puntos.size(); i++){
            if(puntos.get(i).getY()<menor.getY()||((puntos.get(i).getY()==↩
               menor.getY())&&(puntos.get(i).getX()>menor.getX())))
                menor=puntos.get(i);
        }
        return menor;
    }

    //Cambiar el m todo de ordenamiento a MERGE SORT  O(n log n)
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
    public ArrayList<vertice> OrdenarAngularmente(ArrayList<vertice> S, ↩
        vertice m){
            //Ordenamiento angular
            S.remove(S.indexOf(m));
            mergesort(S, 0, S.size(), m);
            S.add(0, m);
            return S;
    }

    public void mergesort(ArrayList<vertice> data, int first, int n, ↩
        vertice m)
   {
      int n1; // Size of the first half of the array
      int n2; // Size of the second half of the array

      if (n > 1)
      {
         // Compute sizes of the two halves
         n1 = n / 2;
         n2 = n - n1;

         mergesort(data, first, n1, m);       // Sort data[first] through ↩
            data[first+n1-1]
         mergesort(data, first + n1, n2, m); // Sort data[first+n1] to the↩
             end

         // Merge the two sorted halves.
         merge(data, first, n1, n2, m);
      }
   }

   private void merge(ArrayList<vertice> data, int first, int n1, int n2, ↩
      vertice m)
   {
      ArrayList<vertice> temp = new ArrayList(); // Allocate the temporary↩
          array
      int copied  = 0; // Number of elements copied from data to temp
      int copied1 = 0; // Number copied from the first half of data
      int copied2 = 0; // Number copied from the second half of data
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
    int i;          // Array index to copy from temp back into data

    // Merge elements, copying from two halves of data to the temporary ↩
       array.
    while ((copied1 < n1) && (copied2 < n2))
    {
       if (Angulo(m,data.get(first + copied1)) < Angulo(m, data.get(↩
          first + n1 + copied2)))
          temp.add(copied++,data.get(first + (copied1++)));
       else
          temp.add(copied++,data.get(first + n1 + (copied2++)));
    }
    // Copy any remaining entries in the left and right subarrays.
    while (copied1 < n1)
       temp.add(copied++,data.get(first + (copied1++)));
    while (copied2 < n2)
       temp.add(copied++,data.get(first + n1 + (copied2++)));

    // Copy from temp back to the data array.
    for (i = 0; i < n1+n2; i++)
       data.set(first + i, temp.get(i));
 }

 public double Angulo(vertice A, vertice B){
      double my, mx, mb, angulo;
      my = B.getY()-A.getY();
      mx = B.getX()-A.getX();
      if(mx==0){
          if(my>0) return 90;
          else if(my<0) return 270;
          else {
              System.out.printf("\nEl punto A y C son los mismos\n");
              return 0;
          }
      }
      else if(my==0){
          if(mx>0) return 0;
          else return 180;
      }
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        else{
            mb=my/mx;
            angulo=Math.atan(mb)*(180/Math.PI);
            if(mx>0){
                if(my<0) return angulo + 360;
                else return angulo;
            }
            else return angulo + 180;
        }
    }
    public boolean estaIzquierda(vertice A, vertice B, vertice C){
        if((((B.getX()-A.getX())*(C.getY()-A.getY()))-((B.getY()-A.getY())↩
            *(C.getX()-A.getX())))>0)
            return true;
        else return false;
    }


    /**
     * Consulta si los vertices involucrados forman una arista registrada ↩
        en el grafo
     * @param a - primer vertice
     * @param b - segundo vertice
     * @return la arista consultada o null si no est  registrada
     */
    public arista esArista(vertice a, vertice b)
    {
        for(int i=0; i< aristas.size();i++)
        {
            if(aristas.get(i).getDesde().equals(a) && aristas.get(i).↩
                getHasta().equals(b))
                return aristas.get(i);
            if(aristas.get(i).getDesde().equals(b) && aristas.get(i).↩
                getHasta().equals(a))
                return aristas.get(i);
        }
        return null;
    }
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```java
/**
 * Lee los datos de v rtices y aristas del Grafo mediante la lectura ↩
    de un archivo.
 * @param archivo - nombre del archivo al cual se debe acceder para la↩
     lectura.
 */
public void LeerGrafo(String archivo)
{
    File gf = null;
    FileReader fr = null;
    BufferedReader br = null;

    try {
       // Apertura del fichero y creacion de BufferedReader para poder
       // hacer una lectura comoda (disponer del metodo readLine()).
       gf = new File (archivo);
       fr = new FileReader (gf);
       br = new BufferedReader(fr);

       // Lectura del fichero
       String linea;
       double x, y;
       int d, h;
       while((linea=br.readLine())!=null)
       {
           if("<vertices>".equals(linea)){
               linea=br.readLine();

               while(! "<\\vertices>".equals(linea)){

                   x=Double.parseDouble(linea.substring(0, linea.↩
                       indexOf(" ")));
                   y=Double.parseDouble(linea.substring(linea.indexOf(↩
                       " ")));

                   addVertice(x,y);
                   linea=br.readLine();
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
                    }
                }

                if("<aristas>".equals(linea)){
                    linea=br.readLine();

                    while(! "<\\aristas>".equals(linea)){

                        d=(int) Double.parseDouble(linea.substring(0, linea↩
                            .indexOf(" ")));
                        h=(int) Double.parseDouble(linea.substring(linea.↩
                            indexOf(" ")));

                        addAristaIndice(d,h);
                        linea=br.readLine();
                    }
                }

            }
        }
        catch(Exception e){
            //e.printStackTrace();
                JOptionPane.showMessageDialog(null, "ERROR, NO es posible ↩
                    abrir el archivo: "+archivo);
        }finally{
            // En el finally cerramos el fichero, para asegurarnos
            // que se cierra tanto si todo va bien como si salta
            // una excepcion.
            try{
                if( null != fr ){
                    fr.close();
                }
            }catch (Exception e2){
                //e2.printStackTrace();
                JOptionPane.showMessageDialog(null, "ERROR, NO es posible ↩
                    abrir el archivo: "+archivo);
            }
        }
    }
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
/**
 * Se crear  un grafo con coordenadas aleatorias, s lo se pide un
 */
public void aletorio(int N)
{
    try{
    int i, d,h;
    Double x,y;
    for(i=0;i<N;i++)
    {
        do{
           x=Math.random()*900;
           y=Math.random()*625;
        }while(vertices.contains(new vertice(x,y)));
        addVertice(x,y);
    }
    int dim = (int)(Math.random()*(((N*(N-1))/2)-(N-1)))+(N-1);

    for(i=0; i<N-1; i++)
        this.addAristaIndice(i, i+1);
    for(i=N-1;i<dim;i++)
    {
        do{
            d=(int) (Math.random()*(N-1))+0;
            h=(int) (Math.random()*(N-1))+0;

            if(d==h || esArista((vertice) vertices.get(d), (vertice) ↩
                vertices.get(h))!=null)
                 d++;
            //System.out.println("hola "+d+" "+h);
        }while(d==h || esArista((vertice) vertices.get(d), (vertice) ↩
            vertices.get(h))!=null);
        addAristaIndice(d,h);
    }

}catch (Exception ex) {
            JOptionPane.showMessageDialog(null,"ERROR, Cantidad ↩
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
                    m xima  Excedida");
            }
    }

    public void eliminarNodo(vertice v){
        for(int i=0; i<this.vertices.size(); i++){
            if(this.vertices.get(i).equals(v)){
                for(int j=0; j<this.aristas.size(); j++){
                    if(this.aristas.get(j).getDesde().equals(v)||this.↩
                        aristas.get(j).getHasta().equals(v)){
                         this.eliminarArista(this.aristas.get(j));
                    }
                }
                this.vertices.remove(i);
            }
        }
    }

    public void eliminarArista(arista a){
        arista aux = this.esArista(a.getDesde(), a.getHasta());
        if(aux!=null){
            this.aristas.remove(a);
        }
    }


    /**
     * se encarga de mostrar el contenido del grafo
     * @return un String con la informaci n del Grafo
     */
    public String toString()
    {
        StringBuilder msn = new StringBuilder();
        msn.append("Grafo "+nombre+": \n vertices:"+vertices.toString()+"\↩
            n aristas:"+aristas.toString());
        return msn.toString();
    }
}
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Código D.2: vertice.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package algoritmobr;
/**
 * @author Joel
 * @author Cristian
 */
public class vertice {
        //ver posibilidad de agregar un ID
        private double x;
        private double y;

        public vertice(){
            x = 0;
            y = 0;
        }

        public vertice(double x, double y){
            this.x = x;
            this.y = y;
        }

        public vertice(vertice x){
            if(x!=null)this.setVertice(x.getX(), x.getY());
        }

        public double getX(){
            return x;
        }
        public double getY(){
            return y;
        }
        public void setVertice(double X, double Y)
        {
            x=X;
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
            y=Y;
        }
        public String toString()
        {
            String m;
            m="(";
            if(x-((int)x)==0)
                m=m+ String.format("%.0f", x);
            else
                m=m+ String.format("%.2f", x);

            m=m+",";
            if(y-((int)y)==0)
                m=m+ String.format("%.0f", y);
            else
                m=m+ String.format("%.2f", y);
            return m+")";
        }

        @Override
        public boolean equals(Object o){
            if(o instanceof vertice){
                if(this.x == ((vertice)o).getX() && this.y == ((vertice)o↩
                    ).getY())
                    return true;
                else
                    return false;
            }

            return false;
        }
}
```

Código D.3: arista.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
 */
package algoritmobr;
/**
 * @author Joel
 * @author Cristian
 */
public class arista {
    private vertice desde;
    private vertice hasta;
    private double pendiente;

    public arista(){

    }

    public arista(vertice d, vertice h){
            this.desde = d;
            this.hasta = h;
    }
    public void setArista(arista X){
        this.setArista(X.desde, X.hasta);
    }

    public void setArista(vertice d, vertice h)
    {
        this.desde = d;
        this.hasta = h;
        this.pendiente = m(desde,hasta);
    }
    public void setDesde(vertice d)
    {
        this.desde=d;
        //this.pendiente = m(desde,hasta);
    }
    public void setHasta(vertice h)
    {
        this.hasta=h;
        //this.pendiente = m(desde,hasta);
    }
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
    public vertice getDesde()
    {
        return desde;
    }
    public vertice getHasta()
    {
        return hasta;
    }
    public double getM()
    {
        this.pendiente = m(desde,hasta);
        return pendiente;
    }



    public double m(vertice p1, vertice p2)
    {
        double m = 0;
        if(p2.getX()-p1.getX()!=0)
            m=(double)( (p2.getY()-p1.getY())/(p2.getX()-p1.getX()) );
        else
            m=Double.POSITIVE_INFINITY;
        if(m == -0 || m== 0) m= 0;

        return m;
    }

    public String toString()
          {
              return "["+desde.toString()+"-"+hasta.toString()+"]";
          }
}
```

Código D.4: AlgoritmoBR.java

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package algoritmobr;

/**
 *
 * @author Cristian
 */
import static interfaz.Interfaz.B;
import static interfaz.Interfaz.R;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.util.Stack;
import java.util.ArrayList;
import java.awt.BasicStroke;

public class AlgoritmoBR {

    /**
     * @param args the command line arguments
     */
    static ArrayList<vertice> VGi;
    ArrayList<vertice> VGe;
    ArrayList<arista> E_no_VGe;
    ArrayList<arista> E_no_VGi;
    ArrayList<arista> uv;
    ArrayList<rectaDual> arrangement;
    static ArrayList<regionConvexa> particiones;
    ArrayList<vertice> ConvexHull;
    long tiempoEjecucion = 0, inicio, fin, tiempoParte1 = 0, tiempoParte2 ↩
       = 0, tiempoParte3 = 0, tiempoParte4 = 0, tiempoParte5 = 0;

    public void init() {
        VGi = new ArrayList<>();
        VGe = new ArrayList<>();
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        E_no_VGe = new ArrayList<>();
        E_no_VGi = new ArrayList<>();
        uv = new ArrayList<>();
        arrangement = new ArrayList<>();
        particiones = new ArrayList<>();
        ConvexHull = new ArrayList<>();
        tiempoEjecucion = 0;
        inicio = 0;
        fin = 0;
        tiempoParte1 = 0;
        tiempoParte2 = 0;
        tiempoParte3 = 0;
        tiempoParte4 = 0;
        tiempoParte5 = 0;
    }

    /**
     * Este m todo permite reportar a todos los segmentos de un grafo (B)↩
         que se
     * intersecan por lo menos con algun segmento de un grafo distinto (R)↩
         en
     * complejidad de tiempo O(n^2) MENTIRA!!!! xD
     *
     * @param R - Grafo Rojo (RED)
     * @param B - Grafo Azul (BLUE)
     */
    public void BR(Grafo R, Grafo B) {
        init();
        //Grafos en consulta
        //interfaz.Interfaz.jTextArea1.append("\n" + R.toString() + "\n");
        //interfaz.Interfaz.jTextArea1.append("\n" + B.toString() + "\n");
        //Parte 1
        inicio = System.nanoTime();
        ConvexHull = R.GrahamScan();
        fin = System.nanoTime();
        tiempoParte1 = fin - inicio;

        //interfaz.Interfaz.jTextArea1.append("\nCH(R):\n");
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
//interfaz.Interfaz.jTextArea1.append(ConvexHull.toString() + " ")↩
    ;
//interfaz.Interfaz.jTextArea1.append("\n");


inicio = System.nanoTime();
setV(ConvexHull, B.vertices);
fin = System.nanoTime();
tiempoParte1 = tiempoParte1 + (fin - inicio);

//interfaz.Interfaz.jTextArea1.append("\nV(Gi):\n");
//interfaz.Interfaz.jTextArea1.append(VGi.toString());
//interfaz.Interfaz.jTextArea1.append("\nV(Ge):\n");
//interfaz.Interfaz.jTextArea1.append(VGe.toString());
interfaz.Interfaz.jTextArea1.append("\nTiempo de Ejecuci n Parte ↩
    1: " + (float) tiempoParte1 / 1000000 + "(ms)");
interfaz.Interfaz.jTextArea1.append("\n\n\n");


//Parte 2
// (a)
if (!VGe.isEmpty()) {
    inicio = System.nanoTime();

    ArrayList<tangente> tngt = new ArrayList();
    tngt = tangentes(VGe, ConvexHull);
    tngt = OrdenRotacional(tngt);

    // (b)

    E_no_VGe = obtenerEGVe(tngt, ConvexHull, B);
    fin = System.nanoTime();

    tiempoParte2 = fin - inicio;
    interfaz.Interfaz.jTextArea1.append("\n\nLos Segmentos ↩
        Externos: " + E_no_VGe.size());
//  interfaz.Interfaz.jTextArea1.append("\n\nLos Segmentos ↩
        Externos: " + E_no_VGe.toString());
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
            interfaz.Interfaz.jTextArea1.append("\nTiempo de Ejecuci n ↩
                Parte 2: " + (float) tiempoParte2 / 1000000 + "(ms)");
        }

        //Parte 3
        if (!VGi.isEmpty()) {

            inicio = System.nanoTime();
            E_no_VGi = obtenerEGi(this.VGi, ConvexHull, R, B);
            fin = System.nanoTime();

            tiempoParte3 = fin - inicio;
            interfaz.Interfaz.jTextArea1.append("\n\nLos Segmentos ↩
                Internos: " + E_no_VGi.size());
            //interfaz.Interfaz.jTextArea1.append("\n\nLos Segmentos ↩
                Internos: " + E_no_VGi.toString());
            interfaz.Interfaz.jTextArea1.append("\nTiempo de Ejecuci n ↩
                Parte 3: " + (float) tiempoParte3 / 1000000 + "(ms)");
        }

        //Parte 4

        inicio = System.nanoTime();
        uv = obtenerUv(R, B);
        fin = System.nanoTime();

        tiempoParte4 = fin - inicio;
        interfaz.Interfaz.jTextArea1.append("\n\nLos Segmentos Mixtos: " +↩
             uv.size());
        //interfaz.Interfaz.jTextArea1.append("\n\nLos Segmentos Mixtos: "↩
            + uv.toString());
        interfaz.Interfaz.jTextArea1.append("\nTiempo de Ejecuci n Parte ↩
            4: " + (float) tiempoParte4/1000000  + "(ms)");

        tiempoEjecucion=tiempoParte1+tiempoParte2+tiempoParte3+↩
            tiempoParte4;
        interfaz.Interfaz.jTextArea1.append("\n\n\n\nTiempo de Ejecuci n ↩
            Parte 1: " + (float) tiempoParte1 / 1000000 + "(ms)");
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        interfaz.Interfaz.jTextArea1.append("\nTiempo de Ejecuci n Parte ↩
            2: " + (float) tiempoParte2 / 1000000 + "(ms)");
        interfaz.Interfaz.jTextArea1.append("\nTiempo de Ejecuci n Parte ↩
            3: " + (float) tiempoParte3 / 1000000 + "(ms)");
        interfaz.Interfaz.jTextArea1.append("\nTiempo de Ejecuci n Parte ↩
            4: " + (float) tiempoParte4 / 1000000 + "(ms)");
        interfaz.Interfaz.jTextArea1.append("\n\nTiempo de Ejecuci n ↩
            Total: " + (float) tiempoEjecucion / 1000000 + "(ms)");

        //Parte 5
        //Ejecuci n del algoritmo con Conjuntos
        if(interfaz.Interfaz.frameDibujo.isVisible()){
            Graphics g = interfaz.Interfaz.frameDibujo.getGraphics();
            Graphics2D g2=(Graphics2D) g;
            if(B.nombre=="B")g2.setColor(Color.BLUE);
            if(B.nombre=="R")g2.setColor(Color.RED);
            g2.setStroke(new BasicStroke(4));
            for(int i=0; i<this.E_no_VGe.size();i++)
            g2.drawLine((int)this.E_no_VGe.get(i).getDesde().getX()+10, (↩
                int)this.E_no_VGe.get(i).getDesde().getY()+25, (int)this.↩
                E_no_VGe.get(i).getHasta().getX()+10, (int)this.E_no_VGe.↩
                get(i).getHasta().getY()+25);
            for(int i=0; i<this.E_no_VGi.size();i++)
            g2.drawLine((int)this.E_no_VGi.get(i).getDesde().getX()+10, (↩
                int)this.E_no_VGi.get(i).getDesde().getY()+25, (int)this.↩
                E_no_VGi.get(i).getHasta().getX()+10, (int)this.E_no_VGi.↩
                get(i).getHasta().getY()+25);
            for(int i=0; i<this.uv.size();i++)
            g2.drawLine((int)this.uv.get(i).getDesde().getX()+10, (int)↩
                this.uv.get(i).getDesde().getY()+25, (int)this.uv.get(i).↩
                getHasta().getX()+10, (int)this.uv.get(i).getHasta().getY()↩
                +25);

        }

    }

    public void setV(ArrayList<vertice> poligono, ArrayList<vertice> ↩
        puntos) {
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        for (int i = 0; i < puntos.size(); i++) {
            if (estaDentro(poligono, puntos.get(i))) {
                this.VGi.add(puntos.get(i));
            } else {
                this.VGe.add(puntos.get(i));
            }
        }
    }

    public boolean estaDentro(ArrayList<vertice> poligono, vertice punto) ↩
        {
        int counter = 0;
        int i;
        int n = poligono.size();
        double xinters;
        vertice p1, p2;
        p1 = poligono.get(0);
        for (i = 1; i <= n; i++) {
            p2 = poligono.get(i % n);
            if (punto.getY() > Math.min(p1.getY(), p2.getY())) {
                if (punto.getY() <= Math.max(p1.getY(), p2.getY())) {
                    if (punto.getX() <= Math.max(p1.getX(), p2.getX())) {
                        if (p1.getY() != p2.getY()) {
                            xinters = (punto.getY() - p1.getY()) * (p2.↩
                                getX() - p1.getX()) / (p2.getY() - p1.getY↩
                                ()) + p1.getX();
                            //OBS: Si los puntos en linea se cuentan ↩
                                dentro, se hace < xinters
                            if (p1.getX() == p2.getX() || punto.getX() <= ↩
                                xinters) {
                                 counter++;
                            }
                        }
                    }
                }
            }
            p1 = p2;
        }
        if (counter % 2 == 0) {
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
            return false;
        } else {
            return true;
        }
    }

    // para paso 2
    /**
     * m todo de ordenamiento optimizado, modificado para almacenar el ↩
        contenido
     * del ordenamiento en una Lista de indices
     *
     * @param data - Lista de datos
     * @param origin - Lista de datos originales
     * @param first - indice del primer elemento de la lista
     * @param n - tama o de la Lista
     * @param indice - Lista de indices resultante
     */
    public void mergesort(ArrayList<vertice> data, ArrayList<vertice> ↩
       origin, int first, int n, ArrayList<Integer> indice) {
        int n1; // Size of the first half of the array
        int n2; // Size of the second half of the array

        if (n > 1) {
            // Compute sizes of the two halves
            n1 = n / 2;
            n2 = n - n1;

            mergesort(data, origin, first, n1, indice);      // Sort data[↩
                first] through data[first+n1-1]
            mergesort(data, origin, first + n1, n2, indice); // Sort data[↩
                first+n1] to the end

            // Merge the two sorted halves.
            merge(data, origin, first, n1, n2, indice);
        }
    }

    /**
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
     * m todo de ordenamiento de apoyo al m todo mergesort
     *
     * @param data - Lista de datos
     * @param origin - Lista de datos Originales
     * @param first - el primero de la Lista
     * @param n1 - el inicio del segmento
     * @param n2 - el final del segmento
     * @param indice - la lista de los indices resultantes
     */
    private void merge(ArrayList<vertice> data, ArrayList<vertice> origin,↩
        int first, int n1, int n2, ArrayList<Integer> indice) {
        ArrayList<vertice> temp = new ArrayList(); // Allocate the ↩
            temporary array
        ArrayList<Integer> tempI = new ArrayList(); // Allocate the ↩
            temporary array
        int copied = 0; // Number of elements copied from data to temp
        int copied1 = 0; // Number copied from the first half of data
        int copied2 = 0; // Number copied from the second half of data
        int i;           // Array index to copy from temp back into data

        // Merge elements, copying from two halves of data to the ↩
            temporary array.
        while ((copied1 < n1) && (copied2 < n2)) {
            if (data.get(first + copied1).getX() < data.get(first + n1 + ↩
                copied2).getX()) {
                 temp.add(copied++, data.get(first + (copied1++)));
            } else {
                 temp.add(copied++, data.get(first + n1 + (copied2++)));
            }
        }
        // Copy any remaining entries in the left and right subarrays.
        while (copied1 < n1) {
            temp.add(copied++, data.get(first + (copied1++)));
        }
        while (copied2 < n2) {
            temp.add(copied++, data.get(first + n1 + (copied2++)));
        }

        // Copy from temp back to the data array.
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        for (i = 0; i < n1 + n2; i++) {
            data.set(first + i, temp.get(i));
            indice.set(first + i, origin.indexOf(temp.get(i)));
        }

    }

    /**
     * Consulta si un vertice dado est  o no a la izquierda de una recta
     *
     * @param A - primer vertice de la recta
     * @param B - segundo vertice de la recta
     * @param C - vertice a consulta
     * @return true si est  a la izquierda, false si no lo est .
     */
    public boolean estaIzquierda(vertice A, vertice B, vertice C) {
        if ((((B.getX() - A.getX()) * (C.getY() - A.getY())) - ((B.getY() ↩
           - A.getY()) * (C.getX() - A.getX()))) > 0) {
            return true;
        } else {
            return false;
        }
    }

    /**
     * Calcula los vertices tangententes del conjunto VGe con respecto a ↩
        CH.
     *
     * @param VGe - Lista del conjunto de vertices VGe
     * @param CH - Lista de vertices que componen el CH
     * @return la Lista de tangentes izquierdas y derechas de VGe con ↩
        repecto a
     * CH.
     */
    private ArrayList<tangente> tangentes(ArrayList<vertice> VGe, ↩
       ArrayList<vertice> CH) {

        ArrayList<tangente> tngt = new ArrayList();
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
int fila = 0;
int i, j, k;

ArrayList<Integer> indices = new ArrayList();

for (i = 0; i < CH.size(); i++) {
    indices.add(i);
}

for (i = 0; i < VGe.size(); i++) {
    tngt.add(new tangente());
}

mergesort((ArrayList<vertice>) CH.clone(), CH, 0, CH.size(), ↩
   indices);
//indices con las posiciones de los puntos ordenados por el eje X
double suma = 0;
for (i = 0; i < CH.size(); i++) {
    suma = suma + CH.get(indices.get(i)).getX();
}
suma = suma / i;

for (int p = 0; p < VGe.size(); p++) {
    //resguardos para los indices ordenados por eje X
    if (VGe.get(p).getX() <= suma) {
        for (i = 0; i < CH.size(); i++) {
            //resguardos para hacer los calculos en el array ↩
                lineal
            if (indices.get(i) == 0) {
                j = CH.size();
                k = indices.get(i);
            } else {
                if (indices.get(i) == CH.size() - 1) {
                    j = indices.get(i);
                    k = -1;
                } else {
                    j = indices.get(i);
                    k = indices.get(i);
                }
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
                }


                if (estaIzquierda(CH.get(j - 1), CH.get(indices.get(i)↩
                    ), VGe.get(p)) && !estaIzquierda(CH.get(indices.get↩
                    (i)), CH.get(k + 1), VGe.get(p))) {
                    tngt.get(fila).setP(VGe.get(p));
                    tngt.get(fila).setTan_der(CH.get(indices.get(i)));
                } else {
                    if (!estaIzquierda(CH.get(j - 1), CH.get(indices.↩
                        get(i)), VGe.get(p)) && estaIzquierda(CH.get(↩
                        indices.get(i)), CH.get(k + 1), VGe.get(p))) {
                        tngt.get(fila).setP(VGe.get(p));
                        tngt.get(fila).setTan_izq(CH.get(indices.get(i↩
                            )));
                    }
                }
                if (tngt.get(fila).estaCompleto()) {
                    i = CH.size();
                }
            }
        } else {
            for (i = CH.size() - 1; i >= 0; i--) {
                //resguardos para hacer los calculos en el array ↩
                    lineal
                if (indices.get(i) == 0) {
                    j = CH.size();
                    k = indices.get(i);
                } else {
                    if (indices.get(i) == CH.size() - 1) {
                        j = indices.get(i);
                        k = -1;
                    } else {
                        j = indices.get(i);
                        k = indices.get(i);
                    }
                }
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
                if (estaIzquierda(CH.get(j - 1), CH.get(indices.get(i)↩
                    ), VGe.get(p)) && !estaIzquierda(CH.get(indices.get↩
                    (i)), CH.get(k + 1), VGe.get(p))) {
                    tngt.get(fila).setP(VGe.get(p));
                    tngt.get(fila).setTan_der(CH.get(indices.get(i)));
                } else {
                    if (!estaIzquierda(CH.get(j - 1), CH.get(indices.↩
                        get(i)), VGe.get(p)) && estaIzquierda(CH.get(↩
                        indices.get(i)), CH.get(k + 1), VGe.get(p))) {
                        tngt.get(fila).setP(VGe.get(p));
                        tngt.get(fila).setTan_izq(CH.get(indices.get(i↩
                            )));
                    }
                }

                if (tngt.get(fila).estaCompleto()) {
                    i = -1;
                }
            }
        }
        if (tngt.get(fila).estaCompleto()) {
            fila++;
        }

    }
    return tngt;
}

/**
 * M todo de ordenamiento optimizado, modificado para ordenar una ↩
    lista de
 * tangentes
 *
 * @param data - Lista de datos
 * @param first - el indice del primer elemento
 * @param n - cantidad de elementos de la Lista
 */
public void mergesort(ArrayList<tangente> data, int first, int n) {
    int n1; // Size of the first half of the array
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        int n2; // Size of the second half of the array

        if (n > 1) {
            // Compute sizes of the two halves
            n1 = n / 2;
            n2 = n - n1;

            mergesort(data, first, n1);       // Sort data[first] through ↩
                data[first+n1-1]
            mergesort(data, first + n1, n2); // Sort data[first+n1] to the↩
                 end

            // Merge the two sorted halves.
            merge(data, first, n1, n2);
        }
    }

    /**
     * M todo de Ordenamiento Optimizado de apoyo a mergesort
     *
     * @param data - Lista de datos
     * @param first - el primer elemento de la Lista
     * @param n1 - indice de inicio del segmento
     * @param n2 - indice de termino del segmento
     */
    private void merge(ArrayList<tangente> data, int first, int n1, int n2↩
        ) {
        ArrayList<tangente> temp = new ArrayList(); // Allocate the ↩
            temporary array
        int copied = 0; // Number of elements copied from data to temp
        int copied1 = 0; // Number copied from the first half of data
        int copied2 = 0; // Number copied from the second half of data
        int i;           // Array index to copy from temp back into data

        // Merge elements, copying from two halves of data to the ↩
            temporary array.
        while ((copied1 < n1) && (copied2 < n2)) {
            if (data.get(first + copied1).getM() < data.get(first + n1 + ↩
                copied2).getM()) {
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
                if (data.get(first + copied1).getM() == 0 && data.get(↩
                    first + copied1).esInfinita()) {
                    temp.add(copied++, data.get(first + n1 + (copied2++))↩
                        ;
                } else {
                    temp.add(copied++, data.get(first + (copied1++)));
                }
            } else {
                if (data.get(first + n1 + copied2).getM() == 0 && data.get↩
                    (first + n1 + copied2).esInfinita()) {
                    temp.add(copied++, data.get(first + (copied1++)));
                } else {
                    temp.add(copied++, data.get(first + n1 + (copied2++))↩
                        ;
                }
            }
        }
        // Copy any remaining entries in the left and right subarrays.
        while (copied1 < n1) {
            temp.add(copied++, data.get(first + (copied1++)));
        }
        while (copied2 < n2) {
            temp.add(copied++, data.get(first + n1 + (copied2++)));
        }

        // Copy from temp back to the data array.
        for (i = 0; i < n1 + n2; i++) {
            data.set(first + i, temp.get(i));
        }

    }

    /**
     * M todo que obtiene el orden rotacional de VGe con respecto a CH, ↩
        en
     * concreto
     *
     * @param origen
     * @return
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
    */
   private ArrayList<tangente> OrdenRotacional(ArrayList<tangente> origen↩
       ) {
        //acomodo de los datos del arreglo
        //System.out.println("Origen (kisawea xd) = "+origen.toString());
        int n = origen.size();
        for (int i = 0; i < n; i++) {
            tangente a = new tangente();
            a.setP(origen.get(i).getP());
            a.setTan_izq(origen.get(i).getTan_der());
            origen.add(i + n, a);
        }
        //ordenamientos
        mergesort(origen, 0, origen.size());

        //reacomodo de lista (a partir de cero)
        n = origen.size();
        while (origen.get(0).getM() < 0) {
            origen.add(n, origen.get(0));
            origen.remove(0);
        }

        return origen;
    }

    /**
     * Este m todo realiza una comparaci n de los puntos externos con la
     * cerradura convexa para determinar los segmentos azules que ↩
        intersecan a
     * la cerradura convexa. para ello utiliza un c liper Rotatorio que ↩
        gira
     * bajo el orden rotacional dado determinando los puntos que estan de ↩
        un
     * lado al otro dentro de las l neas paralelas de soporte del ↩
        C liper
     * Rotatorio.
     *
     * @param OR Orden Rotacional de los Puntos Azules Externos con ↩
        respecto a
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
     * CH(R)
     * @param CH la Cerradura Convexa del Grafo Rojo
     * @param B El Grafo Azul.
     * @return Un Arreglo de los segmentos azules externos que atraviesan ↩
       a
     * CH(R).
     */
    public ArrayList<arista> obtenerEGVe(ArrayList<tangente> OR, ArrayList↩
        <vertice> CH, Grafo B) {
        ArrayList<arista> EGVe = new ArrayList();
        boolean mediaVuelta = false;

        CaliperRotatorio CR = new CaliperRotatorio();
        //busca caliper horizontal
        CR.buscarAntipodal(CH.get(0), CH);

        for (int i = 0; i < OR.size(); i++) {
            CR.ingresa(OR.get(i).getP());

            //mover caliper
            CR.girar(OR.get(i), CH);

            for (int k = 0; k < CR.elementosDentro() - 1; k++) {
                if (CR.estaDentro(CR.getElementoDentro(k), OR.get(i))) {
                    arista aIntersectar = B.esArista(CR.getElementoDentro(↩
                        k), CR.getElementoDentro(CR.elementosDentro() - 1))↩
                        ;
                    if (aIntersectar != null) {
                        if (CR.intersectaCon(aIntersectar)) {
                            if (!EGVe.contains(aIntersectar)) {
                                EGVe.add(aIntersectar);
                            }

                            if (CH.size() == 2) {
                                return EGVe;
                            }
                        }
                    }
                } else {
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
                CR.saleDentro(k);
                k--;
            }
        }

        if (i == OR.size() - 1 && mediaVuelta == false) {
            i = -1;
            mediaVuelta = true;
        }
    }
    return EGVe;
}

public ArrayList<arista> obtenerEGi(ArrayList<vertice> VGi, ArrayList<↩
    vertice> CH, Grafo R, Grafo B) {
    //(a)
    crearArrangement(R.vertices, VGi);
    int pos;
    this.particiones.add(new regionConvexa());
    this.particiones.get(0).addRegion(CH);
    this.particiones.get(0).addPunto(0);
    //(b)
    ArrayList<dualRojo> OR = new ArrayList<>();
    for (int k = 1; k < VGi.size(); k++) {
        //(i)
        pos = buscaRegion(VGi.get(k), this.particiones);
        //(ii)
        if (this.particiones.get(pos).esVacia()) {
            this.particiones.get(pos).addPunto(k);
        } else {
            OR.addAll(getOR(arrangement.get(particiones.get(pos).↩
                puntos.get(0)).getRojo(), arrangement.get(k).getRojo())↩
                );
            ArrayList<tangente> tngt = new ArrayList<>();
            for (int i = 0; i < OR.size(); i++) {
                tngt.add(new tangente());
                tngt.get(i).setP(OR.get(i).getROJO());
                tngt.get(i).setTan_izq(OR.get(i).getAzul());
                tngt.get(i).setM_izq(OR.get(i).getInter().getX());
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
                }
                OR.clear();
                ArrayList<vertice> blue = new ArrayList<>();
                blue.add(arrangement.get(particiones.get(pos).puntos.get↩
                    (0)).getAzul());
                blue.add(arrangement.get(k).getAzul());
                //(B)
                ArrayList<arista> div = new ArrayList<>();
                div = obtenerEGVe(tngt, blue, R);
                //(C)
                if (!div.isEmpty()) {
                    particiones.get(pos).addPunto(k);
                    particiones.get(pos).Particion(div.get(0).getDesde(), ↩
                        div.get(0).getHasta());
                    particiones.remove(pos);
                } else {
                    particiones.get(pos).addPunto(k);
                }
            }
        }
        ArrayList<arista> EGi = new ArrayList<>();
        for (int i = 0; i < VGi.size(); i++) {
            for (int j = i; j < VGi.size(); j++) {
                if ((buscaRegion(VGi.get(i), this.particiones) != ↩
                    buscaRegion(VGi.get(j), this.particiones)) && B.↩
                    esArista(VGi.get(i), VGi.get(j)) != null) {
                    EGi.add(new arista(VGi.get(i), VGi.get(j)));
                }
            }
        }
        this.particiones.clear();
        return EGi;
    }

    public ArrayList<arista> obtenerUv(Grafo R, Grafo B) {
        ArrayList<arista> uv = new ArrayList<>();
        ArrayList<arista> azulUv = new ArrayList<>();
        ArrayList<arista> rojoUv = new ArrayList<>();
        //obtenci n segmentos mixtos
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        for (int i = 0; i < VGe.size(); i++) {
            for (int j = 0; j < VGi.size(); j++) {
                if (B.esArista(VGe.get(i), VGi.get(j)) != null) {
                    azulUv.add(new arista(VGe.get(i), VGi.get(j)));
                }
            }
        }
        //obtencion segmentos rojos
        for (int i = 0; i < R.vertices.size(); i++) {
            for (int j = i; j < R.vertices.size(); j++) {
                if (R.esArista(R.vertices.get(i), R.vertices.get(j)) != ↩
                    null) {
                    rojoUv.add(new arista(R.vertices.get(i), R.vertices.↩
                        get(j)));
                }
            }
        }
        for (int i = 0; i < azulUv.size(); i++) {
            for (int j = 0; j < rojoUv.size(); j++) {
                if (!uv.contains(azulUv.get(i))) {
                    if ((estaIzquierda(azulUv.get(i).getDesde(), azulUv.↩
                        get(i).getHasta(), rojoUv.get(j).getDesde()) && !↩
                        estaIzquierda(azulUv.get(i).getDesde(), azulUv.get(↩
                        i).getHasta(), rojoUv.get(j).getHasta())) || (↩
                        estaIzquierda(azulUv.get(i).getDesde(), azulUv.get(↩
                        i).getHasta(), rojoUv.get(j).getHasta()) && !↩
                        estaIzquierda(azulUv.get(i).getDesde(), azulUv.get(↩
                        i).getHasta(), rojoUv.get(j).getDesde()))) {
                        if ((estaIzquierda(rojoUv.get(j).getDesde(), ↩
                            rojoUv.get(j).getHasta(), azulUv.get(i).↩
                            getDesde()) && !estaIzquierda(rojoUv.get(j).↩
                            getDesde(), rojoUv.get(j).getHasta(), azulUv.↩
                            get(i).getHasta())) || (estaIzquierda(rojoUv.↩
                            get(j).getDesde(), rojoUv.get(j).getHasta(), ↩
                            azulUv.get(i).getHasta()) && !estaIzquierda(↩
                            rojoUv.get(j).getDesde(), rojoUv.get(j).↩
                            getHasta(), azulUv.get(i).getDesde()))) {
                            uv.add(azulUv.get(i));
                        }
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
                    }
                }
            }
        }

        return uv;
    }

    public static int buscaRegion(vertice azul, ArrayList<regionConvexa> ↩
        particiones) {
        for (int i = 0; i < particiones.size(); i++) {
            if (particiones.get(i).Posicion(azul) == 1 || particiones.get(↩
                i).Posicion(azul) == 2) {
                return i;
            }
        }
        return 0;
    }

    public ArrayList<dualRojo> getOR(ArrayList<dualRojo> bj, ArrayList<↩
        dualRojo> bk) {
        ArrayList<dualRojo> OR = new ArrayList<>();
        ArrayList<dualRojo> Pos = new ArrayList<>();
        ArrayList<dualRojo> Neg = new ArrayList<>();
        ArrayList<dualRojo> IP = new ArrayList<>();
        ArrayList<dualRojo> IN = new ArrayList<>();
        for (int i = 0; i < bj.size(); i++) {
            if (bj.get(i).interAzul.getX() < 0) {
                if (bj.get(i).interAzul.getX() == Double.NEGATIVE_INFINITY↩
                    ) {
                    IN.add(bj.get(i));
                } else {
                    Neg.add(bj.get(i));
                }
            } else {
                if (bj.get(i).interAzul.getX() == Double.POSITIVE_INFINITY↩
                    ) {
                    IP.add(bj.get(i));
                } else {
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
                    Pos.add(bj.get(i));
                }
            }
        }
        for (int i = 0; i < bk.size(); i++) {
            if (bk.get(i).interAzul.getX() < 0) {
                if (bk.get(i).interAzul.getX() == Double.NEGATIVE_INFINITY↩
                    ) {
                    IN.add(bk.get(i));
                } else {
                    Neg.add(bk.get(i));
                }
            } else {
                if (bk.get(i).interAzul.getX() == Double.POSITIVE_INFINITY↩
                    ) {
                    IP.add(bk.get(i));
                } else {
                    Pos.add(bk.get(i));
                }
            }
        }
        mergesort2(Pos, 0, Pos.size());
        mergesort2(Neg, 0, Neg.size());
        OR.addAll(Neg);
        OR.addAll(IN);
        OR.addAll(IP);
        OR.addAll(Pos);



        //Ordenar OR
        return OR;
    }

    /**
     * M todo que crea un arreglo dual de l neas a partir de los puntos ↩
        rojos y
     * VGi
     *
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
     * @param R
     * @param VGi
     */
    public void crearArrangement(ArrayList<vertice> R, ArrayList<vertice> ↩
       VGi) {

        for (int i = 0; i < VGi.size(); i++) {
            rectaDual aux = new rectaDual(VGi.get(i), R);
            arrangement.add(aux);
        }
    }

    public void mergesort2(ArrayList<dualRojo> data, int first, int n) {
        int n1; // Size of the first half of the array
        int n2; // Size of the second half of the array

        if (n > 1) {
            // Compute sizes of the two halves
            n1 = n / 2;
            n2 = n - n1;

            mergesort2(data, first, n1);      // Sort data[first] through ↩
               data[first+n1-1]
            mergesort2(data, first + n1, n2); // Sort data[first+n1] to ↩
               the end

            // Merge the two sorted halves.
            merge2(data, first, n1, n2);
        }
    }

    private void merge2(ArrayList<dualRojo> data, int first, int n1, int ↩
       n2) {
        ArrayList<dualRojo> temp = new ArrayList(); // Allocate the ↩
           temporary array
        int copied = 0; // Number of elements copied from data to temp
        int copied1 = 0; // Number copied from the first half of data
        int copied2 = 0; // Number copied from the second half of data
        int i;           // Array index to copy from temp back into data
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        // Merge elements, copying from two halves of data to the ↩
            temporary array.
        while ((copied1 < n1) && (copied2 < n2)) {
            if ((data.get(first + copied1).getInter().getX() > data.get(↩
                first + n1 + copied2).getInter().getX()) || ((data.get(↩
                first + copied1).getInter().getX() == data.get(first + n1 +↩
                 copied2).getInter().getX()) && (data.get(first + copied1).↩
                getInter().getY() < data.get(first + n1 + copied2).getInter↩
                ().getY()))) {
                 temp.add(copied++, data.get(first + (copied1++)));
            } else {
                 temp.add(copied++, data.get(first + n1 + (copied2++)));
            }
        }
        // Copy any remaining entries in the left and right subarrays.
        while (copied1 < n1) {
            temp.add(copied++, data.get(first + (copied1++)));
        }
        while (copied2 < n2) {
            temp.add(copied++, data.get(first + n1 + (copied2++)));
        }

        // Copy from temp back to the data array.
        for (i = 0; i < n1 + n2; i++) {
            data.set(first + i, temp.get(i));
        }
    }

    public long gettiempo()
    {
        return tiempoEjecucion;
    }

    public void destruir() {
        E_no_VGe = null;
        E_no_VGi = null;
        VGe = null;
        VGi = null;
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        arrangement.clear();
        particiones.clear();
        uv.clear();
        tiempoEjecucion = 0;
        inicio = 0;
        fin = 0;
        tiempoParte1 = 0;
        tiempoParte2 = 0;
        tiempoParte3 = 0;
        tiempoParte4 = 0;
        tiempoParte5 = 0;
    }
}
```

Código D.5: CaliperRotatorio.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package algoritmobr;

import java.util.*;

/**
 * @author Joel
 * @author Cristian
 */
public class CaliperRotatorio {

    arista parAntipodal;
    ArrayList<vertice> dentro;
    ArrayList<tangente> OR;
    int d, h;

    /**
     * Constructor del caliper Rotatorio
     */
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
    public CaliperRotatorio() {
        parAntipodal = new arista();
        dentro = new ArrayList();
    }

    /**
     * Ingresa un elemento a la Lista de los elementos dentro del Caliper
     * Rotatorio
     *
     * @param e - elemento a ingresar dentro de los limites del caliper
     */
    public void ingresa(vertice e) {
        dentro.add(e);
    }

    /**
     * La cantidad de elementos que se encuentran dentro de las l neas ↩
        paralelas
     * del caliper Rotatorio en ese instante
     *
     * @return La cantidad de elementos en el Array dentro
     */
    public int elementosDentro() {
        return dentro.size();
    }

    /**
     * Elimina el elemento de la lista de los vertices que est n dentro ↩
        del
     * Caliper Rotatorio
     *
     * @param indice
     */
    public void saleDentro(int indice) {
        dentro.remove(indice);
    }

    /**
     * Se obtienen los elementos dentro del c liper en ese momento
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
     *
     * @param indice
     * @return El v rtice dado por la posici n en el Array
     */
    public vertice getElementoDentro(int indice) {
        return dentro.get(indice);
    }

    /**
     * se encarga de consultar si un elemento del tipo vertice se ↩
        encuentra
     * dentro de las l neas paralelas del Caliper Rotatorio en ese ↩
        instante
     *
     * @return Verdadero si el elemento se encuentra dentro
     */
    public boolean estaDentro(vertice x, tangente OR) {
        vertice di = new vertice();
        vertice hi = new vertice();
        di.setVertice(0, obtenerY(0, parAntipodal.getDesde(), OR.getM()));
        hi.setVertice(0, obtenerY(0, parAntipodal.getHasta(), OR.getM()));

        if (estaIzquierda(di, parAntipodal.getDesde(), x) && estaIzquierda↩
            (parAntipodal.getHasta(), hi, x)) {
            return true;
        } else {
            if (estaIzquierda(hi, parAntipodal.getHasta(), x) && ↩
                estaIzquierda(parAntipodal.getDesde(), di, x)) {
                return true;
            } else {
                return false;


            }
        }

    }

    /**
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
     * M todo realiza el giro de las l neas paralelas dejando al par ↩
        Antipodal
     * correcto a la pendiente del elemento del Orden Rotacional
     *
     */
    public void girar(tangente OR, ArrayList<vertice> CH) {
        boolean dh = false;
        while (!aceptaCaliper(OR, CH)) {
            if (OR.getTan() == parAntipodal.getDesde()) {
                avanzarAntipodalHasta(CH);
            } else {
                if (OR.getTan() == parAntipodal.getHasta()) {
                    avanzarAntipodalDesde(CH);
                } else {

                        if( resta(CH,OR,d)<=resta(CH,OR,h))
                            avanzarAntipodalDesde(CH);
                        else
                            avanzarAntipodalHasta(CH);
                }
            }
            if (h != d) {
                parAntipodal.setArista(CH.get(d), CH.get(h));
            }
        }

    }

    /**
     * Busca el primer par antipodal del caliper (un caliper horizontal)
     *
     * @param desde - primer vertice del Cierre Convexo (No necesariamente↩
         del
     * Oden Rotacional)
     * @param CH - la Lista del cierre Convexo
     */
    public void buscarAntipodal(vertice desde, ArrayList<vertice> CH) {
        parAntipodal.setDesde(desde);
        int p;
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        d = CH.indexOf(desde);
        if (CH.size() == 2) {
            h = 1;
        } else {
            if (CH.size() == 3) {
                if (CH.get(1).getY() > CH.get(2).getY()) {
                    h = 1;
                } else {
                    h = 2;
                }
            } else {

                if (d == CH.size() - 1) {
                    h = 0;
                } else {
                    h = d + 1;
                }

                //para no caer fuera de rango
                if (h == CH.size() - 1) {
                    p = 0;
                } else {
                    p = h + 1;
                }

                while (CH.get(p).getY() > CH.get(h).getY()) {
                    if (h == CH.size() - 1) {
                        h = 0;
                    } else {
                        h++;
                    }
                }
            }
        }
        parAntipodal.setHasta(CH.get(h));
    }

    /**
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
     * Consulta si los vertices dados son capaces de aceptar un caliper en↩
        la
     * direcci n de la pendiente dada
     *
     * @param OR − tangente del Orden Rotacional
     * @param CH − Lista del Cierre Convexo
     * @return true si logra acepta Caliper , false si no logra aceptar ↩
        caliper .
     */
    public boolean aceptaCaliper(tangente OR, ArrayList<vertice> CH) {

        if(CH.size()==2)
            return true;

        if (parAntipodal.getDesde().equals(parAntipodal.getHasta()))
            return false;

        boolean acepta = false;

        if (parAntipodal.getDesde().equals(OR.getTan())) {
            acepta=corroborarPunto(h, parAntipodal.getHasta(),OR, CH);
        }else{
            if (parAntipodal.getHasta().equals(OR.getTan())) {
                acepta=corroborarPunto(d, parAntipodal.getDesde(),OR, CH);
            }else{
                acepta=false;
            }
        }

        return acepta;

    }

    public boolean corroborarPunto(int i, vertice p, tangente OR, ↩
        ArrayList<vertice> CH)
    {
        vertice pfantasma = new vertice();
        vertice panterior = new vertice();
        vertice psiguiente = new vertice();
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        if(OR.esInfinita())
            pfantasma.setVertice(p.getX(), 0);
        else
            pfantasma.setVertice(p.getX()-1, obtenerY(p.getX()-1, p, OR.↩
                getM()));

        if(i==CH.size()-1)
        {
            panterior=CH.get(i-1);
            psiguiente=CH.get(0);
        }else{
            if(i==0)
            {
                panterior=CH.get(CH.size()-1);
                psiguiente=CH.get(i+1);
            }else{
                panterior=CH.get(i-1);
                psiguiente=CH.get(i+1);
            }
        }

        //todos los puntos ya estan asignados
        //pregunta crucial
        if((estaIzquierda(pfantasma,p,panterior) && estaIzquierda(↩
            pfantasma,p,psiguiente)) ||
                (estaDerecha(pfantasma,p,panterior)&&estaDerecha(pfantasma↩
                    ,p,psiguiente)))
        {
            return true;
        }else
            return false;
    }

    /**
     * Calcula la coordenada Y dada la coordenada x, un punto de la recta ↩
         y la
     * pendiente, formando la ecuaci n de la recta correspondiente.
     *
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
     * @param x - coordenada X
     * @param A - Vertice correspondiente a la recta
     * @param m - pendiente de la recta
     * @return la coordenada Y
     */
    public double obtenerY(double x, vertice A, double m) {
        double b = A.getY() - (m * A.getX());
        return ((m * x) + b);
    }

    /**
     * Consulta si un vertice dado est   o no a la izquierda de una recta
     *
     * @param A - primer vertice de la recta
     * @param B - segundo vertice de la recta
     * @param C - vertice a consulta
     * @return true si est   en la recta o en a la izquierda, false si no ↩
        lo
     * est  .
     */
    public boolean estaIzquierda(vertice A, vertice B, vertice C) {
        if ((((B.getX() - A.getX()) * (C.getY() - A.getY())) - ((B.getY() ↩
            - A.getY()) * (C.getX() - A.getX()))) >= 0) {
            return true;
        } else {
            return false;
        }
    }

        /**
     * Consulta si un vertice dado est   o no a la derecha de una recta
     *
     * @param A - primer vertice de la recta
     * @param B - segundo vertice de la recta
     * @param C - vertice a consulta
     * @return true si est   en la recta o en a la Derecha, false si no lo
     * est  .
     */
    public boolean estaDerecha(vertice A, vertice B, vertice C) {
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        if ((((B.getX() - A.getX()) * (C.getY() - A.getY())) - ((B.getY() ↩
            - A.getY()) * (C.getX() - A.getX()))) <= 0) {
            return true;
        } else {
            return false;
        }
    }

    /**
     * entrega informaci n del CALIPER ROTATORIO, especialmente el par ↩
         Antipodal
     *
     * @return un string con la arista del par Antipodal
     */
    public String toString() {
        return parAntipodal.toString();
    }

    /**
     * Consulta si dos segmentos se intersectan dados los vertices que los
     * conforman (comparando con el par antipodal del caliper)
     *
     * @param aIntersectar - arista a consultar
     * @return true si logran intersecarse, false si no se intersecan
     */
    public boolean intersectaCon(arista aIntersectar) {
        if ((estaIzquierda(parAntipodal.getDesde(), parAntipodal.getHasta↩
            (), aIntersectar.getDesde()) && !estaIzquierda(parAntipodal.↩
            getDesde(), parAntipodal.getHasta(), aIntersectar.getHasta())) ↩
            || (estaIzquierda(parAntipodal.getDesde(), parAntipodal.↩
            getHasta(), aIntersectar.getHasta()) && !estaIzquierda(↩
            parAntipodal.getDesde(), parAntipodal.getHasta(), aIntersectar.↩
            getDesde()))) {
                        if ((estaIzquierda(aIntersectar.getDesde(), ↩
                            aIntersectar.getHasta(), parAntipodal.getDesde↩
                            ()) && !estaIzquierda(aIntersectar.getDesde(), ↩
                            aIntersectar.getHasta(), parAntipodal.getHasta↩
                            ())) || (estaIzquierda(aIntersectar.getDesde(),↩
                             aIntersectar.getHasta(), parAntipodal.getHasta↩
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
                        ()) && !estaIzquierda(aIntersectar.getDesde(), ↩
                        aIntersectar.getHasta(), parAntipodal.getDesde↩
                        ())))) {
                         return true;
                     }
                     else return false;
                 }
        else return false;
        /*vertice p = new vertice();
        boolean i = false;

        double s, t, num, denom;

        denom = (parAntipodal.getDesde().getX() * (aIntersectar.getHasta()↩
            .getY() - aIntersectar.getDesde().getY()))
                + (parAntipodal.getHasta().getX() * (aIntersectar.getDesde↩
                    ().getY() - aIntersectar.getHasta().getY()))
                + (aIntersectar.getHasta().getX() * (parAntipodal.getHasta↩
                    ().getY() - parAntipodal.getDesde().getY()))
                + (aIntersectar.getDesde().getX() * (parAntipodal.getDesde↩
                    ().getY() - parAntipodal.getHasta().getY()));

        /*
         * Si el denominador es 0, entonces los segmentos son paralelos
         */
/*
        if (denom == 0) {
            return paralelo(aIntersectar);
        }

        num = (parAntipodal.getDesde().getX() * (aIntersectar.getHasta().↩
            getY() - aIntersectar.getDesde().getY()))
                + (aIntersectar.getDesde().getX() * (parAntipodal.getDesde↩
                    ().getY() - aIntersectar.getHasta().getY()))
                + (aIntersectar.getHasta().getX() * (aIntersectar.getDesde↩
                    ().getY() - parAntipodal.getDesde().getY()));

        if ((num == 0) || (num == denom)) {
            i = true;
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        }

        s = num / denom;

        num = -((parAntipodal.getDesde().getX() * (aIntersectar.getDesde()↩
            .getY() - parAntipodal.getHasta().getY()))
                + (parAntipodal.getHasta().getX() * (parAntipodal.getDesde↩
                    ().getY() - aIntersectar.getDesde().getY()))
                + (aIntersectar.getDesde().getX() * (parAntipodal.getHasta↩
                    ().getY() - parAntipodal.getDesde().getY())));

        if ((num == 0) || (num == denom)) {
            i = true;
        }

        t = num / denom;

        //p.setVertice(t, t);

        if ((0 < s) && (s < 1) && (0 < t) && (t < 1)) {
            i = true;
        } else if ((0 > s) && (s > 1) && (0 > t) && (t > 1)) {
            i = false;
        }

        return i;*/
    }

    /**
     * Consulta si se intersectan aristas verticales y paralelas
     *
     * @param aIntersectar - arista a comprobar
     * @return true si es alguna de las alternativas, false si no lo es
     */
    private boolean paralelo(arista aIntersectar) {
        if (!colineal(parAntipodal.getDesde(), parAntipodal.getHasta(), ↩
            aIntersectar.getDesde())) {
            return false;
        }
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        if (entre(parAntipodal.getDesde(), parAntipodal.getHasta(), ↩
            aIntersectar.getDesde())) {
            return true;
        }
        if (entre(parAntipodal.getDesde(), parAntipodal.getHasta(), ↩
            aIntersectar.getHasta())) {
            return true;
        }
        if (entre(aIntersectar.getDesde(), aIntersectar.getHasta(), ↩
            parAntipodal.getDesde())) {
            return true;
        }
        if (entre(aIntersectar.getDesde(), aIntersectar.getHasta(), ↩
            parAntipodal.getHasta())) {
            return true;
        }

        return false;
    }

    /**
     * Consulta si se intersectan aristas verticales y paralelas; adem s,
     * almacena el punto de intersecci n
     *
     * @param aIntersectar - arista a comprobar
     * @param p - v rtice a almacenar el punto de intersecci n.
     * @return true si es alguna de las alternativas, false si no lo es
     */
    private boolean paralelo(arista aIntersectar, vertice p) {
        if (!colineal(parAntipodal.getDesde(), parAntipodal.getHasta(), ↩
            aIntersectar.getDesde())) {
            return false;
        }

        if (entre(parAntipodal.getDesde(), parAntipodal.getHasta(), ↩
            aIntersectar.getDesde())) {
            asignar(p, aIntersectar.getDesde());
            return true;
        }
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        if (entre(parAntipodal.getDesde(), parAntipodal.getHasta(), ↩
            aIntersectar.getHasta())) {
            asignar(p, aIntersectar.getHasta());
            return true;
        }
        if (entre(aIntersectar.getDesde(), aIntersectar.getHasta(), ↩
            parAntipodal.getDesde())) {
            asignar(p, parAntipodal.getDesde());
            return true;
        }
        if (entre(aIntersectar.getDesde(), aIntersectar.getHasta(), ↩
            parAntipodal.getHasta())) {
            asignar(p, parAntipodal.getHasta());
            return true;
        }

        return false;
    }

    /**
     * consulta si tres puntos son colineales
     *
     * @param a
     * @param b
     * @param c
     * @return true si es cierto, false si no son colineales.
     */
    private boolean colineal(vertice a, vertice b, vertice c) {
        return area(a, b, c) == 0;
    }

    /**
     * obtiene el  rea  de un triangulo formado por tres vertices
     *
     * @param a
     * @param b
     * @param c
     * @return un double que es el  rea  de un tri ngulo
     */
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
    private double area(vertice a, vertice b, vertice c) {
        return (b.getX() - a.getX()) * (c.getY() - a.getY()) - (c.getX() -↩
            a.getX()) * (b.getY() - a.getY());
    }

    /**
     * Asigna a p los datos de a
     *
     * @param p
     * @param a
     */
    private void asignar(vertice p, vertice a) {
        p.setVertice(a.getX(), a.getY());
    }

    /**
     * Comprueba posibilidades de que un punto este en el segmento ↩
        evaluado
     *
     * @param a
     * @param b
     * @param c
     * @return true si cumple la condici n o false si no la cumple.
     */
    private boolean entre(vertice a, vertice b, vertice c) {
        /*
         * Si ab no es vertical, verificar cordenadas x, sino cordenadas y
         */
        if (a.getX() != b.getX()) {
            return ((a.getX() <= c.getX()) && (c.getX() <= b.getX())) || ↩
                ((a.getX() >= c.getX()) && (c.getX() >= b.getX()));
        } else {
            return ((a.getY() <= c.getY()) && (c.getY() <= b.getY())) || ↩
                ((a.getY() >= c.getY()) && (c.getY() >= b.getY()));
        }
    }

    public void cambiarAntipodal(ArrayList<vertice> CH, vertice d, vertice↩
        h) {
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        this.parAntipodal.setArista(d, h);
        this.d=CH.indexOf(d);
        this.h=CH.indexOf(h);
    }

    private void avanzarAntipodalDesde(ArrayList<vertice> CH)
    {
        if(this.d==CH.size()-1)
        {
            this.parAntipodal.setDesde(CH.get(0));
            this.d=0;
        }
        else
        {
            this.parAntipodal.setDesde(CH.get(d+1));
            this.d++;
        }
    }

    private void avanzarAntipodalHasta(ArrayList<vertice> CH)
    {
        if(this.h==CH.size()-1)
        {
            this.parAntipodal.setHasta(CH.get(0));
            this.h=0;
        }
        else
        {
            this.parAntipodal.setHasta(CH.get(h+1));
            this.h++;
        }
    }

    int resta(ArrayList<vertice> CH,tangente OR, int i)
    {
        int tan=CH.indexOf(OR.getTan());

        if( tan > i )
            return tan-i;
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        else
            return tan+(CH.size()-i+1);
    }

}
```

Código D.6: tangente.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package algoritmobr;

/**
 *
 * @author Joel
 */
public class tangente {
    private vertice p;
    private vertice tan_izq;
    private vertice tan_der;
    private double m_izq;
    private double m_der;
    private boolean m_izqesInfinita;
    private boolean m_deresInfinita;

    /**
     * Constructor para la tangente de los puntos externos al CH()
     */
    public tangente()
    {
        this.m_izqesInfinita=false;
        this.m_deresInfinita=false;
    }

    /**
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
     * entrega el punto externo comprometido
     * @return - El vertice de GVe
     */
    public vertice getP()
    {
        return p;
    }

    /**
     * entrega el vertice tangente por la izquierda
     * @return - el vertice tangente por la izquierda
     */
    public vertice getTan_izq()
    {
        return tan_izq;
    }

    /**
     * entrega el vertice tangente
     * @return - el vertice tangente
     */
    public vertice getTan()
    {
        return tan_izq;
    }

    /**
     * entrega el vertice tangente por la derecha
     * @return - el vertice tangente
     */
    public vertice getTan_der()
    {
        return tan_der;
    }

    /**
     * entrega la pendiente de la tangente por la izquierda
     * @return - un double con la pendiente
     */
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
public double getM_izq()
{
    return m_izq;
}

/**
 * entrega la pendiente de la tangente por la derecha
 * @return - un double con la pendiente
 */
public double getM_der()
{
    return m_der;
}

/**
 * entrega la pendiente de la tangente
 * @return - un double con la pendiente
 */
public double getM()
{
    return m_izq;
}

/**
 * Cambia el vertice del conjunto externo
 * @param e - elemento a cambiar
 */
public void setP(vertice e)
{
    p=new vertice();
    if(e==null) System.out.println("CTM E es nulo (Set_P)");
    p=e;
}

/**
 * Cambia el vertice tangente por la derecha
 * @param e - el elemento a cambiar
 */
public void setTan_izq(vertice e)
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
{
    //System.out.println("E : "+e.toString());
    tan_izq=new vertice();
    if(e==null) System.out.println("CTM E es nulo (Set_Izq)");
    tan_izq=e;
    //System.out.println("tan_izq :"+tan_izq.toString());
    if(tan_izq.getX()-p.getX()!=0)
        m_izq=m(p,tan_izq);
    else
        m_izqesInfinita=true;
}

/**
 * Cambia el vertice tangente por la derecha
 * @param e - el elemento a cambiar
 */
public void setTan_der(vertice e)
{
    tan_der=new vertice();
    tan_der=e;
    if(tan_der.getX()-p.getX()!=0)
        m_der=m(p,tan_der);
    else
        m_deresInfinita=true;
}

/**
 * cambiar la pendiente de la tangente izquierda
 * @param e - el elemento a cambiar
 */
public void setM_izq(double e)
{
    m_izq=e;
}

/**
 * cambia la pendiente de la tangente derecha
 * @param e - el elemento a cambiar
 */
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
public void setM_der(double e)
{
    m_der=e;
}

/**
 * calcula la pendiente dados dos vertices
 * @param p1 - primer vertice
 * @param p2 - segundo vertice
 * @return la pendiente de dos vertices
 */
public double m(vertice p1, vertice p2)
{
    double m = 0;
    if(p2.getX()-p1.getX()!=0)
        m=(double)( (p2.getY()-p1.getY())/(p2.getX()-p1.getX()) );
    else
        m=Double.POSITIVE_INFINITY;

    if(m == -0 || m== 0) m= 0;

    return m;
}
/**
 * consulta si los campos de los vertices tangentes est n ya ocupados
 * @return true si est  completo, false si NO est  completo
 */
public boolean estaCompleto()
{
    return (p!=null && tan_izq!=null && tan_der!=null);
}

/**
 * Consulta si la tangente por la izquierda es infinita
 * @return true es infinita, false no es infinita
 */
public boolean m_izqesInfinita()
{
    return m_izqesInfinita;
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
    }

    /**
     * Consulta si la tangente por la derecha es infinita
     * @return true es infinita , false no es infinita
     */
    public boolean m_deresInfinita()
    {
        return m_deresInfinita;
    }

    /**
     * Consulta si la tangente  es infinita
     * @return true es infinita , false no es infinita
     */
    public boolean esInfinita()
    {
        return m_izqesInfinita;
    }

    /**
     * entrega la informaci n de la tangente
     * @return un String de informaci n de la Tangente
     */
    public String toString()
    {
        String m;
        m=p.toString()+" "+tan_izq.toString();

        if(m_izqesInfinita())
            m=m+" infinita";
        else
            if(m_izq - ((int)m_izq)==0)
                m=m+" "+String.format("%.0f", m_izq);
            else
                m=m+" "+String.format("%.2f", m_izq);


        m=m+" "+tan_der.toString();
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        if(m_deresInfinita())
            m=m+" infinita";
        else
            if(m_der - ((int)m_der)==0)
                m=m+" "+String.format("%.0f", m_der);
            else
                m=m+" "+String.format("%.2f", m_der);

        return m;
    }

    /**
     * entrega la informaci n de la tangente
     * @return un String de informaci n de la Tangente
     */
    public String toString(String m)
    {

        m=p.toString()+" "+tan_izq.toString();

        if(m_izqesInfinita())
            m=m+" infinita";
        else
            if(m_izq - ((int)m_izq)==0)
                m=m+" "+String.format("%.0f", m_izq);
            else
                m=m+" "+String.format("%.2f", m_izq);



        return m;
    }


}
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Código D.7: regionConvexa.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package algoritmobr;

import java.util.ArrayList;

/**
 * @author Joel
 * @author Cristian
 */
public class regionConvexa {
    public ArrayList<vertice> region;
    public ArrayList<Integer> puntos;

    public regionConvexa(){
        this.region=new ArrayList<>();
        this.puntos=new ArrayList<>();
    }

    public void addRegion(ArrayList<vertice> B){
        this.region.addAll(B);
    }

    public void addPunto(int A){
        this.puntos.add(A);
    }

    public void reset(){
        this.puntos.clear();
        this.region.clear();
    }

    public boolean esVacia(){
        if(this.puntos.isEmpty())return true;
        else return false;
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
}

public void Particion(vertice desde, vertice hasta){
    int op = 100*Posicion(desde)+Posicion(hasta);
    //System.out.println("op = "+op);
    switch (op){
        case 101:
            //System.out.println("Adentro-Adentro\n");
            partirNoVacia(desde, hasta);
            partirNoVacia(hasta, desde);
            partirVacia(desde, hasta);
            partirVacia(hasta, desde);
            break;
        case 102:
            //System.out.println("Adentro-Borde\n");
            partirNoVacia(desde, hasta);
            partirNoVacia(hasta, desde);
            partirVacia(desde, hasta);
            break;
        case 103:
            //System.out.println("Adentro-Afuera\n");
            vertice A = new vertice(puntoFantasma(hasta, desde));
            partirNoVacia(desde, A);
            partirNoVacia(A, desde);
            partirVacia(desde, A);
            break;
        case 201:
            //System.out.println("Borde-Adentro\n");
            partirNoVacia(desde, hasta);
            partirNoVacia(hasta, desde);
            partirVacia(hasta, desde);
            break;
        case 202:
            //System.out.println("Borde-Borde\n");
            partirNoVacia(desde, hasta);
            partirNoVacia(hasta, desde);
            break;
        case 203:
            //System.out.println("Borde-Afuera\n");
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
                vertice B = new vertice(puntoFantasma(hasta, desde));
                if(B!=null){
                    partirNoVacia(desde, B);
                    partirNoVacia(B, desde);
                }
                break;
            case 301:
                //System.out.println("Afuera-Dentro\n");
                vertice C = new vertice(puntoFantasma(desde, hasta));
                partirNoVacia(C, hasta);
                partirNoVacia(hasta, C);
                partirVacia(hasta, C);
                break;
            case 302:
                //System.out.println("Afuera-Borde\n");
                vertice D = new vertice(puntoFantasma(desde, hasta));
                if(D!=null){
                    partirNoVacia(D, hasta);
                    partirNoVacia(hasta, D);
                }
                break;
            case 303://CASO ESPECIAL, REVISARLO!!!!!!
                //System.out.println("Afuera-Afuera\n");
                //System.out.println("Regi n = "+this.region.toString());
                //System.out.println("Segmento cortado = "+desde.toString↩
                    ()+hasta.toString());
                ArrayList<vertice> ghost = new ArrayList<>();
                ghost.addAll(puntoFantasma2(desde, hasta));
                if(ghost.size()==2){
                    partirNoVacia(ghost.get(0), ghost.get(1));
                    partirNoVacia(ghost.get(1), ghost.get(0));
                }
                break;
        }
        //System.out.println("Puntos azules en esta regi n:\n"+this.↩
            puntos.toString());
    }

    public int Posicion(vertice punto)
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
{
    int counter = 0;
    int i;
    int n=this.region.size();
    double xinters;
    vertice p1,p2;
    p1 = this.region.get(0);
    for (i=1;i<=n;i++) {
        p2 = this.region.get(i%n);
        //si est  en la horizontal
        if(p1.getY()==p2.getY()&&p1.getY()==punto.getY()&&punto.getX()↩
            >Math.min(p1.getX(), p2.getX())&&punto.getX()<Math.max(p1.↩
            getX(),p2.getX()))
             return 2;
        //if(punto.equals(p1)||punto.equals(p2)) return 2;
        if (punto.getY() > Math.min(p1.getY(),p2.getY())) {
            if (punto.getY() <= Math.max(p1.getY(),p2.getY())) {
                if (punto.getX() <= Math.max(p1.getX(),p2.getX())) {
                    if (p1.getY() != p2.getY()) {
                        xinters = (punto.getY()-p1.getY())*(p2.getX()-↩
                            p1.getX())/(p2.getY()-p1.getY())+p1.getX();
                        //si est  en el borde de cualquiera distinto ↩
                            al horizontal
                        if(punto.getX() == xinters)
                            return 2;
                        if (p1.getX() == p2.getX() || punto.getX() <= ↩
                            xinters)
                            counter++;
                    }
                }
            }
        }
        p1 = p2;
    }
    //si las intersecciones son pares, est  fuera; si es impar, est ↩
        dentro
    if (counter % 2 == 0)
        return 3;
    else
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
            return 1;
    }

    public double distPtoRecta(vertice s, vertice d, vertice h){
        double A = h.getY()-d.getY();
        double B = -1*(h.getX()-d.getX());
        double C = (-1*d.getX()*A)-(d.getY()*B);
        double dPR = Math.abs((A*s.getX())+(B*s.getY())+C)/(Math.sqrt(Math↩
            .pow(A, 2)+Math.pow(B, 2)));
        return dPR;
    }

    public ArrayList<vertice> puntoFantasma2(vertice a, vertice b){
        ArrayList<vertice> s = new ArrayList<>();
        vertice s1 = new vertice(a);
        vertice s2 = new vertice(b);
        //s.addAll(segmentoCortado(s1, s2));
        ArrayList<vertice> ghost = new ArrayList<>();
        /*ghost.add(new vertice(inter(s.get(0), s.get(1), s1, s2)));
        ghost.add(new vertice(inter(s.get(2), s.get(3), s1, s2)));*/
        for(int i=0; i<this.region.size(); i++){
            vertice X;
            if(i==this.region.size()-1){
                X = new vertice(inter(s1, s2, this.region.get(i), this.↩
                    region.get(0)));
            }
            else{
                X = new vertice(inter(s1, s2, this.region.get(i), this.↩
                    region.get(i+1)));
            }
                if(X!=null)
                    if(entrePtos(X, s1, s2))
                        ghost.add(X);

        }
        return ghost;
    }

    public vertice puntoFantasma(vertice a, vertice b){
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        vertice s1 = new vertice(a);
        vertice s2 = new vertice(b);
        //ArrayList<vertice> s = new ArrayList<>();
        /*s.addAll(segmentoCortado(s1, s2));
        //System.out.println("seg cortados = "+s.toString()+"\n");
        int x, y;
        if(((estaIzquierda(s.get(0), s.get(1), s1)&&!estaIzquierda(s.get↩
            (0), s.get(1), s2))||(estaIzquierda(s.get(0), s.get(1), s2)&&!↩
            estaIzquierda(s.get(0), s.get(1), s1)))&&!enLinea(s.get(0), s.↩
            get(1), s1)&&!enLinea(s.get(0), s.get(1), s2)){
            x=0; y=1;
        }
        else{
            x=2; y=3;
        }*/
        //System.out.println("segmento cortado = ["+s.get(x).toString()+",↩
            "+s.get(y).toString()+"]\n");
        /*ghost.add(new vertice(inter(s.get(0), s.get(1), s1, s2)));
        ghost.add(new vertice(inter(s.get(2), s.get(3), s1, s2)));*/
        vertice ghost;
        for(int i=0; i<this.region.size(); i++){
            if(i==this.region.size()-1){
                ghost = new vertice(inter(s1, s2, this.region.get(i), this↩
                    .region.get(0)));
            }
            else{
                ghost = new vertice(inter(s1, s2, this.region.get(i), this↩
                    .region.get(i+1)));
            }
                if(ghost!=null)
                    if(entrePtos(ghost, s1, s2))
                        return ghost;

        }
        return null;
    }

    public boolean entrePtos(vertice v, vertice s1, vertice s2){
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        if((v.getX()>=s1.getX()&&v.getX()<=s2.getX()&&v.getY()<=s1.getY()↩
            &&v.getY()>=s2.getY())||(v.getX()>=s2.getX()&&v.getX()<=s1.getX↩
            ()&&v.getY()<=s2.getY()&&v.getY()>=s1.getY()))
            return true;
        else return false;
    }

    public vertice inter(vertice a, vertice b, vertice c, vertice d){
        double A = b.getY()-a.getY();
        double B = -1*(b.getX()-a.getX());
        double C = (a.getX()*A)+(a.getY()*B);

        double D = d.getY()-c.getY();
        double E = -1*(d.getX()-c.getX());
        double F = (c.getX()*D)+(c.getY()*E);

        if(((A*E)-(B*D))!=0){
        double x = ((C*E)-(B*F))/((A*E)-(B*D));
        double y = ((A*F)-(C*D))/((A*E)-(B*D));

        return new vertice(x, y);
        }
        else return null;
    }

    public ArrayList<vertice> segmentoCortado(vertice s1, vertice s2){
        ArrayList<vertice> v = new ArrayList<>();
        ArrayList<vertice> segmento = new ArrayList<>();
        if((estaIzquierda(this.region.get(0), s1, s2)&&!estaIzquierda(this↩
            .region.get(this.region.size()-1),s1, s2))||(estaIzquierda(this↩
            .region.get(this.region.size()-1),s1, s2)&&!estaIzquierda(this.↩
            region.get(0),s1, s2))||(entrePtos(this.region.get(0),s1,s2)&&↩
            entrePtos(this.region.get(this.region.size()-1),s1,s2))){
            v.add(new vertice(this.region.get(0).getX(), this.region.get↩
                (0).getY()));
            v.add(new vertice(this.region.get(this.region.size()-1).getX()↩
                , this.region.get(this.region.size()-1).getY()));
        }
        for(int i=0; i<this.region.size()-1; i++){
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
            if((estaIzquierda(this.region.get(i),s1, s2)&&!estaIzquierda(↩
                this.region.get(i+1),s1, s2))||(estaIzquierda(this.region.↩
                get(i+1),s1, s2)&&!estaIzquierda(this.region.get(i),s1, s2)↩
                )||(entrePtos(this.region.get(i),s1,s2)&&entrePtos(this.↩
                region.get(i+1),s1,s2))){
                 v.add(new vertice(this.region.get(i).getX(), this.region.↩
                     get(i).getY()));
                 v.add(new vertice(this.region.get(i+1).getX(), this.region↩
                     .get(i+1).getY()));
            }
        }
        for(int i=0; i<v.size(); i++){
            segmento.add(v.get(i));
        }
        return segmento;
    }

    public void partirVacia(vertice a, vertice b){
        ArrayList<vertice> part = new ArrayList<>();
        ArrayList<vertice> v = new ArrayList<>();
        vertice s1 = new vertice(a.getX(), a.getY());
        vertice s2 = new vertice(b.getX(), b.getY());
        v.addAll(segmentoCortado(s1, s2));
        //if(distPtoRecta(s1, v.get(0), v.get(1))<distPtoRecta(s1, v.get↩
            (2), v.get(3))){
        //if(estaIzquierda(v.get(0), v.get(1), a) && !estaIzquierda(b, v.↩
            get(2), a) /*&& !enLinea(b, v.get(2), a)*/){
        if((estaIzquierda(v.get(0), s1, s2)&&estaDerecha(v.get(1), s1, s2)↩
             || estaIzquierda(v.get(1), s1, s2)&&estaDerecha(v.get(0), s1, ↩
            s2))&&distPtoRecta(s1, v.get(0), v.get(1))<distPtoRecta(s2, v.↩
            get(0), v.get(1))){
            part.add(v.get(0));
            part.add(v.get(1));
        }
        else{
            part.add(v.get(2));
            part.add(v.get(3));
        }
        if(!enLinea(s1, s2, part.get(0))&&!enLinea(s1, s2, part.get(1))){
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
            part.add(s1);
            mergesort(part, 0, part.size(), PuntoMenorOrdenada(part));
            AlgoritmoBR.particiones.add(new regionConvexa());
            AlgoritmoBR.particiones.get(AlgoritmoBR.particiones.size()-1).↩
                addRegion(part);
        }
        part.clear();
    }

    public void partirNoVacia(vertice a, vertice b){
        boolean addS1=true;
        boolean addS2=true;
        ArrayList<vertice> part = new ArrayList<>();
        vertice s1 = new vertice(a.getX(), a.getY());
        vertice s2 = new vertice(b.getX(), b.getY());
        part.addAll(this.region);
        for(int i=0; i<part.size(); i++){
            if(estaIzquierda(s1, s2, part.get(i))){
                part.remove(i);
                i--;
            }
            else
            {
                if(enLinea(s1, s2, part.get(i))){
                    if(distPtoPto(s1, part.get(i))<distPtoPto(s2, part.get↩
                        (i)))
                         addS1=false;
                    else addS2=false;
                }
            }
        }
        if(addS1)part.add(s1);
        if(addS2)part.add(s2);
        mergesort(part, 0, part.size(), PuntoMenorOrdenada(part));
        AlgoritmoBR.particiones.add(new regionConvexa());
        AlgoritmoBR.particiones.get(AlgoritmoBR.particiones.size()-1).↩
            addRegion(part);

        for(int i=0; i<this.puntos.size(); i++){
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
            if(AlgoritmoBR.particiones.get(AlgoritmoBR.particiones.size()↩
                −1).Posicion(AlgoritmoBR.VGi.get(this.puntos.get(i)))!=3){
                AlgoritmoBR.particiones.get(AlgoritmoBR.particiones.size()↩
                    −1).addPunto(this.puntos.get(i));
                //this.puntos.remove(i);
            }
        }
        part.clear();
    }

    public double distPtoPto(vertice A, vertice B){
        return Math.sqrt(Math.pow((B.getX()−A.getX()), 2)+Math.pow((B.getY↩
            ()−A.getY()), 2));
    }

    public void mergesort(ArrayList<vertice> data, int first, int n, ↩
        vertice m)
   {
      int n1; // Size of the first half of the array
      int n2; // Size of the second half of the array

      if (n > 1)
      {
         // Compute sizes of the two halves
         n1 = n / 2;
         n2 = n − n1;

         mergesort(data, first, n1, m);       // Sort data[first] through ↩
             data[first+n1−1]
         mergesort(data, first + n1, n2, m); // Sort data[first+n1] to the↩
              end

         // Merge the two sorted halves.
         merge(data, first, n1, n2, m);
      }
   }

   private void merge(ArrayList<vertice> data, int first, int n1, int n2, ↩
       vertice m)
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
{
   ArrayList<vertice> temp = new ArrayList(); // Allocate the temporary↩
       array
   int copied  = 0; // Number of elements copied from data to temp
   int copied1 = 0; // Number copied from the first half of data
   int copied2 = 0; // Number copied from the second half of data
   int i;           // Array index to copy from temp back into data

   // Merge elements, copying from two halves of data to the temporary ↩
      array.
   while ((copied1 < n1) && (copied2 < n2))
   {
      if (Angulo(m,data.get(first + copied1)) < Angulo(m, data.get(↩
         first + n1 + copied2)))
         temp.add(copied++,data.get(first + (copied1++)));
      else
         temp.add(copied++,data.get(first + n1 + (copied2++)));
   }
   // Copy any remaining entries in the left and right subarrays.
   while (copied1 < n1)
      temp.add(copied++,data.get(first + (copied1++)));
   while (copied2 < n2)
      temp.add(copied++,data.get(first + n1 + (copied2++)));

   // Copy from temp back to the data array.
   for (i = 0; i < n1+n2; i++)
      data.set(first + i, temp.get(i));
}

 public double Angulo(vertice A, vertice B){
     double my, mx, mb, angulo;
     my = B.getY()-A.getY();
     mx = B.getX()-A.getX();
     if(mx==0){
         if(my>0) return 90;
         else if(my<0) return 270;
         else {
             //El punto A y C son los mismos
             return 0;
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
            }
        }
        else if(my==0){
            if(mx>0) return 0;
            else return 180;
        }
        else{
            mb=my/mx;
            angulo=Math.atan(mb)*(180/Math.PI);
            if(mx>0){
                if(my<0) return angulo + 360;
                else return angulo;
            }
            else return angulo + 180;
        }
    }

    public boolean estaIzquierda(vertice A, vertice B, vertice C){
        if((((B.getX()-A.getX())*(C.getY()-A.getY()))-((B.getY()-A.getY())↩
           *(C.getX()-A.getX())))>0)
            return true;
        else return false;
    }
    public boolean estaDerecha(vertice A, vertice B, vertice C){
        if((((B.getX()-A.getX())*(C.getY()-A.getY()))-((B.getY()-A.getY())↩
           *(C.getX()-A.getX())))<0)
            return true;
        else return false;
    }

    public boolean enLinea(vertice A, vertice B, vertice C){
        if((((B.getX()-A.getX())*(C.getY()-A.getY()))-((B.getY()-A.getY())↩
           *(C.getX()-A.getX())))==0)
            return true;
        else return false;
    }

    public vertice PuntoMenorOrdenada(ArrayList <vertice> puntos){
        vertice menor;
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        menor=puntos.get(0);
        for(int i=1; i<puntos.size(); i++){
            if(puntos.get(i).getY()<menor.getY()||((puntos.get(i).getY()==↩
               menor.getY())&&(puntos.get(i).getX()>menor.getX())))
                menor=puntos.get(i);
        }
        return menor;
    }
    public String toString(){
        String m="";
        m=m+this.region.toString()+"\nContenido = ";
        for(int i=0; i<this.puntos.size(); i++)
            m=m+AlgoritmoBR.VGi.get(this.puntos.get(i));
        return m;
    }

}
```

Código D.8: rectaDual.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package algoritmobr;

import java.util.ArrayList;

/**
 * @author Joel
 * @author Cristian
 */
public class rectaDual {
    private vertice ptoAzul;
    private ArrayList<dualRojo> ptosRojos = new ArrayList<> ();

    public rectaDual(vertice B, ArrayList<vertice> R){
        this.ptoAzul=B;
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        for(int i=0; i<R.size(); i++){
            dualRojo aux = new dualRojo(R.indexOf(R.get(i)), R.get(i), ↩
                inter(ptoAzul, R.get(i)), ptoAzul);
            ptosRojos.add(aux);
        }
        //mergesort(ptosRojos, 0, ptosRojos.size());
    }
    public vertice inter(vertice P, vertice U){
        vertice solucion=new vertice();
        double y, x;
        double a = P.getX();
        double b = -1;
        double c = -1*P.getY();
        double d = U.getX();
        double e = -1;
        double f = -1*U.getY();
        y = ((f * a) - (d * c)) / ((e * a) - (d * b));
        x = (c - (b * y)) / a;
        solucion.setVertice(x, y);
        return solucion;
    }

   /*
    public void mergesort(ArrayList<dualRojo> data, int first, int n)
   {
      int n1; // Size of the first half of the array
      int n2; // Size of the second half of the array

      if (n > 1)
      {
         // Compute sizes of the two halves
         n1 = n / 2;
         n2 = n - n1;

         mergesort(data, first, n1);      // Sort data[first] through data↩
            [first+n1-1]
         mergesort(data, first + n1, n2); // Sort data[first+n1] to the ↩
            end
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
        // Merge the two sorted halves.
        merge(data, first, n1, n2);
    }
}

private void merge(ArrayList<dualRojo> data, int first, int n1, int n2)
{
    ArrayList<dualRojo> temp = new ArrayList(); // Allocate the ↩
        temporary array
    int copied  = 0; // Number of elements copied from data to temp
    int copied1 = 0; // Number copied from the first half of data
    int copied2 = 0; // Number copied from the second half of data
    int i;           // Array index to copy from temp back into data

    // Merge elements, copying from two halves of data to the temporary ↩
        array.
    while ((copied1 < n1) && (copied2 < n2))
    {
        if ((data.get(first + copied1).getInter().getX() < data.get(↩
            first + n1 + copied2).getInter().getX())||((data.get(first + ↩
            copied1).getInter().getX() == data.get(first + n1 + copied2).↩
            getInter().getX())&&(data.get(first + copied1).getInter().↩
            getY() < data.get(first + n1 + copied2).getInter().getY())))
          temp.add(copied++,data.get(first + (copied1++)));
        else
          temp.add(copied++,data.get(first + n1 + (copied2++)));
    }
    // Copy any remaining entries in the left and right subarrays.
    while (copied1 < n1)
        temp.add(copied++,data.get(first + (copied1++)));
    while (copied2 < n2)
        temp.add(copied++,data.get(first + n1 + (copied2++)));

    // Copy from temp back to the data array.
    for (i = 0; i < n1+n2; i++)
        data.set(first + i, temp.get(i));
}*/
 public vertice getAzul(){
     return ptoAzul;
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile


```
    }
    public ArrayList<dualRojo> getRojo(){
        return ptosRojos;
    }

    public String toString(){
        String mensaje="El punto "+ptoAzul.toString()+" se intersecta con↩
            :\n";
        for(int i=0; i<ptosRojos.size(); i++)
            mensaje=mensaje+ptosRojos.get(i).toString();
        return mensaje;
    }
}
```

Universidad del Bío-Bío. Sistema de Bibliotecas - Chile

Apéndice E

MANUAL DE USUARIO DEL BR-SOFTWARE