UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y SISTEMAS

DETECCIÓN DE CUADRILÁTEROS CONVEXOS HETEROCROMÁTICOS EN CONJUNTOS DE PUNTOS 4-COLOREADOS

T E S I S

QUE PARA OPTAR POR EL GRADO DE:
MAESTRA EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A:

ALMA ROSARIO ARÉVALO LOYOLA

DIRECTOR DE TESIS:
DR. JORGE URRUTIA GALICIA
Posgrado en Ciencia e Ingeniería de la Computación

CIUDAD UNIVERSITARIA, CDMX
ENERO 2021

Índice general

Introducción

La geometría es una de las disciplinas más antiguas en la historia de las ciencias. Solemos entenderla como el estudio de las propiedades de formas y medidas en un espacio. Con la axiomatización de Euclides se abstrajeron conceptos como *punto*, *recta*, *distancia*, etcétera, que prevalecen hoy en día como parte importante de la matemática clásica. Con la llegada de las computadoras y con el hecho de que ha sido natural su utilización como herramienta para resolver algunos problemas matemáticos, surge en el siglo pasado la *geometría computacional*, disciplina de estudio que podríamos describir como la encargada de transformar problemas clásicos de geometría en problemas computacionales, o viceversa.

La geometría computacional tiene dos objetivos principales:

- Usar la computación como herramienta para resolver problemas de geometría clásica.
- Usar la geometría clásica adaptada como herramienta para avances tecnológicos.

En relación a estas dos finalidades, podríamos dividir el estudio de la geometía computacional en dos enfoques:

- Al primer enfoque lo llamaremos *algorítmico* y lo podemos pensar como la manera en que logramos que una computadora resuelva algunos problemas geométricos, para lo cual deben ser planteados de manera abstracta y, en específico, de manera computable. Más adelante veremos qué significa que un problema sea computable y qué se requiere para resolverlo.
- El segundo enfoque sería el de aplicar la geometría en problemas que surgen en el desarrollo de la computación y de la tecnología. Con el potencial actual de las computadoras, cada vez es más común utilizar elementos gráficos, tanto en dos dimensiones como simulaciones de un espacio tridimensional, por ejemplo lo que se conoce como realidad virtual. En este contexto, es necesario computar el manejo de esos objetos gráficos y es importante que esto se haga de manera correcta y eficiente.

En realidad ambos enfoques están muy relacionados, pues en esta última finalidad mencionamos que es importante computar de manera eficiente y para garantizar esto, es necesario el estudio formal de los problemas.

El desarrollo de este trabajo de tesis está basado fuertemente en el enfoque algorítmico, pues nos centraremos en abordar, entender y resolver problemas geométricos abstractos, sin ahondar en alguna aplicación directa de ellos.

Para entender por completo el planteamiento de dichos problemas y la manera de resolverlos, es necesario contar con nociones previas sobre computabilidad, complejidad computacional, diseño y análisis de algoritmos, y estructuras de datos.

En el primer capítulo desarrollaremos de manera muy general estos y otros conceptos relacionados, los cuales serán utilizados más adelante para exponer los resultados de esta tesis.

En el segundo capítulo veremos particularmente las definiciones y herramientas de geometría computacional que son útiles en el tipo de problemas estudiados y planteados.

En el tercer capítulo se abordan planteamientos y resultados de problemas existentes, necesarios para entender el problema de nuestro interés, así como para poner en contexto su relevancia en el área.

En el cuarto capítulo se plantea el problema principal de este trabajo de tesis, se exponen sus resultados de manera teórica y, por último, se detallan los algoritmos para su resolución. En el quinto y último capítulo, cerramos con las conclusiones de dicho problema y planteamos algunos problemas abiertos que pueden ser interesantes para abordar en el futuro.

1 Preliminares sobre Teoría de la Computación

En este capítulo presentaremos algunas nociones necesarias para entender el resto de la tesis.

1.1. Resolución de problemas y computabilidad

Tanto en la vida cotidiana como en la labor matemática es común hablar de *problemas*, intuitivamente entendemos qué son y sabemos que el interés es resolverlos. Matemáticamente, es posible determinar propiedades de los problemas y de sus soluciones. Por ejemplo, dada una solución, es posible y útil demostrar que es correcta o demostrar que es la mejor respecto a cierto parámetro. También es útil, por otro lado, demostrar que algún problema no tiene solución. A lo largo de los años se han desarrollado diversas herramientas para resolver toda clase de problemas, de la que nos interesa hablar es de la computadora. Las primeras computadoras eran llamadas simplemente calculadoras y en ocasiones, de hecho, se llegan a utilizar como sinónimos los verbos *calcular* y *computar*. Es claro que un problema de tipo *calcular* 26735^2, una computadora lo resuelve fácil y rápidamente, en cambio hay otros en los que no nos será "tan útil", por ejemplo: *¿cuál es el mejor lugar para ir de vacaciones?* o *¿debo invertir en la bolsa de valores?*, comenzando con que estos dos últimos ni siquiera tienen una solución determinada. Sin embargo, existen problemas cuya solución está bien definida y aun así una computadora no la puede encontrar, o no en un tiempo razonable.

En *teoría de la computabilidad* se estudia qué tipo de problemas puede resolver una computadora, esto se hace definiendo formalmente los modelos de cómputo, de manera que se puede demostrar cuándo un problema es resoluble por determinado modelo. Por no ser tema relevante para los fines de esta tesis, no ahondaremos demasiado en esto. Lo que nos interesa mencionar, en resumen, es que está demostrado que los diversos modelos de cómputo son equivalentes, es decir, que cada uno puede simular a los otros existentes; y es por eso que está bien definido que un problema sea *computable*, incluso sin aclarar cuál es el modelo que se está considerando. El tema se aborda ampliamente en [14].

1.2. Algoritmos

El concepto clave para especificar un proceso computable es el de *algoritmo*. Un algoritmo se suele definir informalmente como una secuencia finita de pasos bien definidos, cada uno de ellos realizable en tiempo finito. La finalidad de un algoritmo es "saber resolver" un problema.

Un *problema* se especifica formalmente como una pregunta que depende de ciertos parámetros que se conocen como datos de *entrada* y que, para valores fijos de estos, existe una solución bien definida. A los datos de la solución, los que serán dados por el algoritmo una vez que resuelve el problema, se les conoce como *salida*.

Se le llama *instancia* de un problema a un conjunto fijo de datos de entrada.

Un algoritmo *correcto* para un problema, es aquel que, para cualquier entrada, al ejecutarse (es decir, seguir los pasos descritos) produce la salida correcta.

Veamos un ejemplo:

Ejemplo 1.1. *Consideremos el problema de determinar si un número natural dado es primo.*

El parámetro de entrada es el número n del que se quiere saber si es o no primo, y la respuesta debe ser *sí* o *no*. Una posible instancia del problema es la pregunta *¿es el 5 un número primo?*, para la cual la respuesta correcta es *sí*. Otra posible pregunta es *¿es el 256 un número primo?*, para la cual la respuesta correcta es *no*.

Por definición, un número es primo si tiene exactamente dos divisores: él mismo y la unidad.

Vamos a describir un algoritmo que resuelve correctamente este problema de decidir si un número entero es o no primo. La secuencia de pasos inicia dividiendo a n por cada entero mayor que 1 y menor que n, en ese orden. Si alguna de esas divisiones resulta en un número entero, entonces n es un número compuesto y el algoritmo responde *no*. Si, por el contrario, se ejecutan las $n - 2$ divisiones y en ninguna se obtiene un resultado entero, entonces el algoritmo responde *sí*.

Es claro que el algoritmo termina en tiempo finito, ya que su entrada corresponde a un número natural n y, para calcular la salida, se hicieron a lo más $n - 2$ operaciones. Por ahora, es importante el hecho de que sea finito, puesto que es parte de la definición de algoritmo. Sin embargo, más adelante veremos la importancia de la cantidad de pasos que ejecuta un algoritmo respecto al tamaño de su entrada.

Es fácil convencerse de que el algoritmo descrito anteriormente es correcto, ya que se busca entre todos los posibles candidatos a ser divisores del número y se comprueba si lo son, haciendo la división. La ejecución termina antes de comprobar todos los candidatos, solamente si alguno resulta sí ser divisor, en cuyo caso se tiene la solución al problema.

En general, al expresar o describir un algoritmo, intentaremos que sea de la forma más transparente posible, de manera que sea fácil convencerse de que es correcto, o bien, demostraremos formalmente que lo es.

1.3. Complejidad

Vimos que la teoría de la computabilidad se encarga de determinar cuáles problemas son los que se pueden resolver. Por otro lado, la *teoría de la complejidad* se encarga de considerar la cantidad de recursos necesarios para resolver un problema computable.

Es importante destacar que para resolver un mismo problema pueden existir varios algoritmos distintos. Una vez asegurando que son correctos, lo siguiente que resulta relevante cuestionarnos es cuál de ellos es más eficiente. Para pensar en eficiencia, vamos a destacar dos aspectos medibles en un algoritmo, estos son el *tiempo* y el *espacio* requeridos para su ejecución.

La medición de estos parámetros de la eficiencia de un algoritmo nos lleva al concepto de *complejidad*.

Para un estudio más extenso sobre temas de *complejidad computacional* se recomienda consultar [8].

Para el propósito de esta tesis nos interesa únicamente destacar la noción de complejidad de un algoritmo y saber cómo expresarla. La complejidad en tiempo es, a grandes rasgos, la cantidad de pasos ejecutados durante el algoritmo, y la complejidad en espacio es la cantidad de datos que se requiere almacenar durante dicha ejecución. La medición específica de estos parámetros depende de algunos aspectos, como del modelo de computación que se esté considerando, así como del hardware en el que sea implementado, sin embargo, para fines de este trabajo, no se ahondará en estos aspectos.

La complejidad de un algoritmo se suele expresar como una función que depende del tamaño de la entrada. Para clasificar a las funciones de complejidad se utiliza la notación de cotas asintóticas. Para mayor referencia, consultar [15]. En este trabajo la que más utilizaremos es la cota superior, que se denota con la letra O.

Definición 1.1. *Dadas dos funciones f y g, decimos que f está en $O(g)$ si existen constantes positivas c y n_0 tales que, siempre que $n \geq n_0$, se cumple*

$$f(n) \leq c\, g(n).$$

La definición anterior nos dice que la función f está acotada superiormente por la función g para valores suficientemente grandes de n. En el caso de la complejidad en tiempo, la función f representa el número de pasos que toma la ejecución de un algoritmo que resuelva el problema para una entrada de tamaño n, en el peor caso. La función g suele ser la más sencilla que acota por arriba, salvo tal vez una constante. Por ejemplo, si decimos que el algoritmo A tiene complejidad $O(n^3)$ significa que alguna función g de la forma cn^3, con

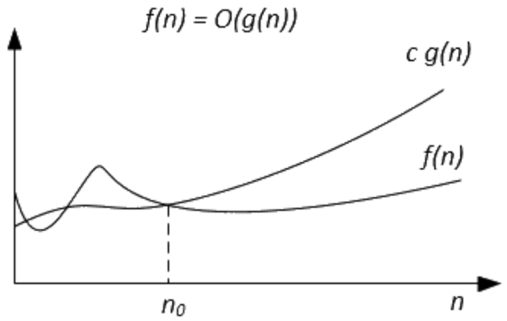

Figura 1.1: Cota superior asintótica.

c constante, acota por arriba (para valores suficientementes grandes de n) a la función f, la cual expresa el número de pasos que ejecuta el algoritmo para resolver a la peor instancia posible (la que toma más pasos) con tamaño de entrada n. La cota superior es la que más utilizaremos al analizar un algoritmo, ya que, como en general se busca reducir el tiempo de ejecución, lo primero que nos interesa garantizar es que ese tiempo esté acotado por arriba. Sin embargo, notemos que una cota superior podría dar información no muy precisa, pues por ejemplo, la función $f(n) = 2n$, es cierto que está en $O(n^6)$, pero claramente su crecimiento es mucho más lento que el de la función n^6. Por esta razón, es importante en ocasiones utilizar también cotas inferiores o cotas justas.

Veamos la definición de la cota inferior que se denota como Ω.

Definición 1.2. *Dadas dos funciones f y g decimos que f está en $\Omega(g)$, si existen constantes positivas c y n_0 tales que, siempre que $n \geq n_0$ se cumple*

$$f(n) \geq c\, g(n).$$

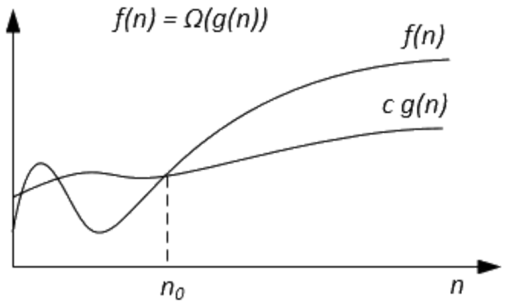

Figura 1.2: Cota inferior asintótica.

La cota inferior se usa sobre todo para mostrar que un algoritmo es óptimo, o para hablar del mejor de los casos de ejecución del mismo. Al igual que la cota superior, hay ocasiones en que podría dar información poco precisa, ya que por ejemplo la función $f(n) = 3n^2$ está en $\Omega(\log n)$, sin embargo f crece

mucho más rápido. Para expresar de manera más precisa el crecimiento de una función para valores grandes de n, es mucho más útil (pero también en muchas ocasiones más dificil de encontrar) una cota justa, que es aquella que cumple con ser cota inferior y cota superior a la vez. Pero, ¿cómo es posible que una función acote tanto por arriba como por abajo a otra?, esto es gracias a las constantes, la función g multiplicada por la constante c_1 es la que acota por abajo, y la función g multiplicada por la constante c_2 es la que acota por arriba a la función f. Técnicamente son distintas funciones las que acotan, pero estas tienen el mismo ritmo de crecimiento, asintónticamente hablando. Entonces, decir que f está en $\Theta(g)$ es afirmar que f crece de manera muy similar a la función g. Veamos la definición formal.

Definición 1.3. *Dadas dos funciones f y g decimos que f está en $\Theta(g)$, si existen constantes positivas c_1, c_2 y n_0 tales que, siempre que $n \geq n_0$ se cumple*

$$c_1\, g(n) \leq f(n) \leq c_2\, g(n).$$

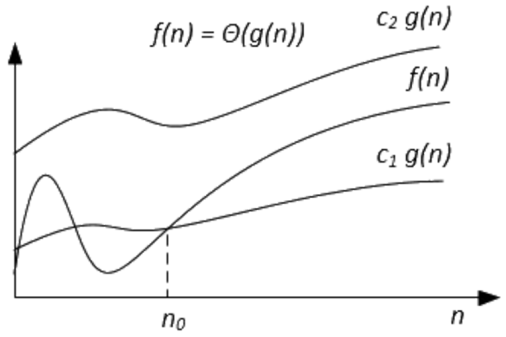

Figura 1.3: Cota justa asintótica.

1.4. Algoritmos de ordenamiento

Para asentar las definiciones anteriores, veamos como ejemplo el problema de ordenamiento. Este toma como entrada una lista de números enteros (o cualquier tipo de números que sean comparables y ordenables), y como salida se desea obtener la lista con los números ordenados del menor al mayor.

Hay varios algoritmos conocidos que resuelven este problema, vamos a analizar *Bubblesort* y *Mergesort*. Usamos los nombres en inglés ya que es como son mayormente reconocidos en la literatura.

1.4.1. Bubblesort u ordenamiento de burbuja

Algorithm 1 Bubblesort

```
1:  function BUBBLESORT(A)          ▷ A es un arreglo con n elementos por ordenar
2:      int i, j, k
3:      n = length(A)
4:      for i = 0 to n − 1 do
5:          for j = i + 1 to n − 1 do
6:              if A[i] > A[j] then
7:                  swap(A[i], A[j])
8:              end if
9:          end for
10:     end for
11:     return A
12: end function
```

Vamos a analizar los pasos de este algoritmo y su complejidad en tiempo. En las líneas 1 y 2 únicamente se declaran variables, lo cual gasta tiempo y espacio constante. En la línea 3 inicia un ciclo **for**, esto nos indica un recorrido de los elementos del arreglo, por medio de una variable auxiliar i a la que se le va asignado el valor de cada índice. Para cada uno de esos valores se ejecuta por completo lo que esté contenido dentro del **for**. Como hay otro **for** anidado, esto significa que para cada valor fijo de i, la variable j tomará los valores a partir de $i + 1$ y hasta $n - 1$ y, por último, para cada valor de j, se hará la comparación del elemento i-ésimo con el elemento j-ésimo, y si están en orden decreciente se intercambian. En resumidas cuentas, en un recorrido lineal del arreglo, cada elemento se compara con todos los que aparecen después de él y se intercambian si es necesario. Notemos que el primer elemento será comparado con $n - 1$ elementos; el segundo, con $n - 2$; y en general el i-ésimo se compara con otros $n - i$. Por lo que en total hacemos $\binom{n}{2}$ comparaciones y, por cada comparación, a lo más un intercambio. Esto significa que el algoritmo se ejecuta en tiempo $O(n^2)$ incluso en el peor de los casos.

1.4.2. Mergesort u ordenamiento por mezcla

Algorithm 2 Mergesort

```
1: function MERGESORT(A) ▷ A es un arreglo con n elementos por ordenar
2:     n = lenght(A)
3:     if n ≤ 1 then
4:         sortedlist = A
5:     else
6:         left = A[0 . . . ⌊(n−1)/2⌋]
7:         right = A[⌊(n−1)/2⌋ + 1 . . . n − 1]
8:         left = MERGESORT(left)
9:         right = MERGESORT(right)
10:        sortedlist = MERGE(left, right)
11:    end if
12:    return sortedlist
13: end function
```

El algoritmo Mergesort utiliza una técnica conocida como *divide y vencerás*, que consiste en resolver subinstancias del problema para después, a partir de esas subsoluciones, obtener la solución final. Esto se plantea como un algoritmo recursivo, donde se parte del hecho de que ordenar una lista vacía o de un elemento es trivial. Ahora, al ejecutar el algoritmo, siendo n el tamaño de la entrada, si $n = 0$ o $n = 1$ se cumple este caso base y se regresa la lista intacta, pues ya está ordenada, si $n \geq 2$ entonces se separa el arreglo en dos partes de tamaño aproximadamente la mitad y se hace la llamada recursiva para ejecutar el algoritmo sobre cada una de estas dos partes, es decir, se seguirá dividiendo en mitades hasta llegar a alguno de los casos base. Considerando que cada una de estas ejecuciones del algoritmo regresa una sublista ya ordenada, nos faltaría, a partir de estas dos sublistas, obtener el orden final de la lista completa. Esto último se logra con un recorrido simultáneo en ambos arreglos, comparando cada vez el elemento actual de un arreglo con el del otro y tomando cada vez al menor de ellos para agregarlo a la lista que será el resultado, seguido de aumentar el índice en ese arreglo.

La complejidad de un algoritmo recursivo se puede obtener resolviendo una recurrencia, sobre esto se puede ahondar en [15]. En este caso la complejidad de *Mergesort* es $O(n \log n)$. Notemos que la función auxiliar *Merge* toma tiempo lineal, ya que aunque usa un ciclo **for** para cada arreglo, estos no están anidados, es decir, cada uno se recorre solo una vez. Por lo que esta función no afecta a la complejidad del algoritmo *Mergesort*.

Está demostrado que la cota inferior para la complejidad en tiempo de los algoritmos de ordenamiento basados en comparaciones y sin alguna información adicional sobre los datos de entrada (solo sabiendo que son n elementos comparables entre sí) es $\Omega(n \log n)$ [15].

Algorithm 3 Función auxiliar Merge

```
1:  function MERGE(L, R)                    ▷ L y R son dos arreglos ordenados
2:      n_1 = lenght(L)
3:      n_2 = lenght(R)
4:      i = 0
5:      j = 0
6:      while i < n_1 and j < n_2 do
7:          if L[i] ≤ R[j] then
8:              add.result(L[i])
9:              i++
10:         else
11:             add.result(R[j])
12:             j++
13:         end if
14:     end while
15:     while i < n_1 do
16:         add.result(L[i])
17:     end while
18:     while j < n_2 do
19:         add.result(R[j])
20:     end while
21:     return result
22: end function
```

1.5. Estructuras de datos

Las estructuras de datos son una herramienta para almacenar y organizar la información, de manera que sea fácilmente accesible y modificable. Hay gran cantidad y variedad de estructuras de datos que se adecúan a determinados requerimientos. Cada algoritmo puede hacer uso de distintas de ellas con el fin de optimizar recursos de espacio y tiempo. Veamos algunos ejemplos:

- Arreglo:

  Un arreglo es una estructura que sirve para almacenar datos, generalmente todos del mismo tipo, en secuencia, es decir, en un orden específico dado por sus índices. Si el arreglo es de tamaño n, se utiliza el conjunto de índices $\{0, 1, \ldots, n-1\}$ y se puede acceder a cada elemento por medio del índice correspondiente. En principio, un arreglo tiene tamaño fijo, así que si se eliminara un elemento de él, los demás conservarán su índice original, y simplemente esa entrada, correspondiente al índice del elemento borrado, quedará vacía. De igual manera, para agregar un elemento nuevo al arreglo se debe establecer en qué índice y se reemplazará al elemento que esté almacenado ahí, si es que hay alguno.

- Lista:

  Una lista es una estructura similar al arreglo, ya que los elementos también están en secuencia. Sin embargo, ésta puede no tener un tamaño fijo, sino que su tamaño va cambiando según se vayan insertando o eliminando elementos. Por esta razón podría considerarse que, a diferencia de un arreglo, una lista nunca tiene entradas vacías.

- Lista ligada:

  La característica principal de una lista ligada es que cada elemento está *enlazado* al elemento siguiente. Esto, en términos de almacenamiento, significa que hay manera de acceder a la dirección de memoria del siguiente elemento.

- Lista circular:

  Cumple la misma característica que la lista ligada, de que a partir de cada elemento se puede acceder al siguiente, con una característica extra: que el último elemento también está enlazado con el primer elemento, por lo que se podría *recorrer* la lista completa y de ahí continuar al inicio.

- Lista doblemente ligada:

  Tiene las mismas características que una lista ligada, pero, en lugar de solo tener acceso al siguiente elemento, cada entrada puede acceder también al anterior.

- Pila:

  La propiedad de una pila es que, siempre que se agrega un elemento, se agrega al final; y además, solo se puede eliminar el elemento que se encuentre al final. Es decir, el primer elemento que podrá salir de la pila es el último que ha entrado.

- Cola:

  La propiedad de una cola es que, siempre que se agrega un elemento, se agrega al final; pero solo se puede eliminar el elemento que se encuentre al inicio. Es decir, el primer elemento que podrá salir es el primero que ha entrado.

Estos son sólo ejemplos de algunas de las estructuras de datos básicas, aunque existen muchas más. En abstracto, cada estructura representa la forma en que están *acomodados* los datos, pero en términos reales esto expresa la manera en que se accede a las direcciones de memoria donde están almacenados. Dependiendo del tipo de problema que se esté resolviendo, puede ser más útil cierta estructura en específico, siempre con el objetivo de que sea clara y eficiente la manera de manipular estos datos (modificar entradas, eliminar o agregar elementos, acceder a sus valores, etcétera).

Las características de las estructuras de datos son descritas como propiedades abstractas y hay muchas maneras de que sean implementadas cumpliendo dichas características. Los lenguajes de programación más usados a la fecha suelen incluir implementaciones de las estructuras más conocidas.

Dado que este trabajo no tiene por objetivo el enfoque implementativo, no se ahondará en eso. Para un estudio más profundo del tema se puede consultar [15].

2 Preliminares sobre Geometría Computacional

En este capítulo vamos a revisar algunas nociones específicas sobre geometría computacional que se relacionan con el problema planteado en esta tesis, y que sirven como preámbulo al mismo. También mencionaremos algunos resultados que serán utilizados más adelante como herramienta para su resolución.

Se darán por hecho algunos conceptos de geometría clásica euclidiana, tales como punto, recta, segmento de recta, ángulo y polígono.

En adelante, estaremos trabajando sobre conjuntos finitos de puntos en el plano, para lo que introduciremos algunas definiciones y notaciones.

Sea $P \subset \mathbb{R}^2$ con cardinalidad $|P| = n$, donde $n \in \mathbb{N}$.

Notación 2.1. *Podemos referirnos a P como un n-*conjunto.

Recordemos que dos puntos distintos siempre definen una recta (Axiomas de Euclides). Sin embargo, para tres o más puntos, puede o no existir una recta que pasa por todos ellos simultáneamente.

Definición 2.1. *A tres o más puntos por los que pasa una misma recta, se les llama* colineales.

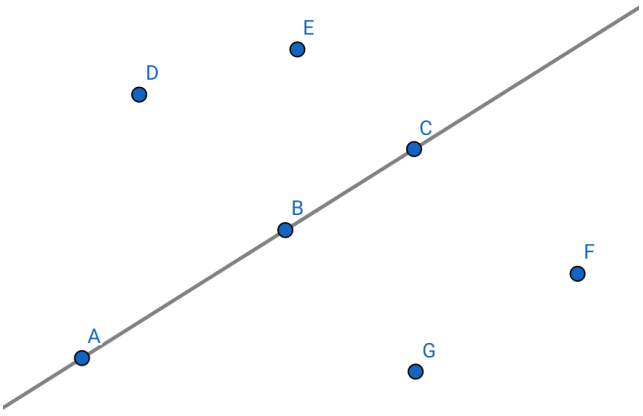

Figura 2.1: A, B y C son colineales.

Definición 2.2. *Decimos que P está en* posición general[1] *si no existen tres puntos de P que sean colineales.*

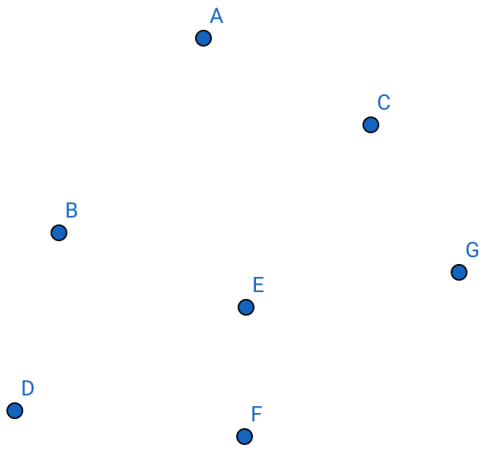

Figura 2.2: Conjunto de puntos en posición general.

A partir de aquí muchos de los resultados de los que hablaremos consideran únicamente conjuntos de puntos en posición general.

[1]El concepto de *posición general* puede ser aplicado en otros contextos o para otra clase de objetos geométricos, y a lo que se refiere siempre es a que no existan relaciones de dependencia innecesarias entre ellos; como en el caso mencionado anteriormente, la relación de colinealidad.

2.1. Ordenar conjuntos de puntos

Cuando tenemos un conjunto de puntos, por lo regular consideramos sus coordenadas en los ejes x y y. Para ordenar al conjunto normalmente basta con ordenar, respecto a alguna de las coordenadas, utilizando cualquier algoritmo de ordenamiento de números. (Ver sección 1.4).

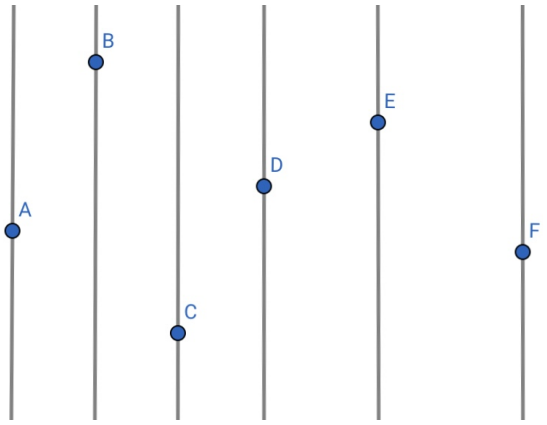

Figura 2.3: Puntos ordenados respecto a su coordenada x.

Sin embargo, hay otro tipo de ordenamiento de puntos que, de hecho, nos será útil en la resolución del problema planteado en esta tesis; este es el *ordenamiento angular*. Para ordenar angularmente debemos considerar un punto fijo de referencia, digamos $p \in P$, y el orden de los puntos será dado por el tamaño de los ángulos formados entre una dirección fija, digamos la horizontal, y los segmentos formados con el punto p. Obtener la lista de ángulos se puede hacer en tiempo lineal y, como sabemos, el ordenamiento nos tomará $\Theta(n \log n)$.

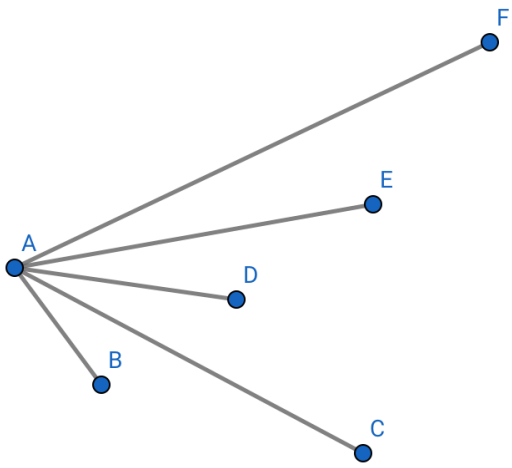

Figura 2.4: Puntos ordenados angularmente respecto al punto A.

2.2. Convexidad y cierre convexo

En todas las áreas matemáticas relacionadas con geometría es usual utilizar el concepto de *convexidad*.

Definición 2.3. *Decimos que* $P \subset \mathbb{R}^2$ *es un* conjunto convexo *si para cada dos elementos* $x, y \in P$ *el segmento* $\overline{xy} \subset P$.

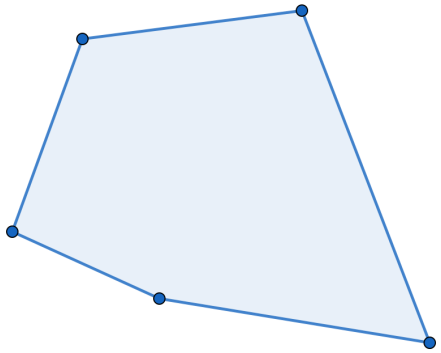

Figura 2.5: Conjunto convexo

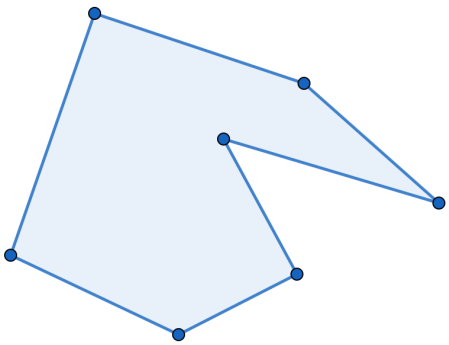

Figura 2.6: Conjunto no convexo

Además nos interesa el mínimo conjunto convexo que contiene a un conjunto dado, al cual se le denomina *cierre convexo*, (también se conoce como casco convexo, envolvente, envoltura, o cerradura convexa). Intuitivamente podemos pensar en una liga o banda elástica que abarca a todo el conjunto P, que es soltada y se ajusta a su alrededor.

Definición 2.4. *El* cierre convexo *de un conjunto* P *es la intersección de todos los conjuntos convexos que contienen a* P *y lo denotamos como* $CC(P)$.

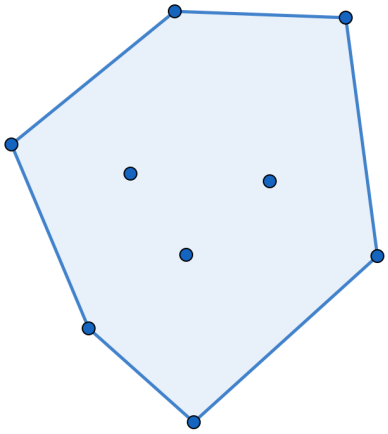

Figura 2.7: Cierre convexo de un conjunto de puntos.

Notemos que, si P tiene sólo un punto, su cierre convexo es él mismo; si tiene sólo dos puntos, $CC(P)$ es el segmento que los une; y si P es un n-conjunto de puntos en posición general, con $n \geq 3$, entonces $CC(P)$ es un polígono convexo, donde cada punto de P puede o no ser vértice de dicho polígono. En ocasiones, el concepto de cierre convexo se usa para referirse específicamente a la frontera de ese polígono; en este caso vamos a denotar como $CC^*(P)$ a dicha frontera. Es destacable el caso en que todos los puntos de P pertenecen a $CC^*(P)$.

Definición 2.5. *Decimos que un conjunto de puntos P está en* posición convexa *si cada punto de P pertenece a CC^*P.*

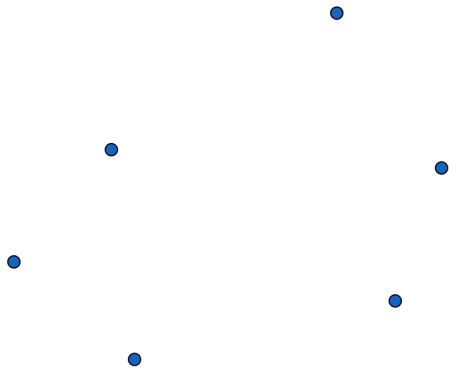

Figura 2.8: Conjunto de puntos en posición convexa.

2.3. Algoritmos para calcular el cierre convexo

Un problema clásico en geometría computacional es el de calcular el cierre convexo de un conjunto de puntos. Vamos a ver dos algoritmos que lo resuelven, los cuales además serán usados para la resolución del problema planteado en esta tesis.

2.3.1. Cierre convexo triangulado con barrido de línea

Para explicar este algoritmo requerimos definir lo que es una triangulación.

Definición 2.6. *Una* triangulación *de un conjunto de puntos P es una subdivisión de $CC(P)$ en triángulos, en la que cada triángulo tiene por vértices a puntos de P y no contiene ningún elemento de P en su interior.*

Vamos a describir un algoritmo que encuentra el cierre convexo de P construyendo una triangulación. Primero es necesario darles un orden a los puntos, tomaremos en este caso un orden de izquierda a derecha, es decir, orden ascendente de la coordenada x de cada punto. Para fines de este orden, vamos a considerar que no hay puntos que coincidan en la misma coordenada x. Sean $p_1, p_2, ...p_n$ los puntos de P etiquetados en dicho orden.

La construcción se hará de forma inductiva. Para los puntos p_1, p_2, p_3 su cierre convexo es el triángulo que los tiene como vértices. Una vez que se tiene el cierre convexo de los primeros i puntos, llamémoslo C_i, se puede encontrar el de los primeros $i + 1$ de la siguiente forma: Agregar un segmento de recta desde p_{i+1} hacia cada uno de los puntos de C_i que sean *visibles* desde p_{i+1}, es decir, para los que dicho segmento no atraviese el interior de C_i. Esto nos deja construido el cierre C_{i+1}.

Este algoritmo se puede consultar más a detalle en [18].

A este tipo de técnica, en que se van procesando los puntos en el orden de alguna de sus coordenadas, se le conoce como *barrido de línea*, o bien, *barrido de plano*. Esto hace referencia a que podemos pensar en una recta vertical, que se va desplazando sobre el plano, y cada uno de los puntos del conjunto P es una *parada* del barrido, en la que se ejecuta alguna acción. El número de paradas es $|P| = n$, por lo que la complejidad de un algoritmo de este tipo depende de n y del tiempo que tome la acción realizada en cada parada.

Notemos que mediante este algoritmo, la forma de construir el cierre convexo es añadiendo punto a punto bajo un determinado orden y, de forma análoga, es posible *desmantelar* dicho cierre convexo, siguiendo el orden inverso.

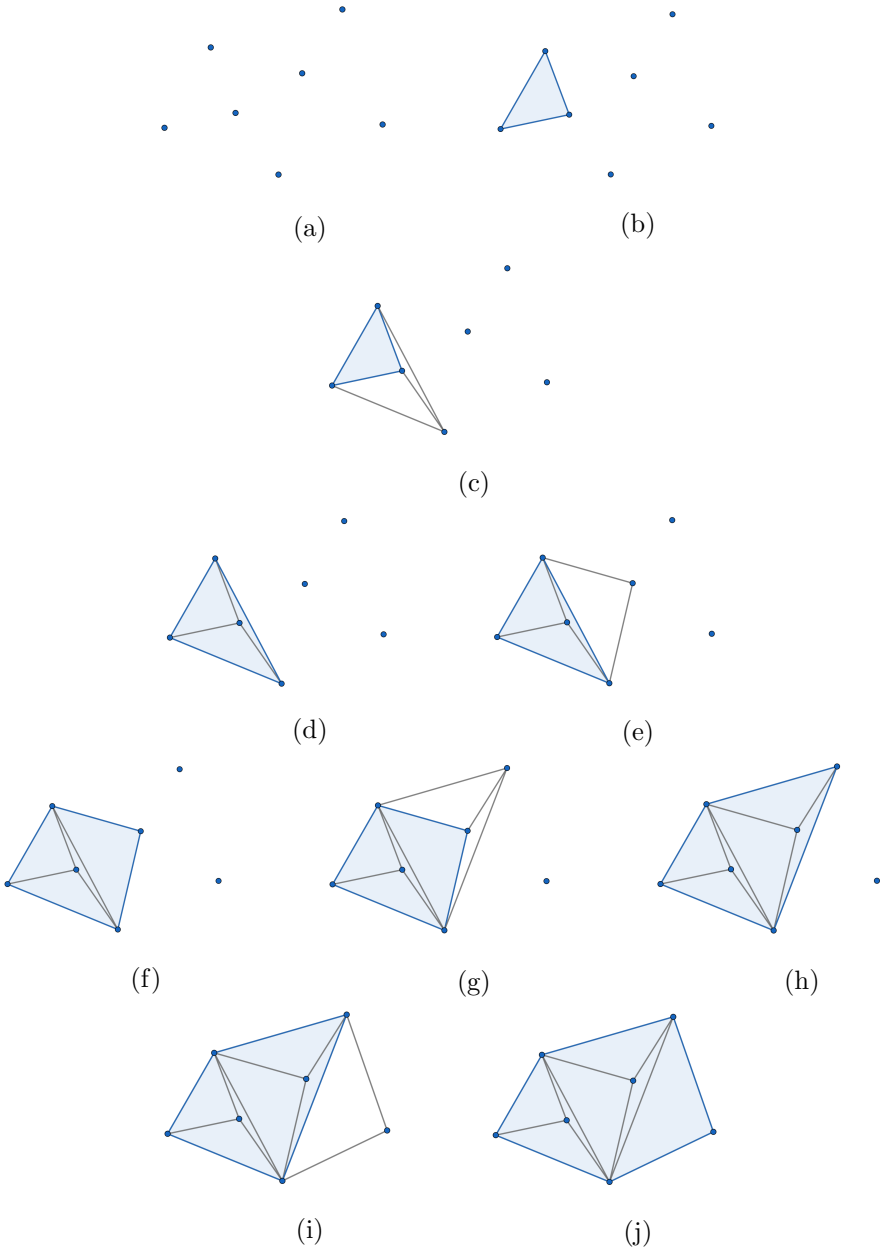

Figura 2.9: Construcción del cierre convexo triangulado de un conjunto de puntos.

2.3.2. Algoritmo de Graham

Este algoritmo, también conocido como *Graham Scan* o *Escaneo de Graham*, debe su nombre a Ronald Graham, quien lo publicó en 1972. Es un método que sirve para encontrar, en un conjunto de puntos, a todos aquellos que pertenecen a la frontera de su cierre convexo.
El primer paso del algoritmo es encontrar al punto más a la izquierda y (en segundo lugar de prioridad) más bajo del conjunto P. Esto se puede hacer con un recorrido de las coordenadas x de los puntos, para encontrar al que tiene menor coordenada x y, en caso de haber más de un mínimo, buscar entre ellos al que tiene menor coordenada y. Al punto encontrado lo llamaremos p_0. Esto es claro que tomará tiempo lineal.

El siguiente paso será ordenar el resto de los puntos angularmente respecto a p_0, y en ese orden los nombraremos $p_1, p_2, ...p_{n-1}$ de manera que las pendientes de $\overline{p_0p_1}$, $\overline{p_0p_2}$, ..., $\overline{p_0p_{n-1}}$ están en orden creciente.

Una vez dado este orden, vamos a hacer un recorrido, observando el ángulo formado por cada tres puntos. Por cómo fueron ordenados los puntos respecto a p_0: el primer ángulo, formado por p_0, p_1 y p_2, siempre va a formar un *giro a la izquierda*. Notemos que, si se recorrieran en ese orden los puntos que pertenezcan a la frontera del cierre convexo, precisamente por la convexidad, todos los ángulos serán de giro a la izquierda. Con esta propiedad en mente, vamos a evaluar cada punto para determinar si pertenece o no a la frontera del cierre convexo.

Vamos a almacenar en una pila S a los puntos que forman la frontera del cierre convexo. Usamos esta estructura para poder almacenar puntos en calidad de *candidatos* y si es necesario, eliminarlos de ella.

Comenzaremos almacenando en S a los puntos p_0 y p_1, que necesariamente forman parte del cierre convexo; después almacenamos a p_2 como candidato. Nombraremos como $s_1, s_2, \ldots, s_k$, en orden, a los elementos de la pila de candidatos.

Para cada punto p_i, con $i \geq 3$, revisamos el ángulo formado con s_{k-1} y s_k, y si éste forma un giro a la izquierda, se agrega p_i como candidato a la pila, ya que al agregarlo se preserva la convexidad. En caso contrario, si forman un giro a la derecha, significa que el último candidato, s_k, no es en realidad parte del cierre convexo, por lo que debe ser removido de la estructura. En este caso, el punto actual p_i debe ser evaluado ahora según el ángulo que forma con los dos anteriores en la pila, s_{k-2} y s_{k-1}, y nuevamente se toman las mismas decisiones dependiendo del giro que forma.

Observemos que, en algún paso, necesariamente p_i será insertado como candidato, y podemos garantizar que, cuando esto suceda, ya se habrán descartado todos los puntos anteriores p_j con $j < i$ que definitivamente no formarán parte del cierre convexo.

Al terminar este proceso para cada p_i con $3 \leq i \leq n$, los puntos que estén almacenados en la pila S nos darán, en orden, los puntos de la frontera del cierre convexo. Notemos que p_0, p_1 y p_n siempre formarán parte de ésta.

Observemos que cada punto p_i es insertado y removido de la estructura a lo más una vez, y el procesamiento para decidir cuándo se agrega y cuándo se elimina toma tiempo constante, por lo que todo el recorrido para determinar los puntos de la frontera del cierre convexo toma tiempo lineal. Como al inicio es necesario haber ordenado los puntos, el algoritmo completo tiene complejidad de tiempo $\Theta(n \log n)$.

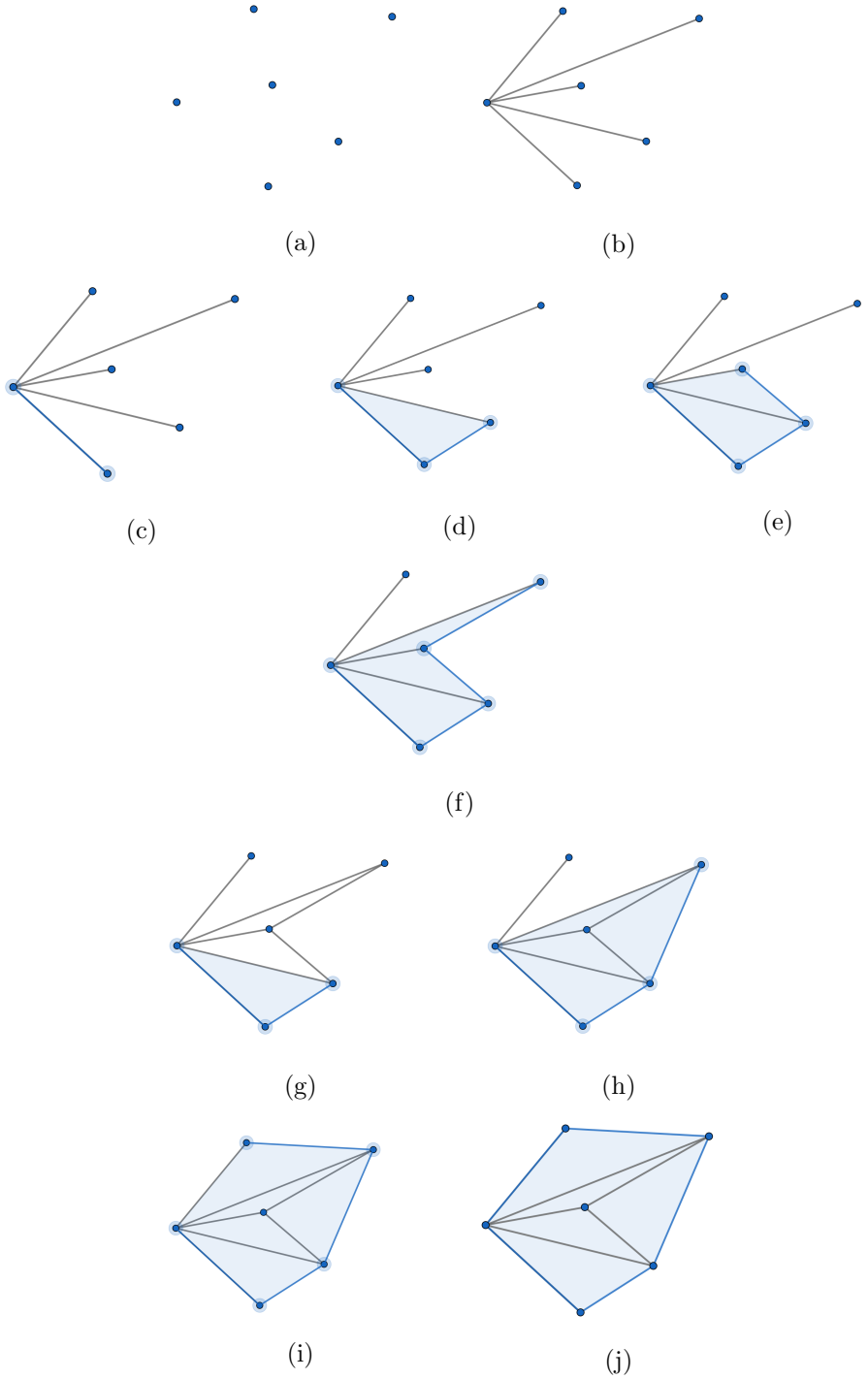

Figura 2.10: Construcción del cierre convexo de un conjunto de puntos, por el algoritmo de Graham.

2.4. Transformación dual

Una técnica bastante utilizada en geometría computacional es un tipo de transformación en el plano que mapea puntos a rectas y rectas a puntos. Esta, en ocasiones, es útil para reducir la complejidad de la resolución de algunos problemas.

Considerando que los puntos en el plano están definidos con dos valores que corresponden a sus coordenadas en x y en y, diremos que el punto p es la pareja ordenada (p_x, p_y). Por otro lado, una recta en el plano se puede definir mediante su ecuación de recta $y = mx+b$, es decir, también son dos parámetros los que definen a cada recta, el valor de m, que corresponde a la pendiente, y el valor de b que corresponde a la ordenada al origen, es decir el valor de la coordenada y en que la recta intersecta al eje de las ordenadas.
La transformación T está definida como sigue:

- Para un punto $p = (a, b)$,
  $T(p)$ es la recta definida por la ecuación $y = ax - b$.
- Para una recta $l : y = cx + d$,
  $T(l)$ es el punto con coordenadas $(c, -d)$.

Notemos que esta dualidad no considera rectas verticales, ya que no existe un valor finito para su pendiente, y por lo tanto tampoco un valor finito para la coordenada en el eje x del que sería su punto dual correspondiente.

Veamos ahora las razones por las que esta transformación nos será de mucha utilidad más adelante.

Proposición 2.1. *Se cumplen las siguientes propiedades para T.*

1.- *La composición de T consigo misma es la identidad, es decir $T(T(z)) = z$, siendo z un punto o una recta.*

2.- *Un punto p pertenece a una recta l si y sólo si el punto $T(l)$ pertenece a la recta $T(p)$.*

3.- *Un punto p está por encima (análogamente, por debajo) de una recta l si y solo si el punto $T(l)$ está por encima (respectivamente, por debajo) de la recta $T(p)$.*

La propiedad 3 de la proposición 2.1 se puede interpretar como que la transformación T preserva el orden pero invierte la orientación.

Para los propósitos del cuarto capítulo de la tesis, nos interesará aplicar esta transformación dual a todos los puntos de un n-conjunto dado, P.

Notación 2.2. *Sea $P = \{p_1, p_2, \ldots, p_n\}$, denotamos como $T(P)$ al conjunto de rectas $\{T(p_1), T(p_2), \ldots, T(p_n)\}$.*

Análogamente, si L es un conjunto de rectas, $T(L)$ es el conjunto de puntos obtenidos de aplicar la transformación T a cada una de las rectas.

Por practicidad, al tratar con conjuntos de puntos y de rectas que fueron obtenidos, uno mediante la transformación T aplicada al otro (el orden no es relevante por la propiedad 1 de la proposición 2.1), simplemente decimos son conjuntos duales.

Veamos ahora algunas propiedades interesantes al considerar un conjunto de puntos y un conjunto de rectas que son duales.

Proposición 2.2. *Sea P un n-conjunto de puntos en el plano y $L = T(P)$ su dual. Se cumplen las siguientes propiedades:*

1.- *L no contiene rectas verticales.*

2.- *L está en posición general, esto es, no tiene tres rectas concurrentes si y sólo si P está en posición general, esto es, no tiene 3 puntos colineales.*

3.- *L no contiene rectas paralelas si y sólo si todos los puntos de P tienen distinto valor en su coordenada x.*

2.5. Arreglos de rectas

Para poder utilizar esta transformación dual para conjuntos de puntos, como lo haremos en el algoritmo propuesto en el capítulo 4, es necesario dotar de estructura a los conjuntos de rectas, para esto proporcionaremos en esta sección algunas definiciones y notaciones.

Consideremos un conjunto L de n rectas en el plano en posición general, sin rectas verticales y sin rectas paralelas.

Definición 2.7. *Llamaremos* vértice *al punto de intersección de cada dos rectas.*

Definición 2.8. *Llamaremos* arista *a cada segmento de recta entre dos vértices consecutivos (esto es, vértices que están en una misma recta y sin otro vértice intermedio). Consideraremos también* aristas no acotadas*, estas son las semirectas que quedan con solo un vértice extremo.*

Definición 2.9. *Por último, llamaremos* cara *a cada region del plano acotada por aristas y sin aristas en su interior.*

La estructura completa, que visualmente es equivalente al conjunto de rectas, se conoce como *arreglo de rectas* y tiene muchas ventajas al momento de estudiar, diseñar o implementar algoritmos con estos conjuntos.

Definición 2.10. *Dado un conjunto L de rectas en el plano, la subdivisión del plano considerando vértices, aristas y caras, inducida por L se conoce como* arreglo *inducido por L y se denota como $A(L)$.*

Definición 2.11. *Un arreglo de rectas es* simple *si no hay tres de ellas que se intersecan en un mismo punto.*

Mencionemos también algunas posibles relaciones destacables entre los objetos definidos para un arreglo de rectas.

Definición 2.12. *Decimos que un vértice y una arista son* incidentes *si el vértice es extremo de esa arista, y que una arista y una cara son* incidentes *si esa arista es parte de la frontera de la cara.*

Definición 2.13. *Decimos que dos vértices son* adyacentes *si son los extremos de una misma arista, y que dos caras son* adyacentes *si comparten alguna arista.*

El conjunto L tiene n rectas, sin embargo, al utilizar su arreglo inducido $A(L)$, se están considerando más objetos y nos interesa conocer la cantidad de ellos, por lo que es útil definir la *complejidad* de un arreglo de rectas.

Definición 2.14. *La* complejidad *de $A(L)$ es la cantidad total de vértices, aristas y caras del arreglo.*

La cantidad de vértices está dada por la cantidad de puntos de intersección de las rectas que, por ser un arreglo simple, solo se intersecarán dos a dos, y por lo tanto es igual a la cantidad de posibles parejas del conjunto, esto es

$$\binom{n}{2} = \frac{n(n-1)}{2}.$$

Para contabilizar a las aristas, notemos que cada una de las rectas es intersecada por el resto, esto es por $n-1$, y entre cada dos de esas intersecciones se forma una arista, ahí van $n-2$ aristas, y además se consideran las dos aristas no acotadas, esto nos da un total de n aristas en cada recta. Por lo tanto, hay n^2 aristas en total.

Para contar el número de caras, lo podemos pensar de manera inductiva. Como base, consideremos que el plano sin rectas consta de una región o cara. Además, si consideramos un arreglo de $i-1$ rectas y añadimos una i-ésima, está aportará i nuevas aristas, a partir de sus intersecciones con las demás rectas, y cada una de sus aristas dividirá en dos a una región existente. Por lo tanto, al agregar la i-ésima recta, se aumentan i caras al arreglo. De manera que el conteo total de caras es:

$$1 + \sum_{i=1}^{n} i = 1 + \frac{n(n-1)}{2} = \frac{n^2 - n + 2}{2}.$$

Notemos que este conteo es exacto por las condiciones que dimos sobre el conjunto de rectas: que no haya 3 concurrentes y que no haya rectas paralelas; ya que gracias a eso, cada dos rectas aportan una intersección y cada intersección corresponde únicamente a dos rectas. Para arreglos que no cumplan dichas condiciones los totales aquí obtenidos son cota superior.

Proposición 2.3. *Si L es un conjunto de n rectas, la complejidad del arreglo $A(L)$ es $O(n^2)$.*

A partir de esto, tenemos el siguiente teoerema, el cual será usado más adelante, para demostrar la complejidad del algoritmo principal de esta tesis.

Teorema 2.1. *El arreglo de rectas correspondiente a la transformación dual de un conjunto de puntos se puede construir en tiempo $O(n^2)$.*

3 Resultados previos

En este capítulo mencionaremos algunos resultados existentes en el área, que ayudarán a poner en contexto el problema que será planteado y a destacar su relevancia.

En geometría computacional, muchos resultados clásicos tienen que ver con configuraciones de puntos en el plano, de los cuales se pueden considerar distintas propiedades de interés, por ejemplo, las distancias entre ellos, emparejamientos, triangulaciones, polígonos convexos, polígonos vacíos, etcétera. Comúnmente se busca garantizar existencia o contabilizar subestructuras con determinada característica. Si además usamos que los puntos estén coloreados, dichas subestructuras las podemos restringir describiendo alguna característica cromática.

3.1. m-ágonos convexos

En el capítulo anterior, en la sección 2.2 se mencionó la definición de convexidad y en específico de conjuntos de puntos en posición convexa. Sobre esto, existe un resultado clásico de 1935, dado por Paul Erdös y George Szekeres [20]. Este resultado generaliza otro, dado por Esther Klein en la misma época, el cual mostramos en la siguiente proposición.

Proposición 3.1. *Para toda configuración de* 5 *puntos en el plano en posición general, es posible encontrar* 4 *de ellos que estén en posición convexa.*

Demostración. Consideremos el cierre convexo de los 5 puntos dados, si al menos 4 de ellos pertenecen a la frontera, entonces la conclusión es trivial. Así que, supongamos que son solo 3: A,B y C, y los dos restantes: D y E están al interior de ΔABC. Observemos que dos vértices del triángulo, digamos A y B, quedan del mismo lado de la recta que pasa por D y E, y por esta razón, es claro que $ABDE$ es un cuadrilátero convexo.

□

Así mismo, fue planteada por la misma Klein la siguiente pregunta, en busca de una generalización de este resultado.

Pregunta 3.1. *Dado un entero positivo m ¿existe un valor $f(m)$ que garantice que todo conjunto de $f(m)$ puntos en el plano contiene m de ellos en posición convexa?*

Erdös y Szekeres mostraron que la respuesta es afirmativa.

Teorema 3.1 (Erdös-Szekeres)**.** *Para cada entero positivo $m > 3$, existe un mínimo entero $f(m)$ tal que todo n-conjunto, con $n \geq f(m)$, contiene un subconjunto de m puntos en posición convexa.*

Notemos que hacemos mención a m puntos en posición convexa, o a un m-ágono convexo, de manera indistinta.

Definición 3.1. *Llamamos* m-ágono *a un polígono simple de m vértices.*

Notación 3.1. *Al mínimo entero $f(m)$ se le denota como $ES(m)$ y se conoce como el* número de Erdös-Szekeres *correspondiente a m.*

Aunque el teorema 3.1 fue dado en 1935, a la fecha se conocen pocos valores exactos del parámetro que nombramos como ES. Es fácil ver que $ES(3) = 3$ y sabemos que $ES(4) = 5$, por la demostración de Klein. Más tarde, fue publicado por Kalbfleisch [27] que $ES(5) = 9$.

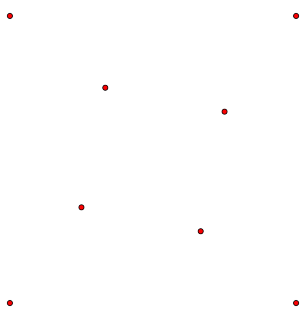

Figura 3.1: Conjunto de 8 puntos que no contiene 5 en posición convexa. Esto muestra que $ES(5) \geq 9$

En 2006, Szekeres y Peters [35] probaron utilizando herramientas computacionales que $ES(6) = 17$. Para valores de $m \geq 7$ no se conocen los números de Erdös-Szekeres.

Con esto se observa que, incluso habiendo garantizado la existencia, encontrar el valor exacto de estos parámetros es realmente otro problema y de gran dificultad.

Erdös y Szekeres también dieron las siguientes cotas:

$$2^{m-2} + 1 \leq ES(m) \leq \binom{2m-4}{m-2} + 1.$$

La cota inferior es la mejor que se ha dado hasta ahora y, por los valores conocidos de ES, se conjetura que es cota justa. La cota superior, $\binom{2m-4}{m-2} + 1$, como cota asintótica, es de orden de $4^{m-o(m)}$ y en 2017 fue mostrado por Andrew Suk en [34] que $ES(m) = 2^{m-o(m)}$.

Por otro lado, en 1973 Erdös y Guy [19] se plantearon contabilizar las apariciones de los polígonos convexos, planteando la siguiente pregunta:

Pregunta 3.2. *Dados dos enteros positivos m y n, ¿cuál es la mínima cantidad de m-ágonos convexos que se garantiza que se forman en cualquier conjunto de n puntos en el plano?*

Para $m = 3$, es trivial la solución $\binom{n}{3}$. A partir de $m = 4$, se vuelve bastante más complicado y se relaciona con un problema conocido como *número de cruces rectilíneos*, del cual no hablaremos en este trabajo, pero se puede consultar el tema en [1], [2] y [3].

3.2. m-hoyos

A partir del teorema de Erdös-Szekeres han surgido diversas variantes. Un problema muy directamente relacionado fue planteado por Erdös en 1978, con la ligera modificación de buscar que el polígono convexo sea vacío, es decir, que no contenga en su interior puntos del conjunto.

Definición 3.2. *Sea P un conjunto de puntos en el plano en posición general. Un m-subconjunto S de P es llamado un m-*hoyo *si los puntos de S son vértices de un polígono sin puntos de P en su interior.*

El problema es el siguiente:

Pregunta 3.3. *¿Existe un entero $h(m)$, para cada entero $m \geq 3$, tal que todo n-conjunto, con $n \geq h(m)$ contiene un m-hoyo convexo?*

Para $m \leq 5$ la respuesta es afirmativa. De hecho, la demostració del caso $m = 4$ se puede ver directamente de la demostración original de Klein, ya que dicha prueba es exhaustiva y en todos los casos aparece también un cuadrilátero convexo vacío. Por lo tanto, $h(4) = ES(4) = 5$. En 1978 Harborth [24] mostró que $h(5) = 10$. Sin embargo, años más tarde Horton [25] probó que, si $n = 7$, no existe un valor finito que cumpla con lo requerido, es decir, que existen conjuntos de puntos arbitrariamente grandes que no contienen algún subconjunto de 7 puntos que sean vértices de un polígono convexo y vacío.

El caso $m = 6$ estuvo abierto durante mucho tiempo, hasta que en 2007 y 2008, Nicolás [31] y Gerken [23] respectivamente, probaron de manera independiente la existencia de $h(6)$. En [32], se muestra que $h(6) \geq 30$, exhibiendo un conjunto de 29 puntos que no contiene hexágonos convexos vacíos.

Erdös planteó nuevamente contabilizar las apariciones que se pueden garantizar en cualquier conjunto de puntos.

Pregunta 3.4. *Dados dos enteros positivos m y n, ¿cuál es la mínima cantidad de m-hoyos convexos que se garantiza que se forman en cualquier conjunto de n puntos en el plano?*

Por el resultado de Horton [25], sabemos que para $m \geq 7$ nunca se garantiza la existencia de m-hoyos convexos. Para los casos $3 \leq m \leq 6$ se han dado cotas

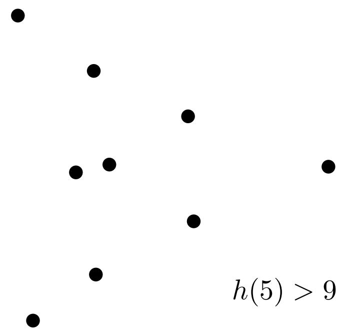

Figure 1: A set of 9 points with no 5-hole.

Figura 3.2: Conjunto de 9 puntos sin pentágono convexo vacío.

tanto inferiores como superiores. Las mejores cotas hasta la fecha se pueden consultar en [10] y [16].

Erdős y Szekeres también plantearon la generalización de estos problemas a dimensiones más altas, sin embargo, no se han obtenido muchos resultados. Así mismo, en el plano se han considerado otras variantes. En la siguiente sección mostraremos específicamente la modificación en que los puntos sean coloreados.

3.3. Coloraciones

Muchas de las variantes a los problemas clásicos mencionados se basan en considerar que los conjuntos de puntos sean coloreados.

Definición 3.3. *Una* coloración *es una función finita, $f : P \to \{c_1, c_2, ...c_k\}$. A los elementos de la imagen se les conoce como* colores *y a sus respectivas preimágenes se les conoce como* clases cromáticas.

Una coloración no es más que una función, y el conjunto de clases cromáticas es una partición, sin embargo, la noción de coloración es útil en ocasiones para plasmar visualmente esta característica haciendo uso tal cual de colores.

Una coloración con k colores se conoce como k-coloración y también se dice que el conjunto está k-coloreado.

Definición 3.4. *Sea S un conjunto coloreado de puntos en el plano, en posición general. Supongamos que cada S_i es no vacío y tiene asignado el* color i. *Decimos que un subconjunto $T \subset S$ es:*

- ***monocromático*** *si todos sus elementos son del mismo color.*
- ***policromático*** *si hay elementos de colores distintos.*
- ***heterocromático*** *si todos sus elementos son de colores distintos.*
- ***balanceado*** *si tiene la misma cantidad de elementos de cada color.*

see [18] and [22] for the current bounds. But even in the plane many variants have been considered. Let us only mention a few examples. Bisztriczky and Fejes Tóth [8] proved a generalization replacing points with convex bodies. Bialostocki *et al.* [7], Caro [9] and Károlyi *et al.* [20] gave results on a conjecture (see [7]) according to which there is a number $h(m, q)$, for arbitrary positive integers $m \geq 3$ and q, such that any n-set, $n \geq h(m, q)$, contains an m-set for which the number of interior points is divisible by q. Several authors have studied the number of subsets in convex position of a given size that a sufficiently large point set can have [4, 6, 10, 23, 29, 30, 32, 35]. Let us finally mention the papers by Ambarcumjan [2], Hosono and Urabe [17], Károlyi [18] and Urabe [34], where several issues on partitioning a point set into subsets in convex position are considered.

Let $S = S_1 \cup \cdots \cup S_k$ be a partition of a planar point set S, in general position in the plane. We will assume that each set S_i is non-empty, and will refer to it as the set of points of *color i*. A subset $T \subset S$ is called *monochromatic* if all its points have the same color, and *polychromatic* otherwise. The term *heterochromatic* is used for sets in which every element has a different color.

In this paper we consider the following collection of problems. Given an integer m and a set S as above, possibly with additional requirements for $n = |S|$ to be large enough, can we find an m-hole of S falling into one of the three described chromatic classes? Or an m-subset in convex position?

Although colored versions of several results concerning finite point configurations have already been considered by many authors (see e.g. [1, 3, 11, 36]), the original motivation for us to study these problems came from a

A partir de esta definición surgen nuevas variantes al problema, que fueron inicialmente estudiadas por Devillers, Hurtado, Károlyi y Seara [17].

Problema 3.1. *Dado un entero k, ¿podemos asegurar que todo conjunto S suficientemente grande contiene un k-hoyo con alguna de las características cromáticas de la definición anterior? ¿O un k- subconjunto en posición convexa?*

Notemos que los problemas sobre conjuntos de puntos coloreados serían análogos a los anteriormente expuestos. Vamos a dar nueva notación para los parámetros.

Notación 3.2. *Denotamos como $\eta_M(m,k)$ (respectivamente $\eta_H(m,k)$ y $\eta_P(m,k)$) al mínimo entero con la propiedad de que cualquier configuración con esa cantidad de puntos, en posición general y k-coloreados contiene un m-subconjunto en posición convexa monocromático (respectivamente heterocromático y policromático).*

Notación 3.3. *Denotamos como $MC(m,n,k)$ (respectivamente $HC(m,n,k)$ y $PC(m,n,k)$ a la máxima cantidad de m-hoyos monocromáticos (respectivamente heterocromáticos y policromáticos) que se puede garantizar en cualquier n-conjunto k-coloreado.*

Mencionaremos como ejemplo algunos de los resultados más importantes sobre este tipo de variantes.

En [17] se muestra el valor de las funciones $\eta_M(m,k)$, $\eta_H(m,k)$ y $\eta_P(m,k)$.

Respecto a hoyos en esta variante cromática, recordemos el resultado de Horton, que afirma que si $m \geq 7$, n puede ser arbitrariamente grande sin que se garantice la existencia de 7-hoyos convexos en todo n-conjunto. Es fácil ver que esto se cumple por igual en conjuntos coloreados y eso significa que para toda $m \geq 7$:

$$MC(m,n,k) = HC(m,n,k) = PC(m,n,k) = 0.$$

Para ahondar más en diversos problemas de este tipo que han sido resueltos, se pueden consultar [4], [6], [7], [11], [13], [22], [26], [28],[29], [33] y [37].

Con el objetivo de poner en contexto el resultado de esta tesis, mencionaremos un poco de lo que se ha estudiado para los valores $m = 3$ y $m = 4$.

Está probado que todo conjunto de 10 puntos, 2-coloreado, contiene un triángulo monocromático vacío. Más aún, que cualquier conjunto de n puntos contiene al menos $\lfloor (n-1)/9 \rfloor$ triángulos monocromáticos vacíos.

En 2009, Aichholzer, Fabila-Monroy, Flores-Peñaloza, Hackl, Huemer y Urrutia [4] probaron la cota de $\Omega(n^{5/4})$ y poco después fue mejorada por Pach y Tóth a $\Omega(n^{4/3})$ [33].

La mejor cota superior asintótica para $MC(3,n,2)$ es $O(n^2)$ y se cree que es cota justa.

Conjetura 3.1. *En un conjunto bicoloreado de n-puntos existe una cantidad cuadrática de triángulos monocromáticos vacíos.*

Se sabe también que $MC(3, n, 3) = 0$, es decir, que hay conjuntos 3-coloreados arbitrariamente grandes sin triángulos monocromáticos vacíos.

Recientemente, Fabila-Monroy, Perz y Trujillo [21] mostraron que, si $k \geq 3$ y $n = km$ (la coloración es balanceada), entonces $HC(m, n, k) = \Theta(k^3)$. Esto es, que se garantiza la existencia de una cantidad cúbica (respecto al número de colores) de triángulos heterocromáticos. Observemos que en este caso, podría aumentar el número de puntos sin crecer la cantidad de apariciones.

También fue mostrado que para $m \geq 4$, existen conjuntos de puntos sin m-ágonos vacíos heterocromáticos.

Aichholzer et al. [5] probaron que para n suficientemente grande se asegura la existencia de cuadriláteros vacíos monocromáticos.

Devillers et al. [17] mostraron que para $k \geq 5$ y para cualquier n, existen configuraciones de n puntos sin k-hoyos monocromáticos.

Por otro lado, también se han estudiado problemas del tipo Erdös-Szekeres bajo una variación del concepto de convexidad.

Con lo anterior queda claro que se pueden considerar gran cantidad de variantes de problemas tipo Erdös-Szekeres, se ha estudiado en dimensiones mayores, y con diversas restricciones, variando la cantidad de colores y las características de la estructura de interés.

Destaquemos que, cuando no se puede garantizar la existencia de la estructura buscada, es decir, cuando se prueba que hay conjuntos arbitrariamente grandes que no contienen a dicho objeto, entonces se vuelve interesante poder detectar si la estructura aparece o no en un conjunto particular. También es interesante contabilizar la cantidad de apariciones de objetos en ambos casos: ya sea quelos se garantizan en todo conjunto, o los que aparecen en un conjunto dado. Aquí es donde juega un papel fundamental la computación, ya que se busca desarrollar algoritmos eficientes para resolver este tipo de cuestiones. En el siguiente capítulo, se muestra un planteamiento de este tipo, cuyo resultado de existencia es negativo, y se estudia el problema de detectar su aparición.

4 Planteamiento y resolución del problema

En este capítulo se plantea el problema original de esta tesis y se proporciona un algoritmo para resolverlo, utilizando algunas de las herramientas mencionadas en capítulos anteriores.

Consideremos un n-conjunto, P, de puntos en el plano en posición general, coloreados con 4 colores. Notemos que, por el resultado de Erdös-Szekeres (teorema 3.1), si $n \geq 5$, P necesariamente contiene un cuadrilátero convexo; y de la misma manera, con que una de las clases cromáticas tenga al menos 5 puntos contendrá uno monocromático.

Sin embargo, veamos que, sin importar el valor de n, P no necesariamente contiene un cuadrilátero convexo heterocromático, es decir, con sus 4 vértices de colores distintos, incluso cuando las clases cromáticas tengan la misma cardinalidad.

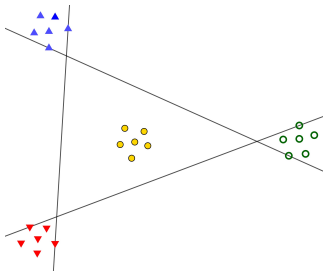

Figura 4.1: Existen conjuntos arbitrariamente grandes de puntos, coloreados con 4 colores, que no contienen algún cuadrilátero convexo heterocromático.

Por esta razón, es interesante pensar en diseñar algoritmos para detectar esos cuadriláteros. Notemos que un algoritmo de fuerza bruta puede tener complejidad de $O(n^4)$, que es la cantidad cuadriláteros que se podrían formar con los puntos de P. Por lo que incluso uno de $O(n^3)$ sería una mejora en eficiencia.

Vamos a dar un algoritmo de tiempo cuadrático que determina si P contiene o no un cuadrilátero convexo heterocromático, lo cual es el principal resultado de este trabajo de tesis.

Enseguida daremos el planteamiento formal del problema y algunas definiciones que serán usadas para la explicación del algoritmo.

4.1. Problema

Problema 4.1. *Sea un conjunto P de puntos en el plano, en posición general, y coloreados con cuatro colores, $P = C_1 \cup C_2 \cup C_3 \cup C_4$. Determinar si existen puntos $a \in C_1$, $b \in C_2$, $c \in C_3$ y $d \in C_4$, en posición convexa.*

Daremos ahora algunas definiciones que serán útiles para dar solución a este probema.

4.2. Definiciones

Definición 4.1. *Dado un punto p en el plano, llamaremos $H^+(p)$ al semiplano superior determinado por la recta horizontal que pasa por p.*

Definición 4.2. *Dado un punto p y un conjunto de puntos $S \in H^+(p)$, vamos a considerar un* orden angular *de los puntos de S con respecto a p, que será el orden que se les da a los elementos de S de acuerdo al ángulo, formado con la horizontal, de cada recta que pasa por cada punto $q \in S$ y por p.*

Notemos que este orden es en contra de las manecillas del reloj, alrededor del punto p.

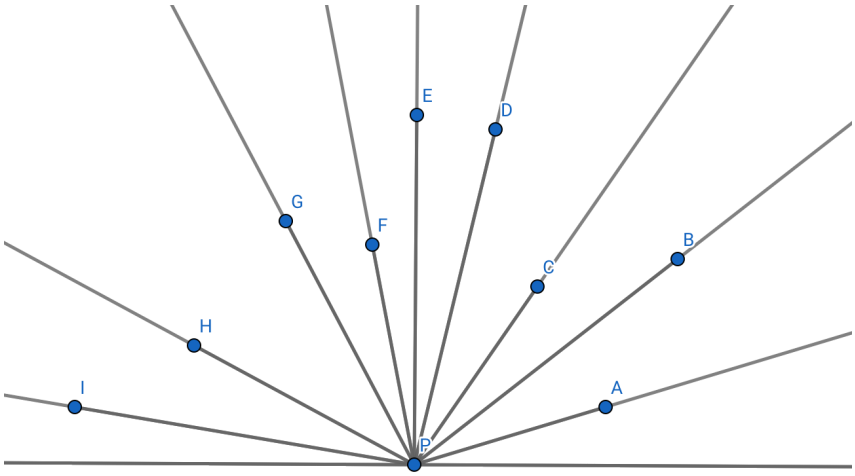

Figura 4.2: Orden angular respecto al punto p.

Definición 4.3. *Llamaremos* barrido angular *a una técnica análoga al barrido de línea, pero en la que las paradas están dadas por el orden angular de los puntos.*

Consideremos dos conjuntos convexos, ajenos. Sabemos que tienen 4 rectas tangentes en común, dos de ellas exteriores y dos interiores. Notemos que, siempre que las tangentes externas no sean rectas verticales, hay una de ellas que queda por abajo de los conjuntos y otra por arriba.

Definición 4.4. *Dados dos conjuntos A y B, en caso de existir la tangente exterior por abajo a sus correspondientes cierres convexos (ver párrafo anterior), a esta recta la llamaremos* recta de soporte *de A y B.*

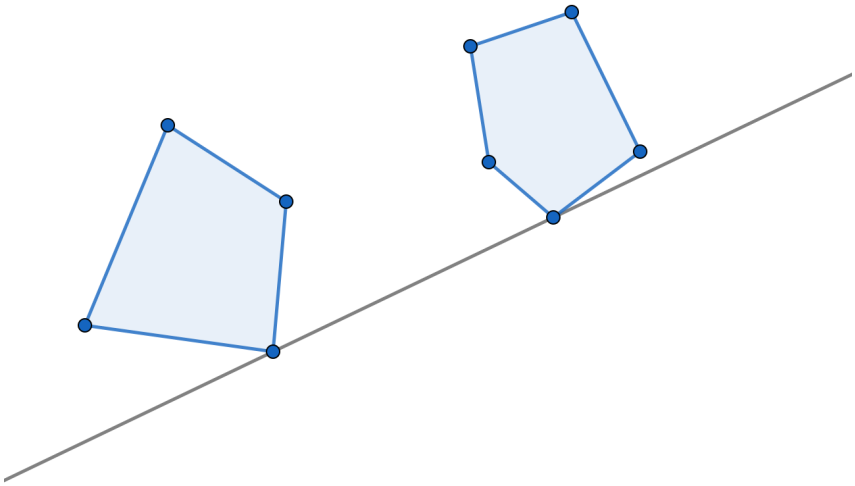

Figura 4.3: Dos polígonos convexos y su recta de soporte.

4.3. Algoritmo

Proponemos los siguientes pasos:

1.- Fijemos un punto $p \in P$. Buscaremos detectar si existe un cuadrilátero convexo heterocromático cuyo vértice más bajo es p.

2.- Fijemos el orden angular (respecto a p) de la aparición de los colores restantes en los otros tres vértices del cuadrilátero.

3.- Sin pérdida de generalidad, suponemos que $p \in C_1$, y los vértices de colores C_2, C_3 y C_4, aparecen en ese orden (angularmente respecto a p).

4.- Consideremos a los vértices de color distinto al color de p que están en el semiplano $H^+(p)$, $Q = (P \setminus C_1) \cap H^+(p)$.

Nombremos a las clases cromáticas de Q:

$$C_2' = C_2 \cap H^+(p)$$

$$C_3' = C_3 \cap H^+(p)$$

$$C_4' = C_4 \cap H^+(p)$$

5.- Consideremos al conjunto Q con su orden angular respecto a p. En la siguiente sección daremos los detalles de cómo hacer este ordenamiento.

6.- Haremos un barrido angular sobre los puntos de Q, de manera que en cada parada correspondiente a un punto $q \in Q$ mantegamos el cierre convexo de los puntos de C_2' que están antes que q (incluyéndolo) y el cierre convexo de los puntos de C_4' que están después que q, así como la recta de soporte de dichos cierres. Posteriormente profundizaremos en cómo es posible mantener esas estructuras.

7.- Durante el mismo barrido, si un punto de parada es $q \in C_3'$, verificaremos si está por encima de la recta de soporte actual. En caso de que sí, entonces sabremos que existe un cuadrilátero convexo heterocromático con el orden de colores deseado. A saber, el cuadrilátero formado por los vértices p, q y los dos extremos de la recta de soporte considerada en ese paso.

4.4. Ordenamiento angular

Dado un punto p en el plano y un conjunto Q de puntos en $H^+(p)$, el orden angular de Q con respecto a p es el orden de los elementos de Q, dado por los ángulos respecto a la horizontal de las rectas que contienen a cada uno de los segmentos pq, para cada $q \in Q$.

Por el teorema 2.1 sabemos que es posible obtener el arreglo de rectas de la transformación dual en tiempo cuadrático. Veamos que, a partir de esto, se puede obtener el ordenamiento angular respecto a p en tiempo lineal.

Teorema 4.1. *Dado el arreglo de rectas correspondiente a la transformación dual T de un conjunto de puntos, es posible ordenarlo angularmente respecto a un punto p, en tiempo $O(n)$.*

Demostración. Recordemos que, bajo la transformación dual, tanto el punto p como los puntos de Q corresponden a rectas, y por las propiedades de esta transformación, sabemos que la recta que pasa por los puntos p y q en el primal, digamos $l(\overline{pq})$, corresponde en el dual al punto de intersección entre las rectas $T(p)$ y $T(q)$. Es decir, $T(l(\overline{pq})) = T(p) \cap T(q)$. Además sabemos que al aplicar a una recta la transformación T, el punto obtenido tendrá como coordenada x el valor de la pendiente de la recta original, con esto concluimos que para cada punto $q \in Q$ la pendiente de la recta $l(\overline{pq})$ es la coordenada x del punto de intersección entre las rectas $T(p)$ y $T(q)$. Como la pendiente está directamente relacionada con su ángulo, nos basta con conocer el orden (por su coordenada x) de esos puntos de intersección para saber el orden angular de los elementos de Q con respecto a p. Por la estructura de almacenamiento del arreglo de rectas, sabemos que es posible acceder a todos los vértices de una misma recta, en este caso $T(p)$, en tiempo lineal y los obtendremos en el orden relativo a sus coordenadas. A partir de esto, únicamente es necesario hacer un ajuste para determinar cuál de los segmentos correspondientes forma el menor ángulo respecto a la horizontal, ya que no necesariamente es el que tiene la menor abcisa. Esto no aumenta la complejidad.

4.5. Preprocesamiento para la construcción de cierres convexos triangulados

En la seción 2.3 se describió un algoritmo para construir un cierre convexo triangulado de un conjunto de puntos, teniendo en cuenta un orden de los puntos, en ese momento se dio por hecho que el orden era respecto a la coordenada x y se utilizó la técnica de barrido de línea. Notemos que se puede seguir un algoritmo análogo utilizando un orden angular de los puntos y la técnica de barrido angular dada por ese orden. El tiempo de construcción de igual manera será lineal y una vez construido se puede *desmantelar* por capas en tiempo lineal.

Recordemos que en la descripción del algoritmo de detección, presentado en la sección 4.3, fijamos a un punto p y el orden de aparición de los colores, y llamamos Q al conjunto de puntos de color diferente al de p, los siguientes pasos ahí mencionados, corresponden a una subrutina para determinar si existe un cuadrilátero convexo heterocromático que tenga a p como punto más bajo y los otros tres puntos, visitados en orden angular respecto a p tengan colores C_2, C_3 y C_4 respectivamente.

El primer paso de esa subrutina fue ordenar a los puntos de Q angularmente respecto a p, lo cual fue explicado en la sección anterior, el siguiente paso se enunció textualmente como sigue:

"Haremos un barrido angular sobre los puntos de Q, de manera que en cada parada correspondiente a un punto $q \in Q$ mantegamos el cierre convexo de los puntos de C_2' que están antes que q (incluyéndolo) y el cierre convexo de los puntos de C_4' que están después que q, así como la recta de soporte de dichos cierres. Posteriormente profundizaremos en cómo es posible mantener esas estructuras".

Durante ese barrido se está considerando que en cada paso conocemos a los cierres convexos de dos conjuntos que se están actualizando constantemente, para poder hacer esto requerimos de un preprocesamiento con los puntos de color C_2 y los puntos de color C_4 que pertenecen a Q. Éste usa también la técnica de barrido angular.

1.- En un primer barrido consideramos a los puntos de C_2' en su orden angular con respecto al punto p, construimos su cierre convexo triangulado. Notemos que, por construcción, en la i-ésima parada se tiene al cierre convexo de los primeros i puntos de C_2'.

2.- En un segundo barrido, consideraremos a los puntos de C_4' pero en el orden contrario al orden angular que tienen respecto a p, de esta manera, en la j-ésima parada de este barrido inverso se tendrá el cierre convexo de los últimos j puntos de C_4'.

De esta manera, en el barrido final, para cada punto $q \in Q$, si hay i puntos de C_2' antes de q y j puntos de C_4' después de q, conocemos exactamente a los cierres convexos que nos interesan.

Notemos que este preprocesamiento se puede hacer en tiempo lineal, ya que son dos barridos con una cantidad lineal de paradas cada uno y en la explicación del algoritmo de cierre convexo por barrido de línea, vimos que cada parada toma tiempo constante.

4.6. Barrido angular para la actualización de cierres convexos y rectas de soporte

Vamos a describir ahora al barrido que sí es parte del algoritmo, dando por hecho que antes se hizo el preprocesamiento.

Lo más importante de que en cada parada del barrido se tengan actualizados los cierres convexos correspondientes a los conjuntos de color C_2 y C_4 que están, respectivamente, antes y después del punto de parada, es tener actualizada también a su recta de soporte, con la finalidad de que cuando una parada del barrido corresponda a un punto $r \in Q$ de color C_3, podamos comprobar si este queda por encima de la recta de soporte actual y en caso de que la respuesta

sea afirmativa, podemos asegurar que el punto r, el punto p y los dos extremos de la recta de soporte actual, forman un cuadrilátero convexo heterocromático que cumple con las condiciones establecidas (tiene como punto más bajo a p y los otros tres puntos, visitados en orden angular respecto a p, tienen colores C_2, C_3, C_4 respectivamente).

Sabemos que este barrido se hace sobre todos los puntos de Q, que incluyen puntos de color C_2, C_3 y C_4, veamos los pasos a realizar en cada parada del barrido, según el color del punto correspondiente.

En la parada que corresponde al punto $q \in Q$:

Caso 1. Si q es de color C_2, agregamos a q al cierre convexo anterior de color C_2, y revisamos si es necesario actualizar a la recta de soporte.

Caso 2. Si q es de color C_3, los cierres convexos y su recta de soporte no cambian, únicamente revisamos si q está por encima de la recta de soporte actual, y de ser así, el algoritmo termina con una respuesta afirmativa, pues como ya vimos, esto exhibiría la existencia de un cuadrilátero convexo heterocromático.

Caso 3. Si q es de color C_4, eliminamos a q del cierre convexo anterior de color C_4, y revisamos si es necesario actualizar a la recta de soporte.

La decisión de si es necesario actualizar a la recta de soporte, es fácil de comprobar. En el caso 1, esto sucederá cuando q quede por debajo de la recta de soporte que se tenía en el paso anterior, ya que como ahora q también pertenece al cierre convexo de color C_2, ahora q será extremo de la nueva recta de soporte. En el caso 3, será necesario actualizar a la recta de soporte solamente si q era el extremo de la recta de soporte en el paso anterior, ya que q será eliminado del cierre convexo considerado de color C_4.

Notemos que estas actualizaciones no exceden de tiempo lineal en el barrido completo.

4.7. Pruebas

Vamos a demostrar que el algoritmo es correcto y que la complejidad de su tiempo de ejecución es $O(n^2)$. En el caso en que la respuesta del algoritmo es afirmativa es porque, para un punto p, se exhibió un cuadrilátero convexo heterocromático cuyo punto más bajo es p y este es el testigo del problema. En el caso en que la respuesta es negativa significaría que se ejecutó el procedimiento para todos los puntos $p \in S$ y con ninguno de estos se formó el cuadrilátero buscado.

Teorema 4.2. *Existe un cuadrilátero convexo heterocromático en S si y solo si el algoritmo da respuesta afirmativa.*

Demostración. La implicación de regreso es clara porque cuando el algoritmo responde afirmativo exhibe al cuadrilátero. Demostraremos la otra implicación por contradicción, supongamos que existe (al menos) un cuadrilátero convexo heterocromático en S y que el algoritmo da respuesta negativa. Tomemos un cuadrilátero convexo heterocromático en particular, al que llamaremos R, y sean a, b, c y d sus vértices en sentido antihorario y, sin pérdida de generalidad, consideremos que a es el que tiene la menor ordenada de los cuatro. Observemos que en algún momento del algoritmo se tomó al punto a como el punto p fijo; y como b, c, y d tienen colores distintos entre sí y distintos al de a, en algún momento se consideró ese orden de colores. En el barrido angular, al pasar por el punto c se tendrán computados el cierre convexo de los puntos del color de b que aparecen antes de c (incluyéndolo) y el cierre convexo de los puntos del color de d que estén después de c, como sabemos que el algoritmo dio respuesta negativa, entonces c queda por debajo de la recta de soporte de esos dos cierres. Por la definición de recta de soporte, esto implica que cualquier otra recta que una puntos contenidos en esos cierres convexos también está por encima de c. Sin embargo, por el orden presupuesto de los puntos b, c y d en sentido antihorario y por la convexidad de R, tenemos que c está por encima del segmento bd, lo cual nos lleva a una contradicción. En otras palabras, si el algoritmo no detectara a $R = abcd$ en la evaluación con a como punto más bajo y los colores de bcd en ese orden, el algoritmo de todas maneras daría respuesta afirmativa exactamente en el momento en que el barrido se detiene en c.

□

Teorema 4.3. *El algoritmo tiene complejidad en tiempo total de $O(n^2)$.*

Demostración. Por el teorema 2.1 de la sección 2.4, el preprocesamiento que corresponde a la transformación dual tiene complejidad $O(n^2)$ y teniendo el arreglo de rectas que corresponde a dicha transformación podemos obtener, respecto a cualquier punto, el orden angular de un subconjunto de puntos en tiempo $O(n)$. Una vez fijo un punto p, y un orden de los 3 colores restantes, se obtiene el orden mencionado en $O(n)$, y se hace el barrido completo con las actualizaciones correspondientes también en $O(n)$. Esto se repite para cada punto del conjunto (n veces) y para cada posible orden de 3 colores (6 veces). La complejidad de este proceso será $6n(O(n) + O(n)) = O(n^2)$.

□

5 Conclusiones y trabajo futuro

Como principal conclusión a este trabajo, primeramente quiero destacar la relevancia de la interacción entre las matemáticas y las ciencias de la computación en un área como geometría computacional.

- Por un lado, vemos que en geometría combinatoria y discreta, incluso para los problemas puramente teóricos, puede ser de gran importancia la utilización de recursos computacionales. Se sabe que recientemente es cada vez más común que las cotas para parámetros geométricos sean encontradas por medio de programas computacionales, y es claro que estos deben ser diseñados de forma eficiente para obtener soluciones en tiempo razonable.

- En contraparte, podemos observarque, en el aspecto aplicado de la computación, desde que se busca un manejo de objetos geométricos en la computadora, se vuelve relevante el uso de estructuras de datos y algoritmos que permitan almacenar y manipular dichos objetos de manera eficiente. Esto implica haber llevado a cabo un estudio formal de los problemas.

- En conjunto, los dos puntos anteriores nos muestran la relevancia del estudio de la geometría computacional tanto en un aspecto teórico como aplicado.

- Es interesante observar el fenómeno de cómo a partir de un problema clásico se plantean tantas variantes como se puede, lo cual lleva a estudiar un mismo tipo de resultado en distintas formas, en ocasiones colindantes con diversas áreas de las matemáticas y de las ciencias de la computación.

Creemos que el algoritmo aquí mostrado (sección 4.3), para detectar cuadriláteros convexos heterocromáticos en conjuntos de puntos 4-coloreados, es óptimo en complejidad. Sin embargo, no ha sido demostrado aún.

Por lo tanto, el trabajo pendiente en relación a este tema podría ser:

- Mostrar que el algoritmo es óptimo, o bien, obtener uno de menor complejidad.

- Considerar una modificación al problema, pidiendo además de convexidad que el cuadrilátero sea vacío.

- Estudiar el problema análogo para pentágonos convexos heterocromáticos en conjuntos de puntos 5-coloreados.
- Contabilizar la cantidad de apariciones para cada uno de los objetos mencionados.
- Estudiar el problema en dimensiones superiores.

Bibliografía

[1] Bernardo M Ábrego, Silvia Fernández-Merchant, Jesús Leaños, and Gelasio Salazar. A central approach to bound the number of crossings in a generalized configuration. *Electronic Notes in Discrete Mathematics*, 30:273–278, 2008.

[2] Bernardo M Ábrego, Silvia Fernández-Merchant, Jesús Leaños, and Gelasio Salazar. The maximum number of halving lines and the rectilinear crossing number of K_n for $n \leq 27$. *Electronic Notes in Discrete Mathematics*, 30:261–266, 2008.

[3] Bernardo M Ábrego, Silvia Fernández-Merchant, and Gelasio Salazar. The rectilinear crossing number of K_n: Closing in (or are we?). In *Thirty essays on geometric graph theory*, pages 5–18. Springer, 2013.

[4] Oswin Aichholzer, Ruy Fabila-Monroy, David Flores-Peñaloza, Thomas Hackl, Clemens Huemer, and Jorge Urrutia. Empty monochromatic triangles. *Computational geometry*, 42(9):934–938, 2009.

[5] Oswin Aichholzer, Ruy Fabila-Monroy, Hernán González-Aguilar, Thomas Hackl, Marco A Heredia, Clemens Huemer, Jorge Urrutia, and Birgit Vogtenhuber. 4-holes in point sets. *Computational geometry*, 47(6):644–650, 2014.

[6] Oswin Aichholzer, Thomas Hackl, Clemens Huemer, Ferran Hurtado, and Birgit Vogtenhuber. Large bichromatic point sets admit empty monochromatic 4-gons. *SIAM Journal on Discrete Mathematics*, 23(4):2147–2155, 2010.

[7] Oswin Aichholzer, Jorge Urrutia, and Birgit Vogtenhuber. Balanced 6-holes in linearly separable bichromatic point sets. *Electronic Notes in Discrete Mathematics*, 44:181–186, 2013.

[8] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[9] Imre Bárány and Gyula Károlyi. Problems and results around the Erdös-Szekeres convex polygon theorem. In *Japanese Conference on Discrete and Computational Geometry*, pages 91–105. Springer, 2000.

[10] Imre Bárány and Pavel Valtr. Planar point sets with a small number of empty convex polygons. *Studia Scientiarum Mathematicarum Hungarica*, 41(2):243–269, 2004.

[11] Sergey Bereg, José Miguel Díaz-Báñez, R Fabila-Monroy, Pablo Pérez-Lantero, A Ramírez-Vigueras, Toshinori Sakai, Jorge Urrutia, and Inmaculada Ventura. On balanced 4-holes in bichromatic point sets. *Computational Geometry*, 48(3):169–179, 2015.

[12] Tibor Bisztriczky and G Fejes Tóth. A generalization of the Erdös-Szekeres convex n-gon theorem. *Journal für die reine und angewandte Mathematik*, 1989(395):167–170, 1989.

[13] Peter Brass. Empty monochromatic fourgons in two-colored point sets. *Geombinatorics*, 14(2):5–7, 2004.

[14] S Barry Cooper. *Computability theory*. CRC Press, 2003.

[15] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[16] Knut Dehnhardt. *Leere konvexe Vielecke in ebenen Punktmengen*. PhD thesis, Uitgever niet vastgesteld, 1987.

[17] Olivier Devillers, Ferran Hurtado, Gyula Károlyi, and Carlos Seara. Chromatic variants of the Erdös–Szekeres theorem on points in convex position. *Computational Geometry*, 26(3):193–208, 2003.

[18] Herbert Edelsbrunner. *Algorithms in combinatorial geometry*, volume 10. Springer Science & Business Media, 2012.

[19] Paul Erdös and Richard K Guy. Crossing number problems. *The American Mathematical Monthly*, 80(1):52–58, 1973.

[20] Paul Erdös and George Szekeres. A combinatorial problem in geometry. *Compositio mathematica*, 2:463–470, 1935.

[21] Ruy Fabila-Monroy, Daniel Perz, and Ana Laura Trujillo-Negrete. Empty rainbow triangles in k-colored point sets. *Computational Geometry*, page 101731, 2020.

[22] Erich Friedman. 30 two-colored points with no empty monochromatic convex fourgons. *Geombinatorics*, 14:53–54, 2004.

[23] Tobias Gerken. Empty convex hexagons in planar point sets. *Discrete & Computational Geometry*, 39(1-3):239–272, 2008.

[24] Heiko Harborth. Konvexe fünfecke in ebenen punktmengen. *Elemente der Mathematik*, 33:116–118, 1978.

[25] Joseph D Horton. Sets with no empty convex 7-gons. *Canadian Mathematical Bulletin*, 26(4):482–484, 1983.

[26] Clemens Huemer and Carlos Seara. 36 two-colored points with no empty monochromatic convex fourgons. *Geombinatorics*, 19(1):5–6, 2009.

[27] JD Kalbfleisch. A combinatorial problem on convex regions. In *1970 Proc. Louisiana Conf. on Combinatorics, Graph Theory and Computing*, pages 180–188, 1970.

[28] VA Koshelev. On erdös–szekeres problem and related problems. *ArXiv e-prints*, 2009.

[29] Lingling Liu and Yuqin Zhang. Almost empty monochromatic quadrilaterals in planar point sets. *Mathematical Notes*, 103(3-4):415–429, 2018.

[30] Walter Morris and Valeriu Soltan. The Erdös-Szekeres problem on points in convex position–a survey. *Bulletin of the American Mathematical Society*, 37(4):437–458, 2000.

[31] Carlos M Nicolás. The empty hexagon theorem. *Discrete & Computational Geometry*, 38(2):389–397, 2007.

[32] Mark Overmars. Finding sets of points without empty convex 6-gons. *Discrete & Computational Geometry*, 29(1):153–158, 2002.

[33] János Pach and Géza Tóth. Monochromatic empty triangles in two-colored point sets. *Discrete Applied Mathematics*, 161(9):1259–1261, 2013.

[34] Andrew Suk. On the Erdös-Szekeres convex polygon problem. *Journal of the American Mathematical Society*, 30(4):1047–1053, 2017.

[35] George Szekeres and Lindsay Peters. Computer solution to the 17-point Erdös-Szekeres problem. *The ANZIAM Journal*, 48(2):151–164, 2006.

[36] Géza Tóth and Pavel Valtr. Note on the Erdös-Szekeres theorem. *Discrete & Computational Geometry*, 19(3):457–459, 1998.

[37] R Van Gulik. Two-colored points with no empty monochromatic convex fourgons. *Geombinatorics XV*, pages 32–33, 2005.