Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco
Departamento de Computación

Anti-thickness geométrico de gráficas completas con hasta diez vértices.

TESIS QUE PRESENTA

David Gustavo Merinos Sosa

PARA OBTENER EL GRADO DE

Maestro en Ciencias en Computación

DIRECTORA DE LA TESIS

Dra. María Dolores Lara Cuevas

México, Ciudad de México **2019**

Abstract

Resumen

Dedicación

A mi hermano Eduardo, a mis padres María e Isaías, a Jehová mi Dios.

Agradecimientos

Agradezco al CONACYT sin el cual no habría sido posible completar este programa de maestría. Estoy también muy agradecido con mi asesora de tesis, la Doctora María Dolores Lara Cuevas por aceptarme como tesista y apoyarme en todo momento en el desarrollo de este trabajo. Gracias a mis compañeros de generación por las incontables horas de diversión.

Índice general

Capítulo 1

Introducción

Este trabajo está ubicado en el área de la geometría combinatoria y en el área de geometría computacional.

Una gráfica es un conjunto de vértices junto con un conjunto de aristas que unen pares de vértices. Cuando la gráfica tiene todas las aristas posibles decimos que es una gráfica completa. Cuando representamos la gráfica en el plano, es decir, cuando sus vértices son puntos en $\mathbf{R}^2$ y sus aristas son curvas que unen dos puntos, decimos que tenemos un dibujo de la gráfica. Cuando las aristas son todas segmentos de recta, decimos que la gráfica es geométrica. Cuando las aristas del dibujo de la gráfica se intersectan a pares la gráfica es un thrackle.

En el área de geométria combinatoria existe un problema en el cual se busca obtener una descomposición de una gráfica completa con tamaño mínimo y en el que cada uno de los elementos de la partición sea un thrackle. En este trabajo exploramos una solución a este problema para gráficas completas de hasta diez vértices utilizando herramientas de la geometría computacional. Este problema, que recibe el nombre de anti-thickness de una gráfica completa K_n y se denota como $At_g(K_n)$ ha sido estudiado con anterioridad para gráficas geométricas cuyos vértices están en posición convexa (Fabila-Monroy *et al.* (2018a)). A esta variación del problema la denotamos como $At_c(K_n)$. Se sabe que

$$At_c(K_n) = n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor.$$

Dichos resultados fueron encontrados usando técnicas combinatorias y geométricas. Para el caso en el que los puntos de la gráfica completa están en posición

general existen las siguientes cotas:

$$\frac{n-1}{2} \leq At_g(K_n) \leq n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor .$$

La cota superior proviene del caso en el que los puntos están en posición convexa mientras que la cota inferior proviene del hecho de que un thrackle máximo tiene a lo más n aristas. Sin embargo, esto significaría que cuando los puntos están en posición general los thrackles máximos son disjuntos en aristas. Esto no es verdad en posición convexa, lo cual da lugar para cuestionarnos si en realidad esta cota es justa para posición general.

En esta tesis encontramos que en efecto $At_g(K_n) = n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$ para $n \leq 10$. Nosotros usamos la información que proveen los tipos de orden para conjuntos de hasta diez puntos para inducir una gráfica completa y obtener descomposiciones en thrackles usando algoritmos exhaustivos. Además buscamos información acerca del número de cruce de las descomposiciones para tratar de explicar las características de los conjuntos de puntos en posición general que alcanzan el anti-thickness geométrico de la posición convexa.

La tesis está organizada de la siguiente manera: en el siguiente capítulo se explican a detalle y de manera más formal las definiciones que usamos en este trabajo y que son necesarias para entender el desarrollo de la tesis, después hacemos un recuento de los resultados obtenidos acerca del anti-thickness geométrico y tratamos de explicar el origen del concepto tomando en cuenta un problema propuesto con anterioridad, luego, en la sección de resultados, explicaremos los resultados del trabajo y cómo fueron obtenidos. En esta sección exponemos los algoritmos usados para las búsquedas exhaustivas. Finalmente mencionaremos las conclusiones y posible trabajo futuro.

Capítulo 2

Antecedentes

En este capítulo damos algunas definiciones necesarias para presentar lo que se conoce como descomposición de gráficas completas en thrackles. Este es el principal problema que se trata en esta tesis. Empezamos estableciendo conceptos relacionados con gráficas abstractas y después hablaremos de gráficas en el plano, posteriormente explicamos el concepto de tipo de orden y cómo se utiliza en este trabajo, continuamos hablando del anti-thickness abstracto y del anti-thickness geométrico y finalmente explicamos el número cromático de una gráfica.

2.1. Gráficas

El concepto base del trabajo, del cual se desprenden otras definiciones, es el de gráfica. Todas las definiciones que presentamos en esta sección fueron tomadas de Chartrand & Zhang (2008).

Una *gráfica* G está compuesta por un conjunto no vacío V de objetos a los que llamamos *vértices* y por un conjunto E, de parejas de elementos de V, a los que llamamos *aristas*. Denotamos a la arista e compuesta por los vértices u y v como (u, v). Para describir a la gráfica G compuesta por el conjunto V de vértices y el conjunto E de aristas escribimos $G = (V, E)$. Para referirnos al conjunto de vértices de G escribimos $V(G)$ y para referirnos al conjunto de aristas de G escribimos $E(G)$. En la figura 2.1 presentamos un ejemplo de una gráfica.

Decimos que *dos vértices* $u, v \in V(G)$ *son adyacentes* si existe la arista $(u, v) \in E(G)$. Decimos que *dos aristas* $e_1, e_2 \in E(G)$ *son adyacentes* si inci-

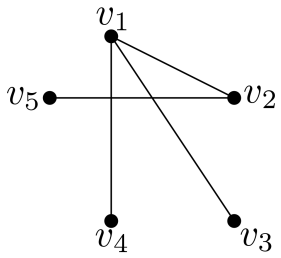


Figura 2.1: Una gráfica con cinco vértices y cuatro aristas. Los vértices v_1 y v_2 son adyacentes y las aristas (v_1, v_3) y (v_1, v_4) son adyacentes.

den en el mismo vértice. La figura 2.1 ilustra un ejemplo de estos conceptos. Una gráfica es *completa* si cada pareja de vértices en la gráfica es adyacente. Mostramos un ejemplo de adyacencia de aristas en la figura 2.1 y un ejemplo de una gráfica completa en la figura 2.2. Para denotar una gráfica completa con n vértices escribimos K_n. Una gráfica G es *bipartita* si es posible dar una partición[1] de $V(G)$ en dos subcojuntos U y W de tal manera que cada arista de G tenga un extremo en U y otro extremo en W. Presentamos un ejemplo de gráfica bipartita en la figura 2.3.

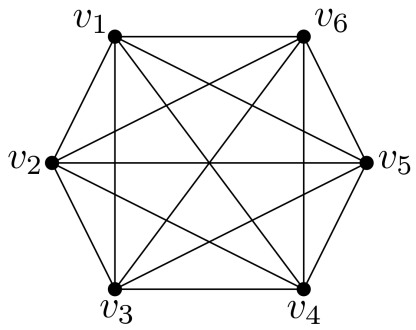


Figura 2.2: La gráfica completa con 6 vértices tiene una arista por cada par de vértices.

[1]Una partición P de un conjunto X es una colección de subconjuntos que cumplen lo siguiente:

- Ningún elemento de P es el conjunto vacío.
- La unión de todos los elementos de P es exactamente el conjunto X.
- La intersección de cualesquiera dos elementos de P es vacía.

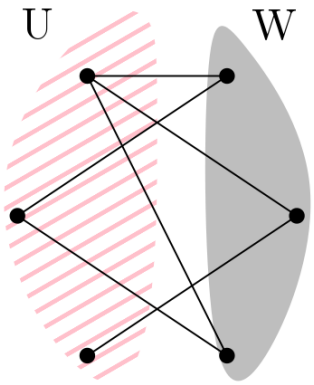


Figura 2.3: Un ejemplo de una gráfica bipartita con bipartición U, W, ambos conjuntos son de tamaño tres.

Una *descomposición* $\mathcal{D}$ de una gráfica G es una colección $\mathcal{D} = \{G_1, G_2, \ldots, G_k\}$ de subgráficas de G, que cumple con dos condiciones:

1. Ninguna subgráfica G_i contiene vértices aislados.
2. Cada arista de G pertenece a exactamente una subráfica G_i de $\mathcal{D}$.

En este trabajo decimos que una gráfica G cubre a una arista e si sucede que $e \in E(G)$.

Nótese que, en otras palabras, las subgráficas de la colección son disjuntas en aristas y su unión cubre a $E(G)$. La figura 2.4 ilustra un ejemplo de una descomposición de la gráfica K_4.

En este trabajo hacemos descomposiciones de dibujos de gráficas (abstractas) en gráficas geométricas que cumplen con cierta propiedad, que explicamos más adelante. En la siguiente sección exponemos el concepto de dibujo de una gráfica y de gráfica geométrica.

2.1.1. Gráfica geométrica

En esta sección abordamos uno de los conceptos clave de este trabajo,que son las gráficas geométricas. Empezamos explicando el concepto de un *dibujo* de una gráfica (abstracta), para continuar con la descripción de un dibujo de una gráfica, con características especiales, al que llamamos *gráfica geométrica*.

Los primeros dos párrafos de esta sección fueron tomados de Pach (2013a). El tercer párrafo fue extraido de Lara & Rubio-Montiel (2019). El cuarto párrafo fue tomado de Pach & Sterling (2011).

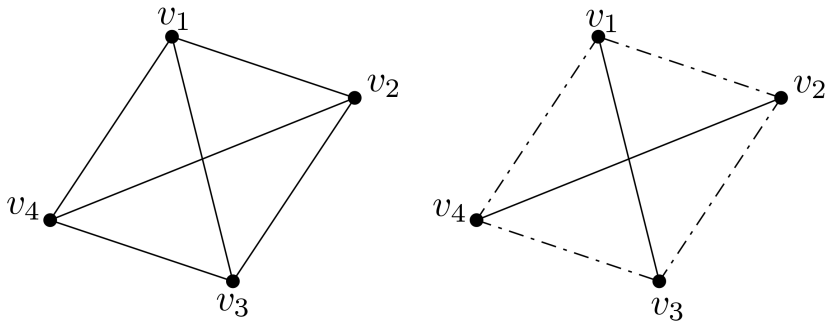


Figura 2.4: Un ejemplo de una descomposición $\mathcal{D}$ de K_4 en dos gráficas. Aquí $\mathcal{D} = \{G_1, G_2\}$ donde G_1 es la gráfica inducida por las aristas $(v_1, v_4), (v_1, v_2), (v_2, v_3), (v_3, v_4)$ y G_2 es la gráfica inducida por las aristas $(v_1, v_3), (v_2, v_4)$.

Un *dibujo* $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ de una gráfica G es una representación de la gráfica G en el plano tal que 1) cada vértice de G es representado por un punto en el plano y 2) cada arista de G es representada como una curva simple continua que conecta un par de puntos. El conjunto de vértices V y el conjunto de aristas E de G son los puntos y las curvas, respectivamente. Sin perdida de generalidad nos referimos al conjunto de puntos de G como $V(\mathsf{G})$, y les llamamos vértices, y nos referimos al conjunto de curvas de G como $E(\mathsf{G})$, y les llamamos aristas.

Cuando restringimos las curvas que representan a las aristas del dibujo de G a segmentos de recta, llamamos al dibujo de la gráfica *gráfica geométrica*. Una gráfica geométrica es completa si existe un segmento de recta entre cada par de vértices de $V(\mathsf{G})$. En la figura 2.5 mostramos un dibujo de K_5 y una gráfica geométrica de K_5. Sea S un conjunto de n puntos en posición general en el plano y sea G una gráfica geométrica de G. Decimos que G está definida sobre S si $V(\mathsf{G}) = S$. Cualquier conjunto S de puntos en posición general induce una gráfica geométrica completa.

Todos los conceptos definidos para gráficas (abstractas) han sido heredados de manera natural para los dibujos de gráficas, sin embargo, como la gráfica geométrica está definida en el plano, es necesario redefinir el concepto de adyacencia de aristas. Decimos que dos aristas $e_1, e_2 \in E(\mathsf{G})$ se *cruzan* si existe un punto p, en alguna de las aristas, tal que en p la arista e_1 pasa de un lado de la arista e_2 hacia el otro lado. Decimos que dos aristas $e_1, e_2 \in E(\mathsf{G})$ son *adyacentes* si comparten un vértice. En este trabajo decimos que dos aris-

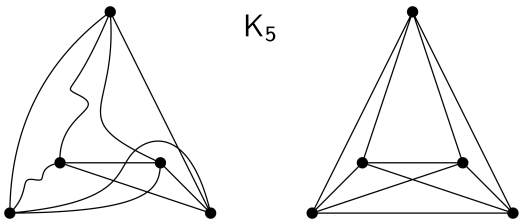


Figura 2.5: A la izquierda observamos un dibujo de K_5 y a la derecha observamos una gráfica geométrica de K_5.

tas de una gráfica geométrica se *intersectan* si son adyacentes o si se cruzan. Mostramos un ejemplo de intersección de aristas en la figura 2.6.

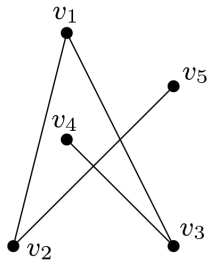


Figura 2.6: En este ejemplo la arista (v_1, v_2) no se intersecta con la arista (v_3, v_4) (son disjuntas) pero sí se intersecta con la arista (v_2, v_5). La arista (v_2, v_5) se cruza con la arista (v_3, v_4) y por lo tanto se intersectan.

El concepto que estudiamos en esta tesis está relacionado con gráficas geométricas donde cada par de aristas se intersectan una vez, estas gráficas geométricas reciben el nombre de thrackles. En la siguiente sección explicamos formalmente qué son los thrackles.

2.1.2. Thrackles

Sea G un dibujo de una gráfica G. Decimos que G es un *thrackle* si cada par de aristas se intersecta exactamente una vez. La figura 2.7 ilustra un ejemplo de thrackle. Los thrackles fueron definidos por John Conway en la decada de 1960 (Pach (2013a)). Conway también conjeturó que el número de aristas en un thrackle no puede exceder el número de sus vértices (Fulek & Pach (2011)). Un thrackle de n vértices es *máximo* si tiene exactamente n aristas. La figura 2.7 muestra un thrackle máximo.

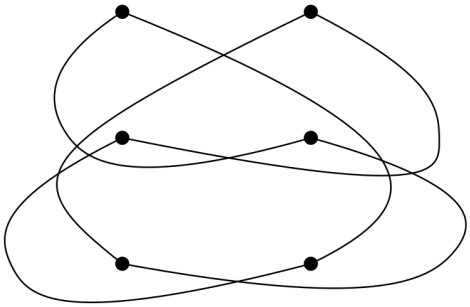
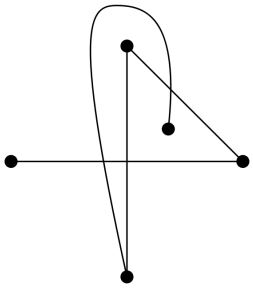
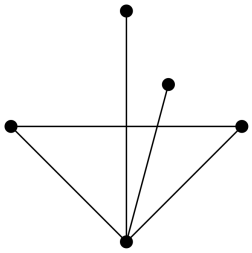


Figura 2.7: Un thrackle máximo sobre un conjunto de seis vértices.



(a) Un thrackle con cinco vértices.



(b) Un thrackle geométrico con cinco vértices.

Figura 2.8: Ambas figuras ilustran thrackles definidos sobre el mismo conjunto de puntos. En los dos casos el thrackle dibujado es máximo.

Una gráfica (abstracta) G es *thrackleable* si puede ser dibujada en el plano como un thrackle.

Una descomposición por thrackles D, de una gráfica geométrica G, es una colección $D = \{\mathsf{G}_1, \mathsf{G}_2, \ldots, \mathsf{G}_k\}$ de subgráficas geométricas que cumple con tres condiciones:

1. Cada subgráfica G_i es un thrackle.

2. Ninguna subgráfica G_i contiene vértices aislados.

3. Cada arista de G pertenece a exactamente una subgráfica G_i de D.

Un thrackle en el que todas sus aristas son segmentos de recta es conocido como *thrackle geométrico*(Schaefer (2018)). En la figura 2.8 presentamos un ejemplo de thrackle y un ejemplo de thrackle geométrico. En este trabajo nos

referimos a los thrackles geométricos como thrackles ya que solo estudiamos descomposiciones de gráficas con thrackles geométricos.

Además definimos la intersección entre dos thrackles como sigue:

Definición 1. *Intersección de dos thrackles.* Sea T_i y T_j dos thrackles con el mismo número de arista, definimos la intersección de T_i y T_j como la intersección de sus conjuntos de aristas correspondientes. En otras palabras:

$$T_i \cap T_j = E(T_i) \cap E(T_j).$$

Decimos que dos thrackles T_i y T_j son disjuntos cuando $T_i \cap T_j = \emptyset$. Además, decimos que un thrackle T es de *tamaño* k si $E(T) = k$.

En este trabajo representamos un thrackle T de tamaño k con una tupla de enteros positivos $\{e_1, e_2, \ldots, e_k\}$ tales que $e_1 < e_2 < \cdots < e_k$. Decimos que $T = \{e_1, e_2, \ldots, e_k\}$ tiene a las aristas $e_1, e_2, \ldots, e_k$. Esta representación nos permite dar un orden para dos thrackles.

Definición 2. *Ordenamiento de dos thrackles.* Sea $T_1 = \{e_1, e_2, \ldots, e_{t_1}\}$ y $T_2 = \{f_1, f_2, \ldots, f_{t_2}\}$ dos thrackles con tamaño t_1 y t_2 respectivamente. Decimos que $T_1 < T_2$ si y solo si existe $e_i \in T_1$ y $f_i \in T_2$ tal que $e_i < f_i$ o bien, si $t_1 > t_2$.

Para ejemplificar la definición anterior supongamos que $T_1 = \{1, 3, 6, 8, 12\}$ y que $T_2 = \{1, 3, 6, 9, 10\}$, en este caso $T_1 < T_2$ ya que $8 < 9$. Por otro lado, si $T_3 = \{1, 3, 4\}$ tenemos que $T_1 < T_3$ porque T_3 tiene tres aristas y T_1 tiene cinco aristas.

Dada una gráfica completa (abstracta) esta puede ser dibujada en el plano de muchas maneras, solo basta con mover un punto en cualquier dirección para obtener diferentes dibujos de la misma gráfica completa, de hecho hay un número no finito de dibujos para una sola gráfica abstracta. Estudiarlos todos no es posible y por ello necesitamos discretizar el número de posibles dibujos geométricos para una sola gráfica. El tipo de orden es una herramienta que otorga un número finito de dibujos combinatoriamente diferentes para gráficas abstractas.Explicamos este concepto a continuación.

2.1.3. Tipo de Orden

Para entender cómo funciona el tipo de orden de un conjunto de puntos debemos definir la orientación de una tripleta de puntos.

Tres puntos (p, q, w) en el plano en posición general, pueden tener una orientación en sentido horario o una orientación en sentido anti-horario. Decimos que (p, q, w) está orientada en sentido horario si w está a la derecha del segmento dirigido $\overline{pq}$. Si w está a la izquierda de dicho segmento entonces (p, q, w) está orientada en sentido anti-horario. Si la tripleta está orientada en sentido horario asignamos a esa tripleta el valor -1. Si la tripleta está orientada en sentido anti-horario asignamos a esa tripleta el valor $+1$. En la figura 2.9 mostramos un ejemplo de cada orientación posible para una tripleta.

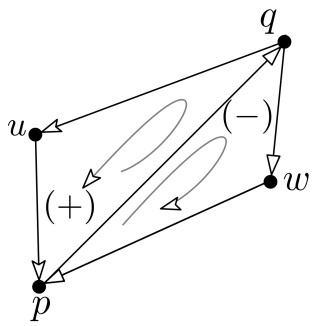


Figura 2.9: Esta figura muestra las posibles orientaciones de una tripleta de puntos. La tripleta $\{p, q, w\}$ tiene asignado el valor de (-1) porque w está orientado en sentido horario con respecto del segmento $\overline{pq}$. La tripleta $\{p, q, u\}$ tiene asignado el valor de $(+1)$ porque u está orientado en sentido anti-horario con respecto del mismo segmento.

Las definiciones siguientes fueron tomadas de Aichholzer *et al.* (2002). El tipo de orden de un conjunto $S = \{p_1, p_2, \ldots, p_n\}$ de puntos en posición general, es una función que asigna a cada tripleta ordenada $i, j, k \in \{1, 2, \ldots, n\}$ la orientación de la tripleta de puntos $\{p_i, p_j, p_k\}$.

Decimos que dos conjuntos S_1 y S_2 son *combinatoriamente equivalentes* si tienen el mismo tipo de orden de otra forma, si no son equivalentes decimos que son *combinatoriamente distintos*. Si S_1 y S_2 son combinatoriamente equivalentes dos segmentos en S_1 se cruzan si y solo si los segmentos correspondientes en S_2 se cruzan. Solo hay una manera (combinatoriamente equivalente) de acomodar tres puntos, dos maneras de acomodar cuatro puntos y tres maneras de acomodar cinco puntos. En la figura 2.10 ilustramos los diferentes tipos de orden para conjuntos de tres y cuatro puntos. En la figura 2.11 presentamos conjuntos combinatoriamente equivalentes de cinco

puntos. En la figura 2.12 mostramos los diferentes tipos de orden para un conjunto de cinco puntos.

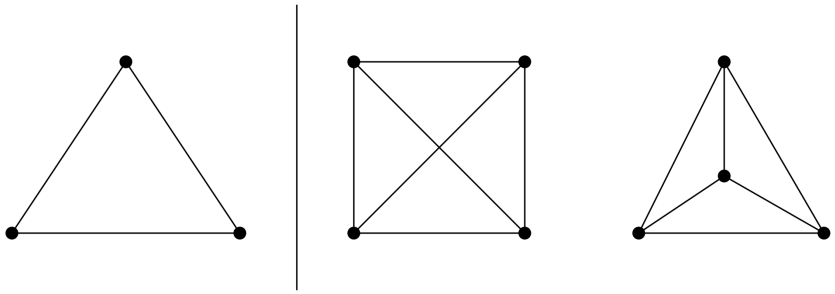


Figura 2.10: Las diferentes maneras combinatoriamente diferentes de acomodar 3 y 4 puntos.

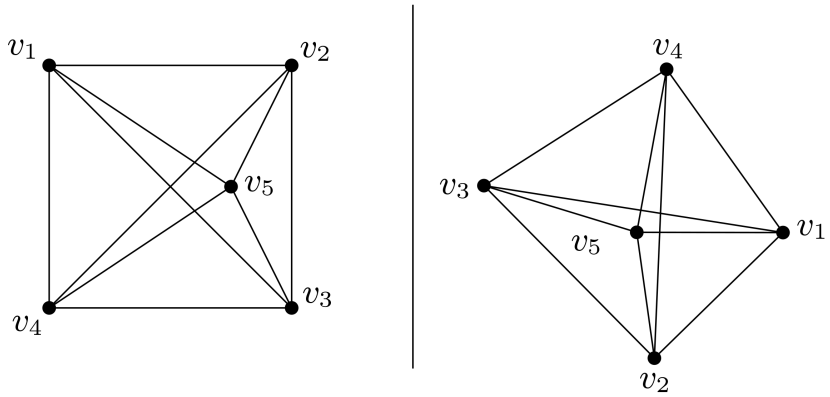


Figura 2.11: Estos dos conjuntos de puntos tienen el mismo tipo de orden. Observe que el valor de cada una de las tripletas del dibujo que está a la izquierda es igual al valor de las tripletas del dibujo a la derecha.

No es trivial enumerar o contar los conjuntos combinatoriamente diferentes, por ejemplo, dados n puntos podemos colocarlos en posición convexa y obtener el primer tipo de orden para n puntos, luego podemos colocar $n-1$ puntos en posición convexa y un punto dentro del $(n-1)$-ágono y evaluar de cuántas maneras combinatoriamente distintas es posible colocar un punto dentro del polígono. Posteriormente podemos ver que pasa con $n-2$ puntos en posición convexa y dos dentro del $(n-2)-$ágono y así sucesivamente hasta que tengamos 3 puntos en posición convexa y $n-(n-3)$ dentro del

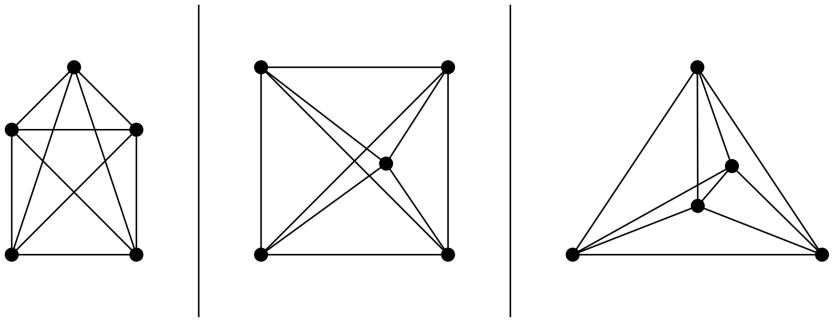


Figura 2.12: Las tres maneras diferentes de distribuir 5 puntos en el plano. Cualquier otra configuración es equivalente a alguna de estas tres configuraciones.

triángulo. En la figura 2.13 explicamos una parte de este proceso. Usando una técnica parecida se sabe que para $n = 10$ hay más de 14 millones de tipos de orden. En el trabajo de Aichholzer *et al.* (2002) se ofrece una base de datos que contiene los conjuntos combinatoriamente diferentes para toda $3 \leq n \leq 10$. En la tabla 2.1 se presenta el número de conjuntos diferentes para cada n y el tamaño en bytes de la base de datos.

En este trabajo buscamos descomposiciones de gráficas geométricas completas en thrackles. Como se mencionó antes, un conjunto de n puntos en posición general induce una gráfica completa de n vértices en el plano. Nosotros analizamos cada tipo de orden para cada $n \leq 10$ induciendo la gráfica completa de n vértices. Después examinamos sus thrackles y luego buscamos una descomposición. Cuando buscamos una descomposición que minimiza el número de thrackles utilizados estamos buscando el anti-thickness de la gráfica. Dicho concepto será explicado formalmente en seguida.

2.2. Anti-thickness y anti-thickness geométrico

Las siguientes definiciones fueron tomadas de Dujmovic & Wood (2017).

Definición 3. *Anti-thickness de una gráfica.* El anti-thickness de una gráfica G es el entero k más pequeño tal que existe una partición de $E(G)$, de tamaño k, en la que cada elemento de la partición es una gráfica thrackleable.

La figura 2.14 ilustra un ejemplo del anti-thickness de K_5.

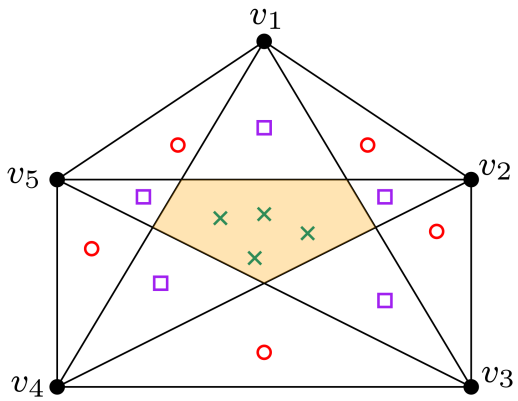


Figura 2.13: La figura ilustra las diferentes maneras de colocar un punto dentro de un pentagono, para formar un conjunto de 6 puntos en total. Hay al menos 3 formas diferentes de colocar dicho punto, 1) es posible ponerlo dentro del pentágono pero fuera del polígono en forma de estrella inducido por las aristas del ciclo interior. 2) Es posible ponerlo dentro de uno de los "picos" del polígono en forma de estrella y 3) es posible ubicarlo en el área rellena. Los puntos de 1) están representados con un círculo sin rellenar, los puntos de 2) con un cuadrado y los puntos de 3) con una cruz. La razón por la que existe más de una ocurrencia de un mismo tipo de punto es porque existe una equivalencia en la etiquetación de los vértices tales que los cruces se preservan.

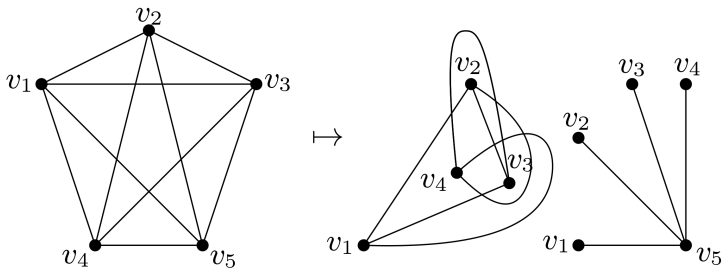


Figura 2.14: La figura muestra a K_5 a la izquierda y a la derecha dos thrackles cuya unión es K_5. Consideremos las aristas de la gráfica completa inducida por los vértices $1, 2, 3, 4$, estas aristas inducen un thrackle mientras que las aristas con un extremo en el vértice 5 inducen otro thrackle. El anti-thickness de K_5 es precisamente igual a dos. Este resultado se discute en el capítulo de resultados.

n	Número de tipos de orden	Tamaño (bytes)
3	1	6
4	2	16
5	3	30
6	16	192
7	135	1890
8	3315	53040
9	158817	5 717 412
10	14309547	572 381 880

Tabla 2.1: Tipos de orden para cada $n \leq 10$.

Cuando deseamos que los thrackles usados en la descomposición de la gráfica sean geométricos, entonces podemos definir el anti-thickness geométrico como sigue.

Definición 4. *Anti-thickness geométrico de una gráfica.* El anti-thickness geométrico de una gráfica G es el entero k más pequeño tal que existe un dibujo G de G para el cual hay una partición de $E(\mathsf{G})$, de tamaño k, en la que cada elemento de la partición induce un thrackle.

Nótese que en la partición que realiza el anti-thickness de la gráfica, cada elemento de la partición tiene uno o más dibujos, mientras que en la partición que realiza el anti-thickness geométrico de la gráfica, cada elemento es una subgráfica geométrica del dibujo original.

En la figura 2.15 damos un ejemplo del anti-thickness geométrico de K_5.

También es posible definir el anti-thickness de una gráfica geométrica como sigue:

Definición 5. *Anti-thickness de una gráfica geométrica* El anti-thickness $At_g(\mathsf{G})$ de una gráfica geométrica G es el entero k más pequeño tal que existe una partición de $E(\mathsf{G})$, de tamaño k, en la que cada elemento de la partición es un thrackle.

Dar una descomposición de una gráfica en thrackles es equivalente a encontrar conjuntos independientes de aristas que comparten ciertas propiedades. A su vez, encontrar los conjuntos independientes de una gráfica está relacionado a encontrar el *número cromático* de una gráfica (abstracta). Discutimnos esto en la siguiete sección.

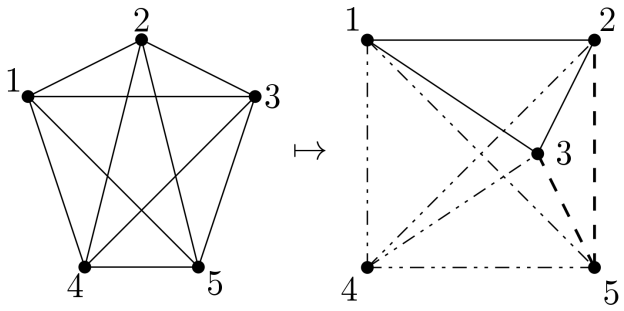


Figura 2.15: En esta figura podemos observar una descomposición de K_5 en 3 thrackles geométricos. Esta descomposición se muestra en la figura del lado derecho, cada thrackle está dibujado con diferentes patrones de línea. El anti-thickness geométrico de K_5 es exactamente tres, esto es demostrado en la sección de resultados.

2.3. Número cromático

Las siguientes definiciones fueron tomadas de Chartrand & Zhang (2008).

Una *coloración propia* de los vértices de una gráfica G es la asignación de colores a los vértices de G tal que cada vértice tiene un solo color asignado y dos vértices adyacentes tienen diferentes colores. Un color puede ser un color como rojo, verde, amarillo, etc. cuando el número de colores a usar es pequeño, de otra forma se usan enteros $1, 2, \ldots, k$ para algún entero positivo k para representar los colores. Si la coloración propia usa k colores diferentes decimos que tenemos una *k-coloración* de la gráfica G. Dada una k-coloración de una gráfica G, si V_i es el conjunto de vértices de G que tienen el color i asignado, llamamos a V_i una *clase cromática* de G. El conjunto $\{V_1, V_2, \ldots, V_k\}$ genera una partición en conjuntos independientes de los vértices de G.

Una gráfica G es *k-colorable* si existe una coloración propia de G de tamaño k. El entero positivo k más pequeño para el cual G es k-colorable recibe el nombre de *número cromático* de G. Lo denotamos como $\chi(G)$.

La figura 2.16 muestra un ejemplo de una coloración propia de una gráfica G. En este ejemplo ilustramos cada clase cromática dibujando los vértices con diferentes colores representados por una cruz, un círculo y un cuadrado. Si tenemos una descomposición en k thrackles de una gráfica geométrica y asignamos uno de k colores a cada thrackle de la descomposición de tal manera que no existan dos thrackles del mismo color y si además minimizamos el valor de k entonces k es el anti-thickness de la gráfica geométrica

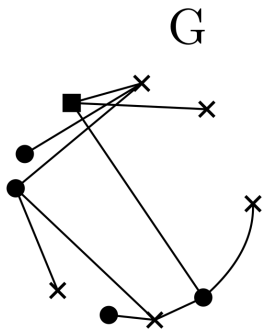


Figura 2.16: Una coloración propia de una gráfica G. Esta coloración es de tamaño 3, por lo tanto decimos que es una 3-coloración de G. Para esta gráfica en particular no existe una coloración más pequeña, por lo que su número cromático es 3. Nótese que los conjuntos independientes son formados por vértices que tienen el mismo color asignado y no son adyacentes entre sí.

dada. Esta idea ha sido utilizada para encontrar el anti-thickness de una gráfica geométrica cuyos vértices están en posición covexa. Detallamos este concepto en la siguiente sección.

En este capítulo explicamos los conceptos necesarios para entender el trabajo realizado en esta tesis. Empezamos hablando de gráficas abstractas y después de su representación en el plano usando aristas que son curvas y usando aristas que son segmentos de recta a las cuales llamamos gráficas geométricas. Continuamos definiendo un tipo especial de gráfica geométrica en la cual cada par de aristas se intersecta, este tipo de gráfica geométrica recibe el nombre de thrackle. Luego explicamos el tipo de orden como herramienta para discretizar el número de dibujos posibles de una gráfica en el plano. Después hablamos del anti-thickness geométrico de una gráfica como el mínimo número de thrackles que existen en una descomposición para todos los dibujos de una gráfica. Finalmente mencionamos el número cromático como el mínimo número de conjuntos independientes que existen en una partición de los vértices de una gráfica. Este último punto es de importancia para el siguiente capítulo pues explicamos cómo el problema de encontrar el anti-thickness geométrico de una gráfica puede ser visto como un problema de encontrar el número cromático. Además hablaremos de cuáles son los aportes más recientes acerca del problema del anti-thickness geométrico.

Capítulo 3

Estado del arte

Comenzamos este capítulo hablando del thickness de una gráfica, concepto que cronológicamente fue definido antes que el anti-thickness, y discutimos su relación con éste. Luego, mencionamos algunos resultados acerca del thickness. Después, explicaremos de qué manera el número cromático de una gráfica puede otorgar una descomposición de una gráfica geométrica. Finalmente discutimos los resultados actuales para el anti-thickness.

El thickness de una gráfica se define como sigue:

Definición 6. [Dillencourt *et al.* (2004)] *Thickness.* Sea G una gráfica, el thickness $\theta(G)$ de G es el mínimo entero k para el cual existe una partición de $E(G)$, de tamaño k, en la que cada elemento de la partición induce una gráfica planar.

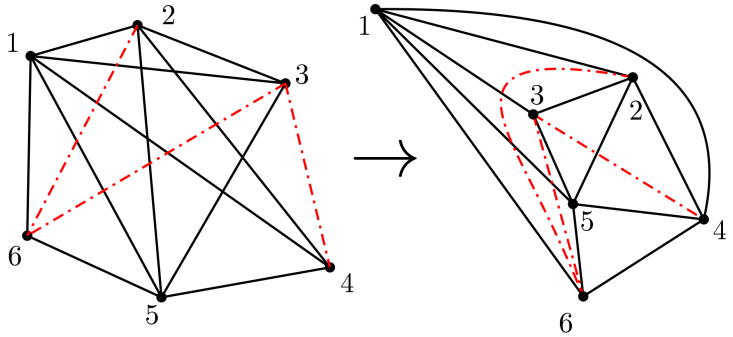
Este concepto también puede extenderse a gráficas geométricas, en este caso estaremos hablando del thickess de una gráfica geométrica:

Definición 7. *Thickness de una gráfica geométrica.* Sea G una gráfica geométrica, el thickness $th(\mathsf{G})$ de G es el mínimo entero k para el cual existe una partición de $E(\mathsf{G})$, de tamaño k, donde cada elemento de la partición induce una gráfica (geométrica) plana.

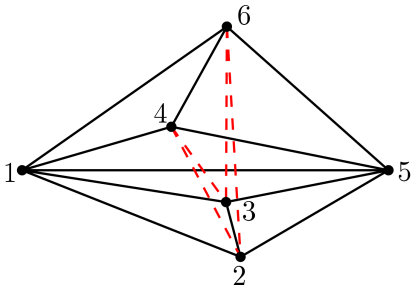
Finalmente podemos definir el thickness geométrico de una gráfica G.

Definición 8. [Dillencourt *et al.* (2004)] *Thickness geométrico.* Sea G una gráfica, el thickness geométrico $\bar{\theta}(G)$ de G es :

$$\bar{\theta}(G) = \text{mín}\{th(\mathsf{G}) : \mathsf{G} \text{ es una gráfica geométrica de G}\}.$$



(a) Una descomposición de tamaño 2 de K_6.



(b) Un dibujo geométrico de K_6 y su descomposición en dos gráficas geométricas planas.

Figura 3.1: La figura (a) muestra que el thickness de K_6 es menor o igual a dos. La figura (b) muestra que el thickness geométrico de K_6 es menor o igual a dos. Sin embargo, en general, estos parámetros no coinciden.

En la figura 3.1 ilustramos un ejemplo del thickness de la gráfica completa de 6 vértices.

Dillencourt *et al.* (2004) demuestran que el thickness geométrico está acotado por:

$$\left\lceil \frac{n}{5.646} + 0.342 \right\rceil \leq \bar{\theta}(G) \leq \left\lceil \frac{n}{4} \right\rceil .$$

En el mismo artículo encuentran el valor exacto del thicknes geométrico de cada gráfica completa con n vértices, para $n \leq 12$, así como para K_{15} y K_{16}. También estudian el thickness geométrico para gráficas completas

bipartitas y demuestran la siguiente cota:

$$\left\lceil \frac{ab}{2a+2b-4} \right\rceil \leq \theta(K_{a,b}) \leq \bar{\theta}(K_{a,b}) \leq \left\lceil \frac{min(a,b)}{2} \right\rceil .$$

Para explicar la relación entre el thickness y el anti-thickness es necesario hablar de coloraciones de vértices de gráficas de adyacencia. Recordemos que dadas dos aristas de una gráfica geométrica G, decimos que estas se *intersectan* si comparten un vértice (son adyacentes) o si se cruzan, y que son *disjuntas* si no se intersectan. La *gráfica de adyacencia* de una gráfica geométrica G tiene como conjunto de vértices a todas las aristas de G y sus aristas se definen a partir del tipo de adyacencia que se considere; se puede definir cuatro diferentes gráficas de adyacencia, estas se listan a continuación y se ilustran en la figura 3.2:

1. La gráfica de adyacencia en la que existe una arista entre dos vértices si las aristas correspondientes en G se cruzan.

2. La gráfica de adyacencia en la que existe una arista entre dos vértices si las aristas correspondientes en G comparten un vértice o son disjuntas.

3. La gráfica de adyacencia en la que existe una arista entre dos vértices si las aristas correspondientes en G se intersectan.

4. La gráfica de adyacencia en la que existe una arista entre dos vértices si las aristas correspondientes en G son disjuntas.

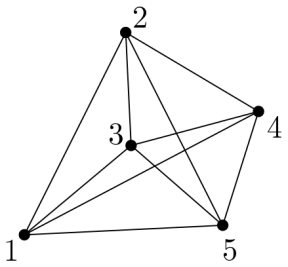
Ahora definimos una gráfica a la que llamamos *gráfica de cruce*.

Definición 9. *Gráfica de cruce.* Sea S un conjunto de n puntos en posición general en el plano y sea $K_n(S)$ la gráfica completa asociada a S. La gráfica de cruce $E_{pp}(S)$ de S es la gráfica que tiene un vértice por cada arista de $K_n(S)$ y una arista entre dos vértices de $E_{pp}(S)$ si sus aristas correspondientes se cruzan en $K_n(S)$.

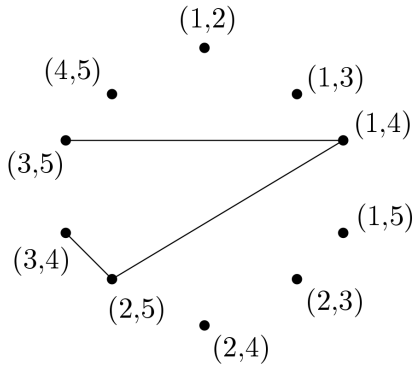
La gráfica $E_{pp}(S)$ corresponde a la gráfica de adyacencia 1.

En la figura 3.2b aparece un ejemplo de la gráfica de cruce $E_{pp}(S)$. Podemos notar que dado que en el dibujo de K_5 hay 3 cruces, en la gráfica $E_{pp}(S)$ hay 3 aristas.

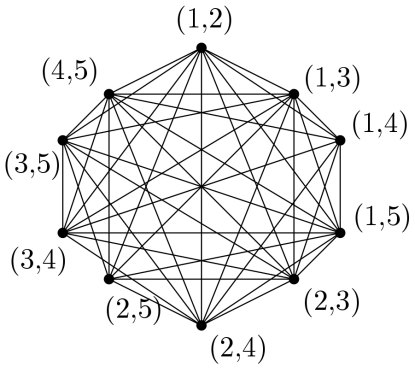
Los conjuntos independientes de $E_{pp}(S)$ corresponden a conjuntos de aristas de $K_n(S)$ que inducen gráficas planas. Luego, una coloración propia de los



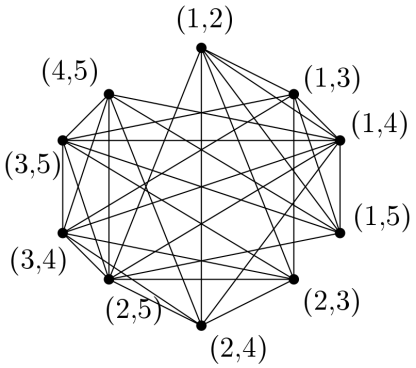
(a) Un dibujo geométrico de K_5 cuyo conjunto de vértices es $S = \{1, 2, 3, 4, 5\}$.



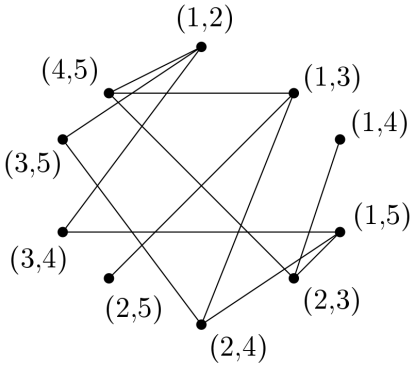
(b) La gráfica de adyacencia 1.



(c) La gráfica de adyacencia 2.



(d) La gráfica de adyacencia 3.



(e) La gráfica de adyacencia 4.

Figura 3.2: Una dibujo de K_5 y sus respectivas gráficas de adyacencia.

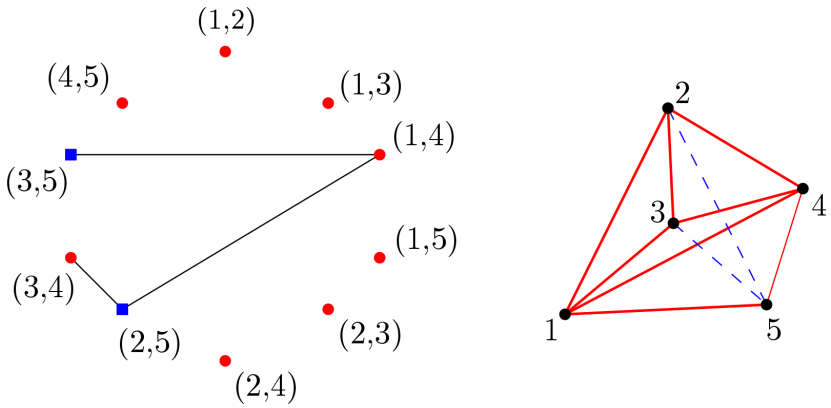


Figura 3.3: De izquierda a derecha: Una coloración propia de los vértices de $E_{pp}(S)$ con dos clases cromáticas indicadas como círculos y cuadros. Una descomposición de un dibujo de K_5 en dos gráficas planas cuyas aristas corresponden a cada una de las clases cromáticas de la coloración de $E_{pp}(S)$.

vértices de $E_{pp}(S)$ corresponde a una descomposición de $K_n(S)$ en gráficas planas. Por lo tanto encontrar el número cromático de la gráfica $E_{pp}(S)$ es equivalente a encontrar el thickness geométrico de $K_n(S)$. La figura 3.3 ilustra esta relación. El thickness geométrico de la gráfica completa de n vértices K_n se puede definir en estos términos como sigue:

Definición 10. *Thickness geométrico de una gráfica.*

$$\bar{\theta}(n) = \text{mín}\{\chi(E_{pp}(S)) : S \subset \mathbb{R}^2 \text{ está en posición general }, |S| = n\}.$$

De la misma manera que definimos la gráfica de cruce podemos definir otras gráficas de adyacencia. Para esto basta con considerar alguno de los otros tipos de adyacencia. En este sentido, en el trabajo de Araujo *et al.* (2005) se definen las gráficas que se listan en seguida, cada una corresponde a una de las cuatro gráficas de adyacencia mencionadas anteriormente. Sea S un conjunto de n puntos en posición general y $K_n(S)$ la gráfica completa asociada a S:

- $W(S)$: Es la gráfica correspondiente a la gráfica de adyacencia 2.
- $I(S)$: Es la gráfica correspondiente a la gráfica de adyacencia 3.
- $D(S)$ Es la gráfica correspondiente a la gráfica de adyacencia 4.

Gráfica	Conjuntos independientes en $K_n(S)$
$W(S)$	*Crossing families*
$I(S)$	*Emparejamientos planos*
$D(S)$	*Thrackles*
$E_{pp}(S)$	*Gráficas planares*

Tabla 3.1: Esta tabla muestra qué representan los conjuntos independientes en $K_n(S)$ para cada una de las gráficas definidas en Araujo *et al.* (2005) y para $E_pp(S)$. Una *crossing familie* es una colección de segmentos que se cruzan a pares Lara & Rubio-Montiel (2019).

A partir de aquí, nos referimos a las gráficas de adyacencia 1,2,3 y 4 como $E_{pp}(S), W(S), I(S)$ y $D(S)$, respectivamente.

Es fácil darse cuenta que la condición de existencia de aristas de $W(S)$ es complementaria a la condición en $E_{pp}(S)$ y viceversa. De la misma forma, las condiciones de $I(S)$ y $D(S)$ son complementarias entre sí. En la tabla 3.1 mostramos esta relación y las estructuras geométricas que inducen, en $K_n(S)$, los conjuntos independientes para cada gráfica de adyacencia.

Open problem garden es un sitio web en el que investigadores de diferentes áreas de las matemáticas, como algebra, combinatoria, teoría de números o teoría de gráficas, publican problemas abiertos para que la comunidad pueda leerlos libremente. En este sitio web Hurtado (2009) presenta un problema en el que se requiere asignar un color a cada arista de una gráfica geométrica completa usando, de manera implicita, cada una de las cuatro gráficas de adyacencia mencionadas antes.

En el artículo de Araujo *et al.* (2005) buscan el número cromático de cada una de las gráficas de adyacencia mencionadas anteriormente, los autores estudian los siguientes parámetros:

$$w(n) = \text{máx}\{\chi(W(S)) : S \subset \mathbb{R}^2 \text{ está en posición general}, |S| = n\}.$$
$$i(n) = \text{máx}\{\chi(I(S)) : S \subset \mathbb{R}^2 \text{ está en posición general}, |S| = n\}.$$
$$d(n) = \text{máx}\{\chi(D(S)) : S \subset \mathbb{R}^2 \text{ está en posición general}, |S| = n\}.$$

Es usual estudiar el dibujo en posición convexa de una gráfica completa ya que se pueden explotar ciertas propiedades geométricas cuando se trabaja con un conjunto de puntos en posición convexa. Esta configuración es representada

por un solo tipo de orden para cada n. Para el caso en el que S está en posición convexa se denotan a estos valores como $w_c(n)$, $i_c(n)$ y $d_c(n)$. Es importante notar que, por consecuencia de la posición convexa, estos valores están definidos para un único tipo de orden. Nótese que $d_c(n) = \chi(D(S))$ para el caso en el que S está en posición convexa.

Los autores demuestran las siguientes cotas:

- $w_c(n) = \Theta(n \log n)$.
- $c_1 n \log n \leq w(n) \leq c_2 n^2 \frac{\log \log n}{\log n}$, para $c_1, c_2 > 0$.
- $i_c(n) = n$.
- $n \leq i(n) \leq C n^{3/2}$ para $C > 0$.
- $2\lfloor \frac{n+1}{3} \rfloor - 1 \leq d_c(n) \leq \text{mín}\left(n-2, n - \frac{\lfloor \log n \rfloor}{2}\right)$.
- $5\lfloor \frac{n}{7} \rfloor \leq d(n) \leq \text{mín}\left(n-2, n + \frac{1}{2} - \frac{\lfloor \log \log n \rfloor}{n}\right)$.

Ahora bien, como los conjuntos independientes de la gráfica $D(S)$ son thrackles, es posible definir el anti-thickness geométrico usando la gráfica $D(S)$ como sigue:

Definición 11. *Anti-thickness geométrico de una gráfica.*

$$At_g(n) = \text{mín}\{\chi(D(S)) : S \subset \mathbb{R}^2 \text{ está en posición general}, |S| = n\}.$$

De manera análoga si los puntos de S están en posición convexa definimos el anti-thickness convexo:

Definición 12. *Anti-thickness convexo de una gráfica.*

$$At_c(n) = \chi(D(S)) : S \subset \mathbb{R}^2 \text{ está en posición convexa}, |S| = n.$$

Como ya mencionamos antes, el anti-thickness convexo de la gráfica completa de n vértices es equivalente a $d_c(n)$. En otras palabras, sea S un conjunto de n puntos en posición convexa:

$$\begin{aligned} At_c(n) &= d_c(n) \\ \Rightarrow \chi(D(S)) &= \text{máx}\{\chi(D(S))\} \end{aligned}$$

Pero como solamente existe un solo dibujo cuando S está en posición convexa, entonces:

$$\text{máx}\{\chi(D(S))\} = \chi(D(S)).$$

A pesar de que en el trabajo de Araujo *et al.* (2005) se busca el número cromático de la gráfica $D(S)$ y con ello, como mencionamos antes, se busca el anti-thickness de una gráfica geométrica asociada a S, es importante notar que la definición $d(n)$ no es equivalente a la de anti-thickness. Dada una gráfica G: en $d(n)$ se busca el máximo de todos los números cromáticos para todas las gráficas geométricas de G mientras que en el anti-thickness se busca el mínimo de todos los números cromáticos para todas las gráficas geométricas de G.

Los autores de Dujmovic & Wood (2017) presentan resultados respecto a el anti-thickness para familias específicas de gráficas como árboles, gráficas outerplanar, y algunos dibujos como 2-tracks, books, k-queues, entre otros.

En su trabajo definen en anti-thickness como sigue:

Definición 13. *Anti-thickness de una gráfica.* Sea G una gráfica el anti-thickness $At(G)$ de G es el mínimo entero k para el cual existe una partición de $E(G)$ de tamaño k en la que cada elemento de la partición induce una gráfica thrackleable.

Además dan una relación entre el thickness y anti-thickness de cualquier gráfica. Concretamente los autores prueban la siguiente cota para toda gráfica con anti-thickness k y thickness t:

$$k \leq t \leq \left\lceil \frac{3k}{2} \right\rceil.$$

Acerca del anti-thickness de gráficas completas en el mismo artículo prueban que

$$\frac{n}{3} \leq At(K_n) \leq \left\lceil \frac{n-1}{2} \right\rceil.$$

Ellos conjeturan que el anti-thickness de una gráfica completa es exactamente $\left\lceil \frac{n-1}{2} \right\rceil$.

Respecto al anti-thickness geométrico, en los siguientes párrafos describimos cómo los autores de Fabila-Monroy *et al.* (2018a) encuentran el anti-thickness geométrico exacto para gráficas cuyo conjunto de vértices está en posición convexa.

Los autores prueban que cuando los vértices de la gráfica geométrica están en posición convexa dos thrackles máximos siempre comparten al menos una arista. Por lo anterior la unión de k thrackles máximos tiene a lo sumo $kn-\binom{k}{2}$ aristas. Entonces, como una gráfica completa de n vértices tiene $\binom{n}{2}$ aristas la resolución de la siguiente desigualdad otorga el resultado de la cota inferior para el anti-thickness convexo.

$$\binom{n}{2} \leq kn - \binom{k}{2}.$$

Esta cota es:

$$At_c(K_n) \geq n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor.$$

La cota superior se obtiene dando una coloración propia de los vértices de la gráfica $D(S)$. Puesto que como explicamos anteriormente este problema es equivalente a encontrar el número cromático de la gráfica. En el articulo logran la coloración trazando caminos en una estructura conocida como poliominó Fabila-Monroy *et al.* (2018b) en la que los vértices de $D(S)$ son las filas y las columnas de dicha estructura. En dicho trabajo cada camino representa un conjunto independiente de vértices en $D(S)$ y por lo tanto respresenta un thrackle en K_n. Los autores concluyen dando el número máximo de caminos posibles en el poliominó, dicho número coincide con el de la cota inferior y luego se tiene que el anti-thickness convexo tiene exactamente el siguiente valor:

$$At_c(n) = n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor.$$

A continuación hablamos del trabajo de Fabila-Monroy *et al.* (2017) en donde encuentran el anti-thickness geométrico de una familia de conjuntos conocidos como doble cadena convexa. Una doble cadena convexa es la configuración de puntos conformada por una *k-cup* y una *l-cap*. Una *k-cup* es una cadena de k puntos en posición convexa donde la parte superior de su *cierre convexo*[1] está delimitado por un solo segmento. Una *l-cap* es una cadena de l puntos en posición convexa donde la parte inferior de su cierre convexo está delimitado por un solo segmento. La doble cadena convexa $C_{k,l}$ tiene las siguientes propiedades:

[1]El cierre convexo de un conjunto de puntos es el polígono con área más pequeña que contenga a todos los puntos del conjunto.

- Para $l \geq k$ la doble cadena convexa es la unión una *k-cup* U y una *l-cap* L.
- Cada punto de L está por debajo de cada segmento de recta definido por dos puntos de U.
- Cada punto de U está por arriba de cada segmento de recta definido por dos puntos de L.

La figura 3.4 muestra un ejemplo de una doble cadena convexa.

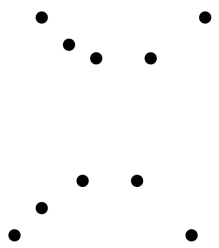


Figura 3.4: Una doble cadena convexa con 5 vértices en su *k-cup* y 5 vértices en su *l-cap*.

El resultado al que se llega en ese trabajo es el siguiente: el anti-thickness geométrico de la gráfica completa inducida por una doble cadena convexa con k puntos en la cadena convexa superior y l puntos en la cadea convexa inferior, $K_{l,k}$, es:

$$At_g(K_{l,k}) = k + l - \left\lfloor \sqrt{2l + \frac{1}{4}} - \frac{1}{2} \right\rfloor$$

Dujmovic & Wood (2017) mencionan que encontrar el anti-thickness geométrico de gráficas completas en posición general es un problema abierto. En esta tesis abordamos este problema para $n \leq 10$.

Hasta ahora las mejores cotas conocidas para el anti-thickness geométrico son :

$$\left\lfloor \frac{n-1}{2} \right\rfloor \leq At_g(K_n) \leq n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor . \tag{3.1}$$

La cota superior se sigue del caso convexo y la cota inferior se sigue del hecho de que una gráfica con n vértices con anti-thickness geométrico k tiene a lo

sumo kn aristas. Discutimos estas ideas con más detalle en la sección 4.1 del capítulo 4.

Los trabajos mencionados en este capítulo muestran cómo el problema original del thickness está relacionado con problema del anti-thickness. Mostramos de qué manera un problema de descomposición de gráficas geométricas equivale a un problema de coloración de gráficas. Los artículos más recientes tratan al anti-thickness como un problema de coloración. En este trabajo no construimos la gráfica $D(S)$ y no coloreamos ninguna gráfica, uestro enfoque es geométrico y computacional.

Es importante notar que en el trabajo de Fabila-Monroy *et al.* (2018a) las descomposiciones de la gráfica completa están conformadas por thrackles máximos. Esta es una de las ideas que usamos en este trabajo para intentar ajustar las cotas del anti-thickness en posición general.

Capítulo 4

Resultados

4.1. Cota inferior del anti-thickness geométrico de K_n

El principal resultado de este trabajo es que encontramos el valor exacto del anti-thickness geométrico para la gráfica completa con hasta diez vértices en posición general. Obtuvimos dicho valor mejorando la cota inferior y usando la cota superior actual, las cuales siguen a continuación:

$$\left\lfloor \frac{n-1}{2} \right\rfloor \leq At_g(K_n) \leq n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor. \tag{4.1}$$

De manera general, si se desea encontrar una cota superior para el anti-thickness geométrico de K_n es necesario encontrar una descomposición en thrackles, para cualquier dibujo de K_n. La cota superior trivial del anti-thickness geométrico de K_n es $\binom{n}{2}$ ya que cada arista de K_n es un thrackle. Si se desea encontrar una cota inferior es necesario demostrar, para todo dibujo de K_n, cuántos thrackles son necesarios para dar una descomposición en thrackles.

Empezamos explicando la cota inferior conocida para el anti-thickness geométrico y después diremos cómo la mejoramos para $n \leq 10$.

Un conjunto de n puntos en el plano induce una gráfica completa con n vértices a la que denotamos como K_n. Como es completa $|E(K_n)| = \binom{n}{2}$. Cuando se quiere encontrar una descomposición de tamaño mínimo en thrackles una idea que resulta intuitiva es buscar que los thrackles tengan el

mayor número posible de aristas. El siguiente teorema es útil para nosotros ya que establece el número máximo de aristas que puede tener un thrackle geométrico.

Teorema 4.1.1. (Pach (2013b)) Toda gráfica geométrica de n vértices en la que no existen dos aristas disjuntas tiene a lo más n aristas. Esto se cumple para toda $n > 2$.

Omitimos la demostración del teorema anterior ya que las ideas de la demostración no son retomadas en este trabajo.

Como mencionamos en el capítulo de antecedentes, un thrackle de n vértices con exactamente n aristas es un thrackle máximo. Por definición, toda descomposición de la gráfica completa cubre sus aristas. Si suponemos que existen k thrackles máximos en la descomposición, entonces la siguiente desigualdad expresa el número de thrackles máximos necesarios para cubrir las $\binom{n}{2}$ aristas de la gráfica completa:

$$kn \geq \binom{n}{2}.$$

Como los thrackles de la descomposición son geométricos, si buscamos la k más pequeña para la cual se cumple la desigualdad anterior entonces k sería el anti-thickness geométrico de la gráfica completa. Resolviendo para k tenemos que al menos $k \geq \left\lceil \frac{n-1}{2} \right\rceil$ thrackles máximos son necesarios para dar una descomposición de K_n. En otras palabras

$$At_g(K_n) \geq \left\lceil \frac{n-1}{2} \right\rceil. \tag{4.2}$$

La tabla 4.1 ilustra el valor de la cota inferior del anti-thickness geométrico dada por la desigualdad 4.2 para $n \leq 10$. Esta cota es la más inmediata ya que usa el hecho de que cada thrackle máximo tiene a lo sumo tantas aristas como vértices. También es la cota actual para el anti-thickness geométrico (Dujmovic & Wood (2017)).

En el trabajo de Fabila-Monroy *et al.* (2018a) encuentran que, dados dos thrackles máximos en posición convexa, estos comparten una arista, y que esto se cumple para cada par de thrackles de la descomposición. En este trabajo verificamos que este resultado también es válido para conjuntos de hasta diez puntos en posición general (no convexa). Usando los tipos de orden de los conjuntos con hasta diez puntos inducimos la gráfica completa, luego

n	$\lceil \frac{n-1}{2} \rceil$
3	1
4	2
5	2
6	3
7	3
8	4
9	4
10	5

Tabla 4.1: Valor de la cota inferior del anti-thickness geométrico usando la cota trivial.

buscamos, para cada uno de los tipos de orden, todos los thrackles máximos y finalmente comparamos dichos thrackles a pares y encontramos que:

- Para todo tipo de orden con al menos dos thrackles máximos, cada par de thrackles máximos tienen intersección no vacía en aristas.
- Existen tipos de orden con solo un thrackle máximo.
- Existen tipos de orden en los que no hay thrackles máximos.

En la tabla 4.2 mostramos cuántos tipos de orden tienen thrackles máximos, cuántos tienen solamente un thrackle máximo y cuántos no tienen ningún thrackle máximo. La figura 4.1 muestra un ejemplo de un tipo de orden, para $n = 9$, en el cual no existe ningún thrackle máximo. Una pregunta interesante es ¿cómo caracterizamos a los tipos de orden cuya gráfica completa no tiene thrackles máximos?

Los algoritmos que diseñamos para dar los resultados anteriores son descritos en la sección 4.6 para no romper con el flujo de esta sección. Para buscar los thrackles máximos usamos el algoritmo 1, para comparar los thrackles a pares utilizamos el algoritmo 2. La lista de tipos de orden con un solo thrackle máximo puede ser descargada de la siguiente liga: `http://computacion.cs.cinvestav.mx/~dmerinos/site/archivos_tesis/withone_max.tar.gz`. La lista de tipos de orden para los cuales no existe un thrackle máximo está en la siguiente liga: `http://computacion.cs.cinvestav.mx/~dmerinos/site/archivos_tesis/without_max.tar.gz`.

Usando estos resultados podemos derivar el siguiente lema.

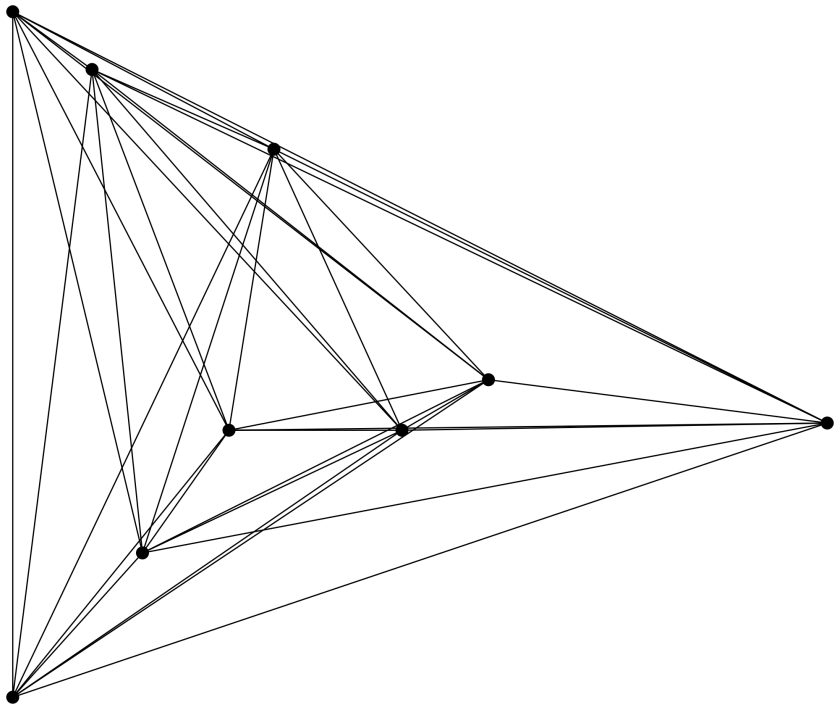


Figura 4.1: Este conjunto de puntos corresponde al tipo de orden 136764 de $n = 9$. La gráfica completa inducida por ellos no contiene ningún thrackle máximo.

n	Tipos de orden(T.O.)	T.O. con solo un thrackle máximo	T.O. sin thrackles máximos
3	1	1(33 %)	0(0 %)
4	2	0(0 %)	1(50 %)
5	3	0(0 %)	2(66 %)
6	16	1(6.25 %)	6(37.50 %)
7	135	7(5.18 %)	50(37.03 %)
8	3315	208(6 %)	1175(35.44 %)
9	158817	10547(6.64 %)	53758(33.84 %)
10	14309547	962517(6.72 %)	4654339(32.52 %)

Tabla 4.2: Mostramos, para cada $3 \leq n \leq 10$, la relación de los tipos de orden con solamente un thrackle máximo y los tipos de orden sin thrackles máximos.

Lema 4.1.2. Sea S un conjunto de puntos en posición general y sean T_1 y T_2 thrakles máximos en $K_n(S)$ con $|V(T_1)|\leq 10$ y $|V(T_2)|\leq 10$. T_1 y T_2 tienen al menos una arista en común.

Demostración. Para cada tipo de orden con hasta diez puntos, generamos todos los thrackles máximos inducidos. Usando este conjunto verificamos que para cada pareja la intersección en aristas es no vacía □

El lema anterior implica que para $n \leq 10$ no es posible encontrar una descomposición en thrackles máximos de tamaño $\left\lceil \frac{n-1}{2} \right\rceil$, ya que, en dicha descomposición, los thrackles máximos no son disjuntos a pares pero esto infringiría las condiciones de una descomposición. En otras palabras una descomposición por thrackles solo podría contener un thrackle máximo. Sin embargo, es posible encontrar, a partir de una colección de thrackles máximos, una colección de thrackles que son disjuntos. Describimos este proceso en el siguiente lema:

Lema 4.1.3. Sea $\mathcal{C} = \{T_1, T_2, \ldots, T_m\}$ una colección de thrackles máximos de K_n con $|E(T_i) \cap E(T_j)| = 1$ y $i, j \in \{1, 2, \ldots, m\}$. Existe una colección de thrackles $\mathcal{D}$ de K_n inducida por $\mathcal{C}$ en la que cada par de thrackles tienen intersección vacía y que cubre el siguiente número de aristas:

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

Demostración. Sea T_i', con $1 \leq i \leq m$ un thrackle inducido por el siguiente conjunto de aristas:

$$E(T_i) - \bigcup_{k=1}^{i-1} E(T_i) \cap E(T_k).$$

Es decir, T_i' es el thrackle que tiene todas las aristas de T_i a excepción de aquellas que T_i comparte con el resto de los thrackles en la colección.

Sea $\mathcal{D} = \{T_1', T_2', \ldots, T_m'\}$. Por construcción, la intersección de cualesquiera dos thrackles de $\mathcal{D}$ es vacía.

Nótese que si una arista e aparece en dos thrackles de $\mathcal{C}$, entonces en $\mathcal{D}$, e aparecerá únicamente en el thrackle con menor etiqueta. De hecho T_1' tiene n aristas, T_2' tiene $n-1$ aristas, T_3' tiene $n-2$ aristas y, en general, T_i' tiene $n-i+1$ aristas. Como $\mathcal{D}$ tiene m thrackles el número de aristas cubiertas por $\mathcal{D}$ es

$$n + (n-1) + (n-2) + \cdots + (n-m+2) + (n-m+1).$$

n	$\lceil \frac{n-1}{2} \rceil$	$\sum_{i=\left(n-\lceil \frac{n-1}{2} \rceil\right)+1}^{n} i$	$\binom{n}{2}$
3	1	3	3
4	2	7	6
5	2	9	10
6	3	15	15
7	3	18	21
8	4	26	28
9	4	30	36
10	5	40	45

Tabla 4.3: Mostramos cuántas aristas son cubiertas con una colección de $\lceil \frac{n-1}{2} \rceil$ thrackles máximos disjuntos en aristas. Se rellenan los casos en los que la colección no cubre todas las aristas.

Podemos escribir esta suma como :

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

□

Es importante notar que este es el máximo número de aristas que una colección con m thrackles disjuntos puede cubrir.

Con los lemas anteriores es posible probar que la cota inferior del anti-thickness geométrico de K_n, mostrada en la ecuación, 4.1 no es justa para toda n. El siguiente teorema establece esta afirmación.

Teorema 4.1.4. Sea $\mathcal{D} = \{T'_1, T'_2, \ldots, T'_{\lceil \frac{n-1}{2} \rceil}\}$ una colección de thrackles disjuntos en aristas inducida por una colección de $\lceil \frac{n-1}{2} \rceil$ thrackles máximos. $\lceil \frac{n-1}{2} \rceil$ thrackles máximos son suficientes para inducir una descomposición de K_n cuando $n = 3, 4, 6$ y no son suficientes cuando $n = 5, 7, 8, 9, 10$.

Demostración. Se sigue del lema 4.1.3. El número de aristas cubiertas por $\mathcal{D}$ para cada $3 \leq n \leq 10$ se muestran en la tabla 4.3. Se marcan de gris los casos en los que no es posible cubrir todas las aristas de K_n. □

Como consecuencia del lema 4.1.3 y el teorema 4.1.4 podemos dar una cota inferior justa para el anti-thickness geométrico de K_n con $3 \leq n \leq 10$.

Teorema 4.1.5. Sea S un conjunto de n puntos en el plano en posición general con $3 \leq n \leq 10$, y sea $K_n(S)$ la gráfica completa inducida por S. El anti-thickness de $K_n(S)$:

$$At_g(K_n) \geq n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor. \tag{4.3}$$

Demostración. Se sigue del teorema 4.1.3 que el número de aristas cubiertas por una colección $\mathcal{D} = \{T_1, T_2, \ldots, T_m\}$ de thrackles, con $|E(T_i) \cap E(T_j)| = 1$, es

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

Para saber cuántos thrackles como minimo son necesarios en la colección para cubrir las $\binom{n}{2}$ aristas de K_n, es necesario resolver la siguiente desigualdad otorga el resultado.

$$-\frac{1}{2}m(m - 2n - 1) \geq \binom{n}{2}. \tag{4.4}$$

De aquí se tiene que m está en el siguiente intervalo:

$$\frac{1}{2}\left(2n - \sqrt{8n+1} + 1\right) \leq m \leq \frac{1}{2}\left(2n + \sqrt{8n+1} + 1\right).$$

Como nosotros estamos interesados en la m más pequeña para la cual se cumple la desigualdad 4.4, basta con tomar el término de la izquierda para obtener el resultado deseado. □

Presentamos los valores dados por la cota inferior del teorema 4.1.5 en la tabla 4.4.

Como consecuencia del teorema 4.1.5 es posible obtener un valor exacto para el anti-thickness geométrico de K_n en posición general.

Teorema 4.1.6. Sea $K_n(S)$ la gráfica completa inducida por un conjunto S de n puntos, con $3 \leq n \leq 10$, en posición general. El anti-thickness geométrico de K_n es:

$$At_g(K_n) = n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor.$$

n	$k = n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$	$\sum_{i=(n-k)+1}^{n} i$	$\binom{n}{2}$
3	1	3	3
4	2	7	6
5	3	12	10
6	3	15	15
7	4	22	21
8	5	30	28
9	6	39	36
10	6	45	45

Tabla 4.4: En la tabla se muestra que la cota inferior del anti-thickness geométrico es justa para $3 \leq n \leq 10$. La segunda columna muestra el tamaño de la colección de thrackles, cuyo tamaño está dado por la cota inferior del anti-thickness geométrico del teorema 4.1.5 para cada n. La tercerca columna muestra cuántas aristas cubre dicha colección. La cuarta columna muestra el número de aristas de K_n.

Demostración. Se sigue del resultado del teorema 4.1.5 y la cota superior del anti-thickness geométrico de K_n, $n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$, cuando K_n es inducida por un conjunto de puntos en posición convexa. □

Si desearamos dar un resultado similar para $n > 10$, bastaría con generalizar el lema 4.1.2. Esto es, demostrar que cada par de thrackles máximos se intersecta en al menos una arista y que esto sucede para toda $n > 10$. Nosotros conjeturamos que este lema también se cumple para $n > 10$.

Debido a que la cota superior del anti-thickness geométrico está dada por el dibujo en posición convexa, en este trabajo decidimos verificar si existen dibujos, diferentes al convexo, para los cuales se alcanza la cota inferior dada por el teorema 4.1.5.

4.2. Descomposiciones inducidas por thrackles máximos

La cota superior del anti-thickness geométrico está dada por el conjunto de puntos en posición convexa. Sea $k = n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$, en esta tesis

verificamos que existen tipos de orden, que no corresponden a la posición convexa, y que inducen una gráfica completa cuyo anti-thickness geométrico es a lo sumo k.

Para comprobar la existencia de tipos de orden con las características descritas anteriormente elegimos aquellos que tienen al menos k thrackles máximos. Luego, para cada uno, analizamos todas sus colecciones de thrackles máximos posibles de tamaño k. Finalmente, para cada colección, evaluamos si esta cubre o no a $E(K_n)$. Cuando una colección de thrackles máximos cubre al conjunto de aristas de la gráfica completa, es posible inducir una descomposición de K_n en thrackles (no necesariamente máximos) asignando cada arista de K_n a un único thrackle que la cubra. Detallamos el algoritmo para buscar colecciones de thrackles máximos que induzcan una descomposición en la sección 4.6.3. La tabla 4.5 muestra los resultados obtenidos de este análisis; observamos que para $n \in \{8, 9, 10\}$ existen tipos de orden que tienen k thrackles máximos que pueden inducir una descomposición.

Verificamos la optimalidad de las colecciones de thrackles máximos y encontramos que ninguna colección, para los tipos de orden descritos anteriormente, es óptima ya que en cada una existe al menos un par de thrackles cuya intersección tiene tamaño mayor a uno. Para este proceso utilizamos el algoritmo 2 de intersección de thrackles descrito en la sección 4.6.2. Si la intersección de cada par de thrackles de la colección es exactamente de tamaño 1, entonces la colección es óptima, de otra manera, la colección no es óptima. Este resultado nos permite dar el siguiente teorema.

Teorema 4.2.1. Sea $K_n(S)$ una gráfica completa inducida por un conjunto S de n puntos, con $3 \leq n \leq 10$, en posición general no convexa. No existen colecciones de thrackles máximos óptimas en $K_n(S)$.

Demostración. Se sigue de los algoritmos de la sección 4.2 y la verificación de optimalidad. El primero busca todas las descomposiciones posibles inducidas por colecciones de thrackles máximos. Econtramos que no existe ninguna colección de thrackles máximos óptima para ningun tipo de orden diferente del convexo para $3 \leq n \leq 10$. □

Las etiquetas de los thrackles máximos para las gráficas dadas por cada tipo de orden de los mostrados en la tabla 4.5, pueden ser consultados en el apéndice A.

La figura 4.2 muestra un ejemplo de una descomposición de K_9, en thrackles, inducida por seis thrackles máximos, cada thrackle es dibujado de

un color diferente.

n	Tipo de Orden	Tamaño de la colección $n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$
8	12	5
8	54	5
9	12	6
9	52	6
9	54	6
9	80	6
9	696	6
9	1080	6
9	1287	6
10	81	6
10	1328	6
10	2243	6

Tabla 4.5: Tipos de orden para los que existe al menos una colección de $n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$ thrackles máximos que cubren a K_n.

Para los casos de $n \in \{3, 4, \ldots, 7\}$ no existe ninguna colección de k thrackles máximos que pueda inducir una descomposición de K_n. Esto es porque, para ninguno de los tipos de orden, en este rango de n, existe una colección de thrackles máximos que cubran las aristas de K_n. Para estos valores de n buscamos, de manera manual, una descomposición con ese número de thrackles (no necesariamente máximos). La figura 4.3 muestra para cada color de n con $3 \leq n \leq 7$, dibujos cuyo anti-thickness es igual a la cota inferior del teorema 4.1.5. De esto se sigue que el anti-thickness geométrico de K_n, con $3 \leq n \leq 7$, es a lo sumo $n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$. Como resultado podemos decir que el anti-thickness geométrico de K_n, con $3 \leq n \leq 10$ es exactamente $n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$.

En esta tesis decidimos analizar qué pasa con el anti-thickness de los tipos de orden en los que no existe ni un thrackle máximo, para esto usamos

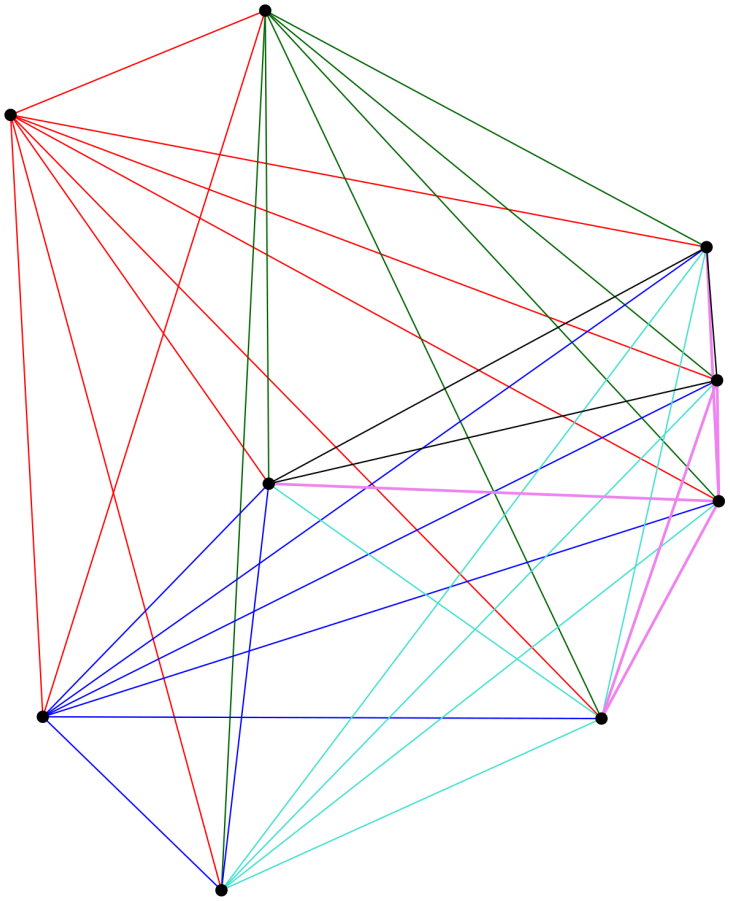


Figura 4.2: Una descomposición de K_9 en seis thrackles, la configuración de puntos corresponde al tipo de orden 12 con respecto de la base de datos de Aichholzer & Krasser (2001). El anti-thickness geométrico de K_9 es 6. Esta descomposición fue inducida por una colección de thrackles máximos que no es óptima, es sencillo ver esto observando que los thrackles tienen 9,7,6,6,5 y 3 aristas respectivamente. Una descomposición óptima de K_9 tiene thrackles de tamaño 9,8,7,6,5 y 4.

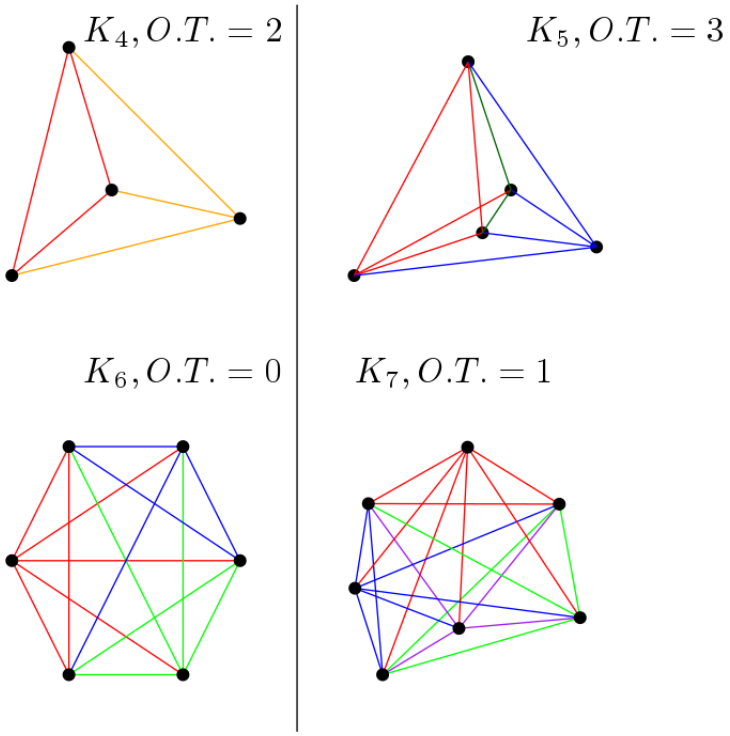


Figura 4.3: Mostramos posibles descomposiciones de la gráfica completa con hasta 7 vértices. Las descomposiciones tienen exáctamente el número de thrackles establecido por la cota inferior del teorema 4.1.5. Estas configuraciones de puntos, a excepción de K_6, son diferentes de la posición convexa. Los tipos de orden correspondientes a cada dibujo están indicados en la figura como $O.T.$

un algoritmo que encuentra, de manera exhaustiva, el anti-thickness de un dibujo dado. Explicamos este resultado a continuación.

4.3. Anti-thickness de gráficas geométricas sin thrackles máximos

Una vez que obtuvimos el anti-thickness geométrico exacto para gráficas completas con hasta 10 vértices, y al observar que podemos alcazar este usando colecciones de thrackles máximos, nos preguntamos ¿Cómo es el anti-thickness para los dibujos que no tienen thrackles máximos? ¿Es mayor? Y sí es así, ¿cuánto más?

Para responder estas interrogantes no es posible formar colecciones de thrackles máximos y evaluar la cobertura de aristas. Por esta razón construimos un algoritmo que encuentra el anti-thickness de un dibujo geométrico inducido por un conjunto de n puntos. El algoritmo es recursivo y usa una técnica de backtracking. Discutimos con más detalle el algoritmo en la sección X.

En teoría, este algoritmo puede encontrar el anti-thickness de cualquier dibujo, para cualquier $n \geq 3$, sin embargo, debido a su costo computacional, solamente hicimos experimentos para $n \leq 8$. Utilizando el algoritmo A para encontrar thrackles máximos descrito en la sección X, listamos cuáles son los tipos de orden que no tienen thrackles máximos. La lista de cada uno de los tipos de orden que cumplen con esta característica puede ser descargada en la siguiente liga: `http.ABCDEFGHIJLKL.com`.

Recordando el trabajo de Araujo *et al.* (2005) donde definen:

$$d(n) = \text{máx}\{\chi(D(S)) : S \subset \mathbb{R}^2 \text{ está en posición general}, |S| = n\},$$

y donde además prueban que :

$$5\left\lfloor \frac{n}{7} \right\rfloor \leq d(n) \leq \text{mín}(n-2, n - \frac{1}{2} - \frac{\lfloor \lg \lg n \rfloor}{2}).$$

Con nuestros resultados podemos decir que encontramos el valor exacto de $d(n)$ para $n \in \{3, 4, 5, 6, 7, 8\}$. Por ejemplo, para $n = 8$, tenemos $5 \leq d(8) \leq 6$, sin embargo, nuestro algoritmo encontró un dibujo de K_8 que requiere al menos 6 thrackles en su descomposición. Esto implica que $6 \leq d(8) \leq 6$ y por lo tanto, $d(8) = 6$. Los números de tipo de orden de tamaño n, para $n \in [3, 8]$, para los que se ejecutó el algoritmo y el anti-thickness encontrado para cada uno pueden descargarse desde la siguiente liga: (`http:ABCDEFGHIJLKL.COM`).

En la siguiente sección analizamos, para $3 \leq n \leq 9$, cómo se comporta la intersección de thrackles, no necesariamente máximos, de los dibujos de K_n.

4.4. Intersección de thrackles de K_n con $3 \leq n \leq 9$.

Debido al resultado del teorema 4.2.1 establecido en la sección 4.2 decidimos estudiar la existencia de colecciones de thrackles, no necesariamente máximos, cuya intersección a pares es vacía y que además puedan cubrir las arsitas de K_n. Para explicar este proceso primero es necesario definir una *partición de un entero.*

Definición 14. *Partición de un entero.*(Knuth (2011b)) Una partición de un entero n es una colección de enteros no negativos $P = \{a_1, a_2, \ldots, a_m\}$ tales que $a_1 \geq a_2 \geq \ldots \geq a_m$ y $a_1 + a_2 + \cdots + a_m = n$.

Es posible usar ciertas particiones de un entero como guía para buscar descomposiciones en thrackles de K_n. Por ejemplo, si deseamos obtener una descomposición en thrackles de K_4, entonces necesitamos cubrir seis aristas. El entero 6 tiene once particiones, tres de ellas son: $\{4, 2\}, \{2, 2, 2\}, \{3, 1, 1, 1\}$. Las cuales pueden traducirse a tres diferentes descomposiciones, la partición $\{4, 2\}$ se traduce en una descomposición con un thrackle con cuatro aristas y uno de dos aristas, la partición $\{2, 2, 2\}$ se traduce en una descomposición con 3 thrackles con dos aristas cada uno, la partición $\{3, 1, 1, 1\}$ se traduce a una descomposición con un thrackle con tres aristas y tres thrackles con una arista. Debido a que nosotros estudiamos descomposiciones de tamaño estrictamente menor a la cota superior del anti-thickness geométrico, que es: $n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$, y como en una descomposición solamente puede existir un thrackle máximo, es posible imponer condiciones sobre las particiones de enteros para las particiones que usaremos. Nos referimos a estas como *particiones válidas de un entero.*

Definición 15. *Partición válida de un entero.* Sea $K_n(S)$ la gráfica completa inducida por un conjunto S de n puntos en posición general. Una partición $P = \{a_1, a_2, \ldots, a_m\}$ es válida si cumple con las siguientes condiciones:

- Si a_1 es igual a n entonces solo existe una ocurrencia de a_1 en P.

- La suma $a_1 + a_2 + \cdots + a_m = \binom{n}{2}$.
- $m < n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$.

No existe una formula cerrada para conocer el número de particiones de algún entero n. Sin embargo, si se considera a las composiciones de un entero n como las secuencias de enteros positivos en donde el orden importa, es decir, dos composiciones diferentes del entero 3 serían $\{2, 1\}$ y $\{1, 2\}$, entonces podemos acotar el espacio de búsqueda de las particiones de un entero. Como el número de composiciones para un entero n es 2^{n-1} podemos decir que existen a lo sumo $O(2^{n-1})$ particiones para el mismo entero n.

Usando esta definición generamos las particiones válidas para todo entero $x \in \left\{\binom{3}{2}, \binom{4}{2}, \cdots, \binom{10}{2}\right\}$. En Knuth (2011b) se ofrece un algoritmo para generar las particiones de algún entero n. Nosotros utilizamos dicho algoritmo en este trabajo modificandolo para que genere solamente particiones válidas de un entero. Discutimos con más detalle el algoritmo en la sección 4.6.4. La tabla 4.6 muestra las particiones generadas por el algoritmo; es posible observar que para $x \in \left\{\binom{3}{2}, \binom{4}{2} \cdots, \binom{7}{2}\right\}$ no existen particiones válidas, esto reafirma, para $3 \leq n \leq 7$ el resultado del teorema 4.1.5 que establece una cota inferior para el anti-thickness geométrico de K_n. Por otro lado, no generamos las particiones para K_10 ya que debido al tamaño de los archivos binarios generados con el algoritmo de búsqueda de thrackles no fue posible diseñar un algoritmo durante el tiempo del desarrollo de este trabajo.

Usando las particiones de la tabla 4.6 diseñamos una serie de algoritmos que buscan thrackles disjuntos usando las particiones como guía. Como hay cerca de 14 algoritmos para realizar esta tarea, expondremos solamente uno de ellos en el algoritmo 7 de la sección 4.6.5, los otros algoritmos son similares a este. Como resultado de estos algoritmos podemos dar el siguiente teorema:

Teorema 4.4.1. Sea $P = \{a_1, a_2, \ldots, a_m\}$ una partición válida para K_8 o para K_9. No existe, para ningún dibujo diferente del convexo, una colección $\mathcal{D} = \{T_1, T_2, \ldots, T_m\}$ de thrackles disjuntos tales que $|E(T_1)| = a_1, |E(T_2)| = a_2, \ldots, |E(T_m)| = a_m$.

Demostración. Se sigue del resultado de la ejecución de los algoritmos para evaluar colecciones de thrackles. Los algoritmos encuentran que, para ninguna partición de la tabla 4.6, existe una colección de thrackles disjuntos cuyo número de aristas corresponde a las particiones válidas de K_8 y K_9. □

n	Particiones válidas de $\binom{n}{2}$
8	$\{8,7,7,6\}$
	$\{7,7,7,7\}$
9	$\{9,8,8,8,3\}$
	$\{9,8,8,7,4\}$
	$\{9,8,8,6,5\}$
	$\{9,8,7,7,5\}$
	$\{9,8,7,6,6\}$
	$\{9,7,7,7,6\}$
	$\{8,8,8,8,4\}$
	$\{8,8,8,7,5\}$
	$\{8,8,8,6,6\}$
	$\{8,8,7,7,6\}$
	$\{8,7,7,7,7\}$

Tabla 4.6: Particiones de enteros del número de aristas de K_8 y de K_9.

Mientras estudiabamos las colecciones de thrackles máximos, decidimos calcular el número de cruce para encontrar una explicación de la existencia de colecciones de thrackles máximos en determinados tipos de orden, detallamos los resultados de este análisis en la siguiente sección.

4.5. Número de cruce de thrackles maximos

En la sección 4.4 dimos como resultado el teorema 4.4.1 que establece la no existencia de ciertas colecciones de thrackles. Durante el desarrollo de este trabajo decidimos analizar el número de cruce de cada uno de los thrackles máximos que hay en una colección que puede inducir una descomposición de K_n. Definimos el número de cruce de una gráfica geométrica a continuación.

Definición 16. *Número de cruce de una gráfica geométrica*(Schaefer (2018)) El número de cruce $cr(\mathsf{G})$ de una gráfica geométrica G es el número de cruces propios entre aristas de G.

Mostramos ejemplos del número de cruce de gráficas geométricas en la figura 4.4; es posible observar que la gráfica K_4 puede ser dibujada con un cruce y también puede ser dibujada como gráfica plana.

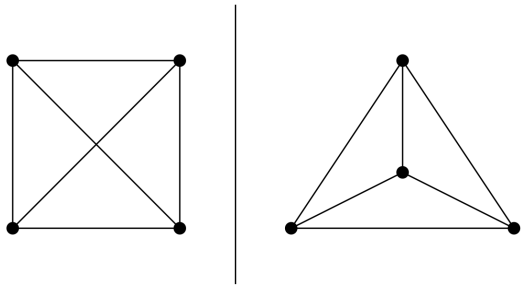


Figura 4.4: A la izquierda, K_4 dibujada con un cruce. A la derecha K_4 dibujada sin cruces.

Para los resultados de esta sección, debemos definir el número de cruce de un thrackle máximo. Para esto, es necesario mencionar cuántos *no cruces* existen en un thrackle geométrico. Para nosotros, un *no cruce* es un par de aristas que son adyacentes entre sí.

Lema 4.5.1. Sea T un thrackle y sea $v \in V(T)$ un vértice de T. El vértice v causa $\binom{deg(v)}{2}$ *no cruces*. En otras palabras: existen $\binom{deg(v)}{2}$ adyacentes a pares para cada vértice $v \in V(T)$.

Demostración. Sea $e = (a, b)$ una arista de T con extremos a y b. Por la definición de thrackle geométrico, toda arista, diferente de e, con extremo en a o en b es adyacente con e y por lo tanto no la cruza. Sea $v \in V(T)$ un vértice con grado $deg(v) = k$ y sea $\{(v, b_1), (v, b_2), \ldots, (v, b_k)\}$ las aristas con un extremo en v. Tenemos que (v, b_1) no cruza a $\{(v, b_2), (v, b_3) \ldots, (v, b_k)\}$, esto es $k-1$ no cruces. Luego, (v, b_2) no cruza a $\{(v, b_3), (v, b_4) \ldots, (v, b_k)\}$, esto es $k-2$ no cruces. Siguiendo de esta manera hasta la arista (v, b_k), como sus no-cruces ya fueron contados anteriormente, entonces, (v, b_k) aporta 0 no cruces. De esta manera, en total, del vértice v podemos contar $(k-1)+(k-2)+\cdots+0$ no cruces, lo que equivale a $\binom{k}{2}$ y como, $k = deg(v)$ el resultado se sigue. □

Teorema 4.5.2. Sea T un thrackle geométrico, con $m = |E(T)|$ el número de cruce $cr(T)$ de T es:

$$\binom{m}{2} - \sum_{u \in V(T)} \binom{deg(u)}{2}$$

Demostración. Como T es un thrackle, cada una de las m aristas debe intersectarse a pares exactamente una vez. En particular, podemos asumir que

cada par de aristas de T se cruza propiamente. Si T está compuesto solamente por aristas que se cruzan a pares, se tiene que para cada $u \in V(T)$ $deg(u) = 1$ y por lo tanto $\sum_{u \in V(T)} \binom{deg(u)}{2} = 0$ y el resultado se sigue. Sin embargo, debemos tener en cuenta que no necesariamente cada par de aristas se cruza sino que algunas pueden ser adyacentes. Dicho de otra manera, debemos contar cuántos no cruces existen en el thrackle. Por el lema 4.5.1 sabemos que existen exactamente $\binom{deg(v)}{2}$ no cruces por cada vértice v del thrackle. El resultado se sigue. □

Basándonos en el resultado del teorema 4.5.2, codificamos un algoritmo para encontrar el número de un thrackle. Y además, para cada colección del apendice A buscamos, para cada thrackle máximo, el número de cruce de cada uno. Detallamos el algoritmo mencionado en la sección 4.6.7.

Encontramos, que en cada colección de la lista anteriormente mencionada, el 50 % de los thrackles máximos de la colección, tiene un número de cruce mínimo con respecto al número de vértices mientras que el otro 50 % de los thrackles tiene número de cruce mucho más alto. Reportamos estos resultados en el apéndice B.

4.6. Algoritmos

En esta sección presentamos los algoritmos usados en este trabajo. Empezamos describiendo el algoritmo para encontar thrackles, de cualquier tamaño, dado un conjunto de puntos en posición general. Después hablamos del algoritmo usado para encontrar la intersección de dos thrackles con el mismo número de aristas. Luego...Finalmente...

Los algoritmos fueron implementados en el lenguaje `C++` en su versión 5 (2017); las implementaciones fueron ejecutadas en un *cluster* con las siguientes características:

- Procesador de 8 núcleos con velocidad de 1.5 GHz.
- Memoria RAM de 64 GB.
- Almacenamiento en disco duro de 128 GB.

Como nuestra implementación no está paralelizada, cada proceso corre en un solo procesador. Por lo que las comparaciones con el tiempo teórico son hechas tomando en cuenta solo un procesador de 1.5 GHz.

Para almacenar información utilizamos, principalmente, una estructura de datos llamada vector. Un vector es un contenedor que representa a un arreglo que puede cambiar de tamaño. Podemos acceder al vector igual que se hace con un arreglo (usando el operador `[]`) con la misma complejidad.

4.6.1. Algoritmo para encontrar thrackles con k aristas

A continuación presentamos un algoritmo para resolver el siguiente problema:

> Problema: Dado n, deseamos encontrar todos los thrackles de tamaño k de todas las gráficasc ompletas $K_n(S)$, donde S es un conjunto de n puntos.
>
> Entrada: Un entero n, con $3 \leq n \leq 10$ y un entero k, con $n \leq k$.
>
> Salida: Por cada tipo de orden de n puntos, una lista de todos los thrackles de tamaño k inducidos por el conjunto.

Este algoritmo requiere tres pasos de preparación para funcionar. Describimos estos pasos a continuación. Para cada tipo de orden S:

1. Leer y almacenar el archivo que contiene el tipo de orden S. Los puntos se almacenan en un vector de tamaño n donde cada elemento es un objeto que contiene las coordenadas x y y de cada uno de los puntos de S.

2. Generar y almacenar las aristas de la gráfica completa inducida por S. Para cada punto almacenado p_i, creamos todas las aristas (no dirigidas) que tienen como punto inicial a p_i y como punto final a p_j, donde $i+1 \leq j \leq n$. Las aristas son almacenadas en un vector de tamaño $\binom{n}{2}$ y las etiquetamos con enteros desde 0 hasta $\binom{n}{2} - 1$.

3. Construir la *matriz de disyunción*. Construimos una matriz binaria de $\binom{n}{2} \times \binom{n}{2}$. La matriz es almacenada en un arreglo de arreglos convencional. Cada indice de las filas y de las columnas representa a cada una de las aristas de $K_n(S)$, de manera tal que la fila i representa a la arista que tiene la etiqueta i, consideramos que la enumeración de las filas y columnas de la matriz empieza desde 0, por lo que $i \in [0, \binom{n}{2} - 1]$. Esto ocurre de la misma manera para las columnas.

La matriz de disyunción tiene un 0 en la entrada (i, j) si las aristas i y j se cruzan o comparten un vértice, y tiene un 1 en en la entrada (i, j) si las aristas i y j son totalmente disjuntas. Es decir, construimos la matriz de adyacencia de la gráfica $D(S)$.

A continuación describimos el algoritmo para encontrar thrackles de tamaño k, este algoritmo usa la matriz de disyunción construida en el paso 3 descrito anteriormente. El pseudocódigo se encuentra en el algoritmo 1.

El algoritmo que diseñamos utiliza la técnica de *backtracking*, recordemos que se desea encontrar todos los thrackles de tamaño k. Para nosotros, un solo thrackle de tamaño k es una solución. Una solución es almacenada en un vector C. Almacenamos las etiquetas de las aristas que conforman un thrackle de tamaño k en las entradas del vector.

En cada iteración, el algoritmo verifica si ya se han encontrado k aristas que se intersectan a pares, para evaluar la intersección se usa la matriz de disyunción.

Supongamos que el vector C tiene entradas desde $C[0]$ hasta $C[j]$ y que $j + 1 < k$. Esto quiere decir que las aristas $C[0], C[1], \ldots, C[j]$ forman un thrackle con $j + 1$ aristas. El algoritmo buscaría extender el tamaño de este thrackle en uno. Para esto hacemos $C[j + 1] = C[j] + 1$ y verificamos que $C[j + 1]$ intersecte a $C[0], C[1], C[2], \ldots, C[j - 1], C[j]$. Si lo hace, entonces $C[j + 1]$ forma parte del thrackle y ahora el thrackle tiene $j + 2$ aristas. En caso contrario hacemos $C[j + 1] = C[j + 1] + 1$, es decir, se descarta la arista que estaba en $C[j + 1]$ y es reemplazada por la arista subsecuente en la etiquetación.

Si en algún momento la entrada $C[j]$ tiene un valor mayor o igual a $\binom{n}{2}$, esto es, que ya agotó los valores posibles para representar alguna arista en K_n, entonces, se incrementa el valor de la entrada $C[j - 1]$ en uno y se continua la verificación a partir de esta entrada. Esto permite que el algoritmo realice el proceso de *backtracking*.

Si el vector C tiene k entradas, es decir, encontró un thrackle con k aristas entonces almacenamos el contenido del vector C en una lista de vectores. Como ya se encontró una solución y esta ha sido procesada, para buscar la siguiente solución, le indicamos al algoritmo que el thrackle tiene actualmente tamaño $k - 1$, provocando así que en la siguiente iteración se busque incrementar el tamaño del thrackle en uno.

Mostramos un ejemplo de cómo se llena el vector C en una ejecución del algoritmo en la figura 4.5. En el inicio de la ejecución del algoritmo, el vector

Algoritmo 1: Algoritmo para encontrar todos los thrackles de tamaño k.

```
1  función EncontrarThrackle (n, k)
   Entrada: Un entero n, un entero k.
   Salida : Una lista de thrackles de tamaño k para cada tipo de
            orden de n.
2  Sea C un vector de tamaño k + 1; C[0] ← 0
3  C[i] ← NIL para 1 ≤ i ≤ k + 1
4  inters_flag ← true
5  curr_size ← 0
6  while C[0] < (n 2) do
7      while curr_size< k do
8          inters_flag = true
9          if C[curr_size] ≥ (n 2) then
10             curr_size ← curr_size −1
11             if curr_size < 0 then
12                 return Lista L y thrackle_counter
13             C[curr_size] ← C[curr_size] + 1
14             continue
15         for i ← 0 . . . curr_size − 1 do
16             inters_flag=inters_flag &&
17             matrix[C[i]][Ccurr_size]
18         if inters_flag == False then
19             C[curr_size] ← C[curr_size] + 1
20             continue
21         else
22             if curr_size+1 == k then
23                 thrackle_counter ← thrackle_counter + 1
24                 Almacenar C en una lista de vectores L.
25                 continue
26         C[curr_size + 1] ← C[curr_size] + 1
27     curr_size ← curr_size +1
```

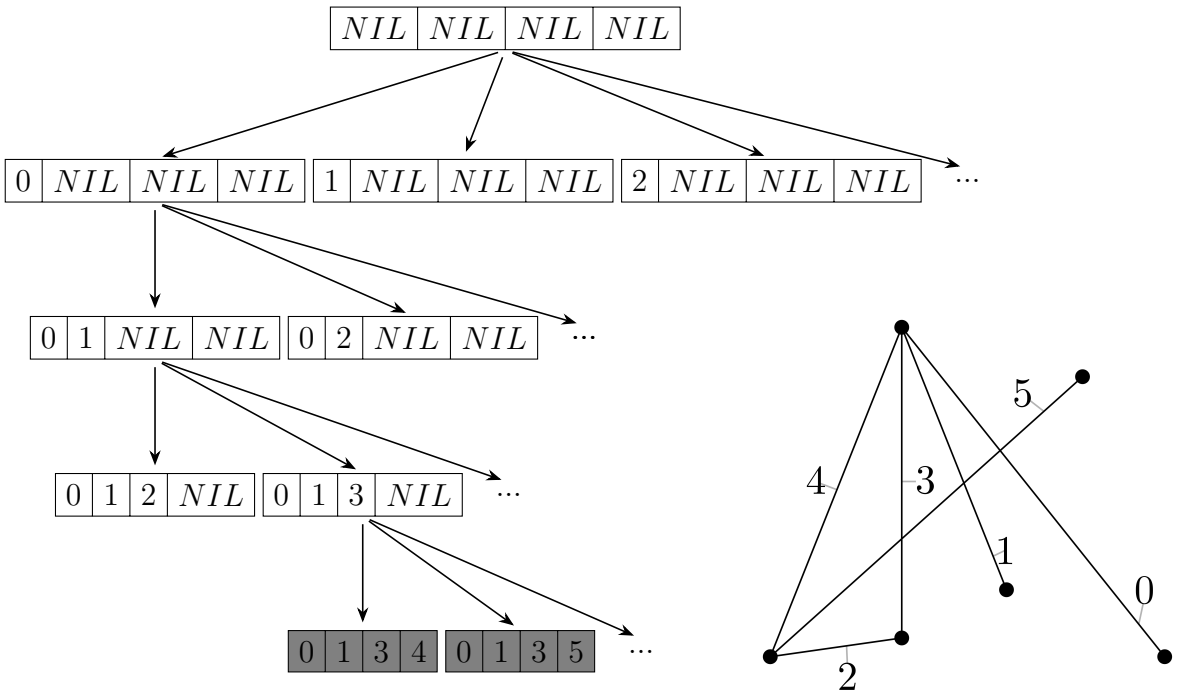


Figura 4.5: La figura muestra un ejemplo de cómo cambia el vector C cuando se busca un thrackle con 4 aristas para alguna $n > 5$. En el lado derecho presentamos un dibujo de una gráfica en la que se buscan los thrackles con 4 aristas. Solamente se representan algunas instancias del vector C. En este ejemplo suponemos que las aristas con etiqueta 1 y 2 no se intersectans. Es posible observar que cuando $C = [0, 1, 2, NIL]$ no se generan, debajo de ese nivel, combinaciones con las etiquetas 1 y 2. Se marcan de fondo gris las soluciones encontradas.

C está vacío, es decir, que todas sus entradas tienen un valor nulo. Esto se representa en la raíz del árbol generado implicitamente por el backtracking. Podemos pensar que el arbol implicito se construye primero a profundiad, similar a un recorrido de búsqueda en profundidad (DFS). El primer subárbol que se genera, después de la raiz, es aquel cuya raiz tiene a $C[0] = 0$ y las demás posiciones con un valor nulo, llamemosle r_1. Si aún no se alcanza el tamaño de thrackle deseado, después generamos, como hijo de r_1, el subárbol cuya raíz tiene a $C[0] = 0$ y $C[1] = 1$ y las demás posiciones con un valor nulo, llamemosle r_2. Si aún no se alcanza el tamaño de thrackle deseado, después generamos, como hijo de r_2, el subárbol cuya raíz tiene a $C[0] = 0, C[1] = 1$ y $C[2] = 2$ y las demás posiciones con un valor nulo. Repetimos este proceso hasta que alcancemos el tamaño de thrackle que buscamos. Cada nodo genera un hijo siempre y cuando las aristas representadas en C se intersecten a pares. En la figura 4.5 mostramos un ejemplo de un árbol generado por el algoritmo que tiene 4 niveles.

En la primera iteración hacemos $C[0] = 0$. Esta inicialización es necesaria para poner en marcha el algoritmo. De esta manera, el algoritmo encuentra primero todos los thrackles que contengan a la arista con etiqueta 0 en la primera posición. Una vez que la posición $C[0]$ tenga un valor mayor o igual a $\binom{n}{2}$, el valor de $C[0]$ será igual a 1. Esto hará que el algoritmo busque los thrackles que contengan a la arista con etiqueta 1 en la primera posición. Este proceso se repite para todos los valores entre 0 y $\binom{n}{2} - 1$.

Este algoritmo solamente genera soluciones que son thrackles. Puesto que en C solo se avanza a la siguiente posición cuando se encuentra que la arista de la posición actual intersecta a las aristas anteriormente establecidas. Además, el algoritmo genera todos los thrackles con k aristas, si no lo hiciera significaría que no evaluó determinada combinación de aristas. Recordemos que las aristas están etiquetadas con enteros desde 0 hasta $\binom{n}{2} - 1$. Supongamos que existe un subconjunto $T = \{a_1, a_2, \ldots, a_k\}$ de las etiquetas $\{0, 1, 2, \ldots, \binom{n}{2} - 1$ que representa un thrackle de tamaño k y que el algoritmo no evaluó. Existe un subárbol cuya raíz, llamemosle r_1 tiene a $C[0] = a_1$ y, por consiguiente r_1, tiene como hijo al subárbol cuya raíz, llamemosle r_2 tiene a $C[0] = a_1$ y a $C[1] = a_2$ y así sucesivamente hasta llegar a un nodo sin hijos que tiene a $C[0] = a_1, C[1] = a_2, \ldots C[k] = a_k$, si el algoritmo no evaluó esta combinación, signfica que no generó alguno de los subárboles, en la línea 25 del algoritmo, que tienen como descendiente al nodo $C[0] = a_1, C[1] = a_2, \ldots C[k] = a_k$ y esto sucede cuando existe al menos un par de aristas $a_i, a_j \in T$ que son disjuntas. Esto contradice la suposición

de que T representa un thrackle, por lo tanto, si T es un thrackle, entonces el algoritmo lo evalúa, esto sucede en el ciclo `for` de la línea 15.

Para el caso particular de la entrada $C[0]$, se toman los valores en el rango $\left[0, \binom{n}{2}\right]$. Las combinaciones que se descartan, al hacer la evaluación en la línea 18, son aquellas que no forman un thrackle, estas son las combinaciones que contienen al menos una arista disjunta de alguna de las otras aristas de la combinación.

Análisis de complejidad

El caso en el que este algoritmo realiza más cálculos es aquel en el que no se descarta ninguna combinación de tamaño t para $t \leq k$ y por lo tanto el algoritmo analiza, en tiempo $O(t)$, cada combinación en el ciclo de la línea 15. Para establecer el costo computacional del algoritmo es necesario saber cuántas combinaciones de tamaño t existen para cada $t \leq k$ y multiplicar esto por el costo de evaluar cada combinación. Como hay $\binom{n}{2}$ aristas y estamos suponiendo que cualquier combinación de tamaño t es válida, existen $\binom{\binom{n}{2}}{t}$ combinaciones de dicho tamaño. Se necesitan $t\binom{\binom{n}{2}}{t}$ operaciones para evaluar todas las combinaciones de tamaño t. En nuestro caso $t \in [1, 10]$, por lo tanto el costo computacional del algoritmo, en el peor caso es:

$$\begin{aligned} O\left(\sum_{t=1}^{10} t\binom{\binom{n}{2}}{t}\right) &= O\left(\binom{\binom{n}{2}}{1} + 2\binom{\binom{n}{2}}{2} + 3\binom{\binom{n}{2}}{3} + \cdots + 10\binom{\binom{n}{2}}{10}\right) \\ &= O(n^2) + O(n^4) + O(n^6) + \cdots + O(n^{20}) \\ &= O(n^{20}), \end{aligned} \tag{4.5}$$

para cada tipo de orden de n puntos.

En teoría, el tiempo que necesitariamos para ejecutar este algoritmo en el cluster es significativo. En la tabla 4.7, mostramos cuánto tiempo es requerido por este algoritmo con complejidadd $O(n^{20})$ tomando en cuenta la velocidad de procesamiento del cluster. También mostramos cuánto tiempo tardó su ejecución para $n = \{8, 9, 10\}$.

Los datos de los thrackles máximos encontrados para cada $3 \leq n \leq 9$ pueden ser descargados de la siguiente liga: `http://computacion.cs.cinvestav.`

n	Tiempo teórico en cluster	Tiempo real en cluster
10	2113.99 años	3 días
9	257.01 años	12 minutos
8	24.36 años	6 segundos

Tabla 4.7: Tiempo de ejecución teórico del algoritmo y tiempo real de ejecución en el cluster.

`mx/~dmerinos/site/archivos_tesis/3to9.tar.gz` el archivo para $n = 10$, separado debido a su tamaño, puede ser descargado de la siguiente liga: `http://computacion.cs.cinvestav.mx/~dmerinos/site/archivos_tesis/10.tar.gz`

4.6.2. Algoritmo para la intersección de dos thrackles

Dados dos thrackles con el mismo número de aristas, con un arreglo de enteros que represente a sus aristas, aprovechamos el ordenamiento lexicográfico para hacer la operación de intersección de thrackles en tiempo lineal. Sea A y B dos vectores que representan las aristas de dos thrackles, con $|A| = |B| = k$. El algoritmo 2 entrega la intersección de A y B y la almacena en un conjunto C. Este algoritmo es secuencial y visita en el peor caso todas las entradas de los dos vectores, en nuestra implementación, los thrackles almacenan la información de sus aristas en un vector de tamaño $\binom{n}{2}$ para cada n en el rango $[3, 10]$.

4.6.3. Algoritmo para encontrar colecciones de thrackles máximos de K_n

Aquí presentamos un algoritmo para resolver el siguiente problema:

> Problema: Determinar si existen y encontrar, para cada tipo de orden de n puntos, con $3 \leq n \leq 10$, las colecciones de thrackles máximos que pueden inducir una descomposición de K_n.
>
> Entrada: Un entero n, con $3 \leq n \leq 10$.
>
> Salida: Para cada tipo de orden de n puntos, una lista de colecciones de thrackles máximos que pueden inducir una descomposición de K_n.

Algoritmo 2: Intersección de dos conjuntos ordenados en tiempo lineal.

```
1  función Intersection (A, B, C)
   Entrada: Dos conjuntos A, B del mismo tamaño y un conjunto C
   Salida  : Un conjunto C con la intersección de A y B
2  i ← 0
3  j ← 0
4  while i < k and j < k do
5      if A[i] < B[j] then
6          i ← i + 1
7          continue
8      if A[i] == B[j] then
9          C ← C ∪ {A[i]}
10         i ← i + 1
11         j ← j + 1
12         continue
13     if A[i] > B[j] then
14         j ← j + 1
15         continue
```

El algoritmo que diseñamos para este resultado es exhaustivo y secuencial.

Este algoritmo necesita un paso de preparación para funcionar: Para cada tipo de orden, para $3 \leq n \leq 10$, examinamos cuáles tienen al menos $n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$ thrackles máximos. De los resultantes, examinamos cuáles tipos de orden cubren, con la unión de sus thrackles máximos, todas las aristas de K_n.

Algoritmo 3: Búsqueda de colecciones de thrackles máximos que inducen una descomposición de K_n

```
1 función FindThrackleCollections (n, ot)
    Entrada: Un entero n, un entero ot que representa algún tipo de
             orden válido de n.
    Salida : Las combinaciones de At_g(K_n) thrackles máximos que
             pueden inducir una descomposición del dibujo de K_n
             inducido por el tipo de orden ot.
2 c ← n − ⌊√(2n + 1/4) − 1/2⌋
3 T ← colección de thrackles máximos del tipo de orden ot
4 k ← |T|
5 C ← combinaciones de tamaño c del conjunto {1, 2, ..., k}
6 foreach t ∈ C do
7 |   solution ← {T[t[1]], T[t[2]], ..., T[t[k]]}
8 |   if solution cubre las aristas de K_n then
9 |   |   Almacenar solution en una lista de soluciones.
```

Después de la etapa de preparación, podemos empezar la búsqueda de las colecciones de thrackles máximos. Mostramos el pseudocódigo en el algoritmo 3. Una vez que sabemos cuáles son los tipos de orden candidatos a tener una colección de thrackles máximos que induzcan una descomposición de K_n, buscamos, de manera exhaustiva, todas las combinaciones posibles de $n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$ thrackles máximos. Para cada una de las combinaciones verificamos si esta cubre o no a las aristas de K_n. Si la combinación cubre, almacenamos las etiquetas en una lista de soluciones y examinamos la siguiente combinación. En caso contrario descartamos esa combinación y continuamos examinando.

Las combinaciones fueron generadas usando el algoritmo descrito en Knuth (2011a). Mostramos el pseudocódigo del algoritmo para generar las combinaciones en el algoritmo 4 en la página 69. Dado un entero t, si queremos

generar las combinaciones de c elementos, algoritmo 4 tiene una complejidad de $O\left(\binom{t}{c}\right) = O(t^c)$.

En nuestro trabajo, c es fija para cada n ya que hacemos $c = At_g(K_n)$. Recordemos que $At_g(K_n) = n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$. Sin embargo, t depende de dos parámetros: uno es n y el otro es el tipo de orden de tamaño n con el que trabajemos. Por ejemplo, para $n = 6$ el tipo de orden 1 tiene 13 thrackles máximos, por lo que buscariamos generar las $\binom{13}{3}$ combinaciones. Por otro lado, el tipo de orden 7 tiene 5 thrackles máximos, por lo que buscariamos generar las $\binom{5}{3}$ combinaciones. Esto tiene un impacto en la complejidad del algoritmo.

Sea $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$ una combinación generada por el algoritmo. $c_i \in \mathcal{C}$ representa una etiqueta de un thrackle máximo. Recordemos que el algoritmo entrega, para cada tipo de orden de tamaño n, una lista $\{\mathcal{C}_\infty, \mathcal{C}_\infty, \ldots, \mathcal{C}_|\}$, donde cada $\mathcal{C}_\rangle$ es una colección de thrackles máximos que pueden inducir una descomposición de K_n. Por ejemplo, si $\mathcal{C}_| = \{23, 45, 99\}$ entonces $\mathcal{C}_|$ representa una colección de tres thrackles máximos, los cuales tienen etiquetas 23, 45 y 99 respectivamente. Es posible buscar, en el archivo generado por el algoritmo 1 que busca thrackles con k aristas, el thrackle que corresponde con determinada etiqueta y realizar operaciones sobre este.

Para cada combinación calculada, el algoritmo verifica si esta cubre o no a las aristas de K_n. Hacemos esta verficicación con un algoritmo secuencial que recorre el arreglo de aristas de cada thrackle identificado por las etiquetas de la combinación. En nuestra implementación cada thrackle de K_n tiene asociado un arreglo de $\binom{n}{2}$ posiciones.

Análisis de complejidad

Sea $x = n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$, para un tipo de orden válido (un tipo de orden con al menos x thrackles máximos) de tamaño n, con m thrackles máximos, el algoritmo 3 genera las $\binom{m}{x}$ combinaciones y después, para cada una, examina, en tiempo $O((n-x) \cdot n^2)$, si esta cubre o no a las aristas de K_n. Luego, este algoritmo tiene complejidad:

$$O\left(\binom{m}{x} \cdot (n-x)n^2\right) = O\left(\binom{m}{n} n^3\right) = O(m^n n^3) \tag{4.6}$$

Es importante notar que la complejidad establecida en la ecuación 4.6 es para un tipo de orden. Nosotros ejecutamos este algoritmo para cada tipo de

Algoritmo 4: Algoritmo de Knuth para generar las combinaciones de tamaño t del conjunto $\{0, 1, 2, \ldots, n-1\}$.

```
1  función KnuthCombinations (n, t)
   Entrada: Un entero n, un entero t con 0 ≤ t ≤ n
   Salida : Las combinaciones de t elementos de los n números en el
            conjunto 0, 1, 2, ..., n − 1.
2  for j = 1 ... t do
3  |  c[j] ← j − 1
4  c[t + 1] ← n
5  c[t + 2] ← 0
6  L2:
7  Almacenar la combinación c[t], c[t − 1], ..., c[0].
8  j ← 1
9  while c[j] + 1 == c[j + 1] do
10 |  c[j] ← j − 1
11 |  j ← j + 1
12 if  j > t then
13 |  return
14 c[j] ← c[j] + 1
15 goto L2
```

orden válido. Para saber cuántos thrackles máximos hay en determinado tipo de orden usamos el algoritmo 1 descrito en la sección 4.6.1. En la tabla 4.9 damos un panorama de los valores que toma m para $6 \leq n \leq 10$. Con esto ilustramos también los valores que toma $\binom{m}{At_g(K_n)}$, es decir, el número de combinaciones a evaluar por el algoritmo en el peor caso.

A pesar de que la complejidad descrita por la ecuación 4.6 es bastante grande, en la ejecución que realizamos obtuvimos tiempos de ejecución aceptables para el trabajo. En la tabla 4.8 describimos el tiempo de ejecución necesario para el tipo de orden de tamaño $n = \{8, 9, 10\}$ que tiene el mayor número de thrackles máximos y no es el convexo, y el tiempo de ejecución que tomó analizar todos los tipos de orden válidos.Podemos notar que a pesar de que en el caso de K_{10} hay aproximadamente $1.8E^{12}$ combinaciones posibles para uno de los tipos de orden de 10 puntos, el algoritmo termina en un tiempo considerable para nosotros.

n	m	Tiempo de ejecución teórico	Tiempo de ejecución real
8	94	2×10^{450} años	0.70 segundos
9	213	3×10^{193} años	25 minutos
10	459	1.3×10^{74} años	4 días y 2 horas

Tabla 4.8: Mostramos el tiempo de ejecución real del algoritmo para todos los tipos de orden. La segunda columna indica cuántos thrackles máximos hay en como máximo en algún tipo de orden válido para cada n. La tercera columna indica cuánto tiempo necesitaría para acabar, en el peor caso. La última columna indica cuánto tiempo se necesitó en realidad.

n	Número máximo de thrackles en un T.O.	$At_g(K_n)$	Número de combinaciones
6	5	3	10
7	16	4	1,820
8	49	5	1,906,884
9	134	6	7,177,979,809
10	333	6	1,809,928,822,548

Tabla 4.9: Se cuantifica el número de combinaciones a examinar por el algoritmo para el tipo de orden válido con más thrackles de cada $3 \leq n \leq 10$. Para los casos de $n = \{3, 4, 5\}$ no existen tipos de orden, diferentes al convexo, tal que la unión de thrackles máximos cubra las aristas de K_n.

4.6.4. Algoritmo para generar particiones válidas de un entero

El algoritmo proporcinado en Knuth (2011b) genera las particiones de algún entero n con $O(n^2)$ operaciones, mostramos el pseudocódigo en el algoritmo 5. Una vez que las particiones son generadas, examinamos aquellas que tengan menos de $n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$ elementos y evaluamos si son válidas. Una partición P, de tamaño $n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$, de n es una *partición guía* si 1) todo elemento de P es menor o igual a n y 2) si algún $p_i \in P$ es igual a n, entonces solamente hay una ocurrencia de p_i en P.

Nuestra función para evaluar si una partición es una partición guía consiste en visitar cada posición de un arreglo con a lo sumo $n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$

Algoritmo 5: Algoritmo de Knuth para generar las particiones de un entero n.

```
1  función KnuthPartitions (n)
   Entrada: Un entero n, con n ≥ 1.
   Salida : Las particiones P = {a_1, a_2, ..., a_k} del entero n, tal que
            a_1 ≥ a_2 ≥ ... ≥ a_k} y a_1 + a_2 + ··· + a_k = n.
2  P1. a_0 ← 0, m ← 1
3  P2. a_m ← n, q getsm − [n = 1]
4  P3. Almacenar la partición {a_1, a_2, ..., a_k}.
5  if a_q ≠ 2 then
6      goto P5
7  P4. a_q ← 1 q ← q − 1, m ← m + 1, a_m ← 1.
8  goto P4
9  P5. if q == 0 then
10     return
11 else
12     x ← a_q − 1
13     a_q ← x
14     n ← m − q + 1
15     m ← q + 1
16 P6. if n ≤ x then
17     goto P2
18 else
19     a_m ← x
20     m ← m + 1
21     n ← n − x
22     goto P6
```

elementos y verificar que solamente exista una ocurrencia de n. Mostramos el pseudocódigo en el algoritmo 6. Esta tarea requiere analizar, en tiempo $O(1)$ cada una de los elementos del arreglo, cuyo tamaño es $O(n)$, por ello la prueba tiene complejidad lineal. Esta función es usada por el algoritmo de generación de particiones guía.

Algoritmo 6: Algoritmo para evaluar si una partición es válida o no.

```
1  función ParticionValida (n, P)
   Entrada: Un entero n, una partición P de n de tamaño k.
   Salida  : true, si la partición P es válida, false en otro caso.
2  c ← 0
3  foreach i ∈ P do
4      if i > n then
5          return false
6      if i == n then
7          c ← c + 1
8      if c > 1 then
9          return false
10 return true
```

Análisis de complejidad

El algoritmo para generar las particiones de un entero n tiene complejidad $O(n^2)$. Adicionalmente, nosotros verificamos que las particiones tengan menos de $n - \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$ elementos en tiempo lineal. Luego, para cada partición con el tamaño deseado, evaluamos si es válida o no. El costo total de este algoritmo es:

$$O(n^2 \cdot n \cdot n) = O(n^4)$$

La razón principal por las que definimos las particiones guía es porque pueden ser utilizadas como guía para buscar descomposiciones de K_n donde cada elemento de la partición indica el tamaño de un thrackle. A continuación presentamos cómo utilizamos estas particiones para construir una serie de algoritmos.

4.6.5. Algoritmo para encontrar descomposiciones por thrackles de K_n usando particiones de enteros

Este algoritmo fue diseñado para resolver el siguiente problema:

> Problema: Dada una partición $P = \{a_1, a_2, \ldots, a_m\}$ de un entero n con $m < At_g(K_n)$, deseamos encontrar si existe, para algún dibujo de n, una colección $\mathcal{C} = \{T_1, T_2, \ldots, T_m\}$ de thrackles disjuntos tales que $|E(T_i)| = a_i$.
>
> Entrada: Una partición guía P de n.
>
> Salida: *true* si existe una colección de thrackles con las caracteristicas descritas en el problema. $false$ en otro caso.

Como resultado del algoritmo para encontrar particiones válidas descrito en la sección 4.6.4, tenemos una lista de particiones válidas para $n \in \{8, 9\}$ que presentamos en la tabla 4.10. En esta función se cargan, a memoria, los datos de los thrackles, en el caso de K_{10}, los archivos de los thrackles máximos y thrackles con 9 aristas exceden el tamaño de la memoria del cluster. Por ello las particiones de $n = 10$ no fueron evaluadas en este trabajo.

A continuación presentamos, con un caso particular, el esquema general de los algoritmos usados para verificar si existen colecciones de thrackles cuyos tamaños correspondan a los elementos de una partición. Este algoritmo utiliza la técnica de *backtracking*, presentamos el pseudocódigo en el algoritmo 7.

Para el funcionamiento de este algoritmo es necesario tener los datos de los thrackles de cada tamaño indicado por la partición, por ejemplo, si se trabaja con $n = 8$ y la partición guía es $\{8, 7, 7, 6\}$ se debe usar la función de búsqueda de thrackles con k aristas descrito en la sección 4.6.1 para buscar los thrackles con ocho, siete y seis aristas. No es necesario tener dos archivos diferentes para cada uno de los dos thrackles de tamaño siete, es suficiente tener un archivo para cada tamaño diferente.

Tomemos el caso de la primera partición de ocho, $P = \{8, 7, 7, 6\}$, en este caso queremos verificar si existe o no una colección de cuatro thrackles, uno de tamaño ocho, dos de tamaño siete y uno de tamaño seis, tal que sean disjuntos a pares. Para buscar esta colección, podemos diseñar un algoritmo que evalúe, exhaustivamente, la intersección de cada thrackle A de tamaño ocho, con cada thrackle B de tamaño siete, si existen A y B disjuntos, verificamos

Algoritmo 7: Algoritmo para buscar una colección de thrackles disjuntos de K_n dada una partición válida de n.

```
1  función 8876 (n, P)
   Entrada: Un entero n, Una partición válida P de n de tamaño k.
   Salida : true, si existe una colección de thrackles disjuntos en la
            que los tamaños de los thrackles corresponden a los
            elementos de P; false en otro caso.
2  Begin
3      Sea Z un vector booleano de 28 posiciones. for i ← 0...28 do
4          Z[i] = 0
5      end
6      Almacenar el estado actual de Z en el arreglo Z_1.
7      foreach Thrackle A de tamaño 8 do
8          Sea A.bedges el vector booleano que representa las aristas de
             A.
9          for i ← 0...26 do
10             if A.bedges[i] == 1 then
11                 if Z[i] == 1 then
12                     Restaurar Z a Z_1.
13                     Descartar thrackle actual y continuar con el
                        siguiente de tamaño 8.
14                 end
15                 else
16                     Z[i] ← 1
17                 end
18             end
19         end
20         Almacenar el estado actual de Z en el arreglo Z_2.
21         foreach Thrackle B de tamaño 7
22             Sea B.bedges el vector booleano que representa las aristas
                de B.
23             for i ← 0...26 do
24                 if B.bedges[i] == 1 then
25                     if Z[i] == 1 then
26                         Restaurar Z a Z_2.
27                         Descartar thrackle actual y continuar con el
                            siguiente de tamaño 7.
28                     end
29                     else
30                         Z[i] ← 1
31                     end
32                 end
33             end
34     end
```

```
36
37      Almacenar el estado actual de Z en el arreglo Z_3.;
38      foreach Thrackle C de tamaño 7 do
39          Sea C.bedges el vector booleano que representa las aristas de
             C.;
40          for i ← 0...26 do
41              if C.bedges[i] == 1 then
42                  if Z[i] == 1 then
43                      Restaurar Z a Z_3.;
44                      Descartar thrackle actual y continuar con el
                         siguiente de tamaño 7.;
45                  else
46                      Z[i] ← 1;
47          Almacenar el estado actual de Z en el arreglo Z_4.;
48          foreach Thrackle D de tamaño 6 do
49              Sea D.bedges el vector booleano que representa las aristas
                 de D.;
50              for i ← 0...26 do
51                  if D.bedges[i] == 1 then
52                      if Z[i] == 1 then
53                          Restaurar Z a Z_4.;
54                          Descartar thrackle actual y continuar con el
                             siguiente de tamaño 6.;
55                      else
56                          Z[i] ← 1;
```

n	Particiones válidas de $\binom{n}{2}$
8	$\{8,7,7,6\}$
	$\{7,7,7,7\}$
9	$\{9,8,8,8,3\}$
	$\{9,8,8,7,4\}$
	$\{9,8,8,6,5\}$
	$\{9,8,7,7,5\}$
	$\{9,8,7,6,6\}$
	$\{9,7,7,7,6\}$
	$\{8,8,8,8,4\}$
	$\{8,8,8,7,5\}$
	$\{8,8,8,6,6\}$
	$\{8,8,7,7,6\}$
	$\{8,7,7,7,7\}$

Tabla 4.10: Particiones de enteros del número de aristas de K_8 y de K_9.

que exista algún thrackle C de tamaño siete que sea disjunto con A y con B y, finalmente, si existe, buscamos un thrackle D, de tamaño seis que sea disjunto con A, B y C.

Para verificar que los thrackles sean disjuntos, mantenemos un arreglo booleano Z de $\binom{8}{2}$ posiciones. Cuando se analiza un thrackle T de tamaño 8 recorremos el arreglo de aristas de T, cuyo tamaño es también $\binom{8}{2}$. Como T tiene 8 aristas entonces el arreglo de aristas de T tiene exactamente 8 posiciones con valor uno y el resto con valor cero. Para cada posición p de T cuyo valor sea uno, marcamos la misma posición en Z con un uno. Esto significa que el thrackle T cubre a la arista p.

Si en algún momento debemos marcar alguna posición p del arreglo Z y esta ya tiene un valor de uno entonces un thrackle, anteriormente analizado, ya cubre esa arista y por lo tanto el thrackle que se está verificando actualmente no es disjunto con alguno anterior. Por lo tanto descartamos ese thrackle y continuamos verificando con el thrackle subsecuente del archivo.

Si no encontramos ningún thrackle que sea disjunto con los anteriormente analizados, el algoritmo descarta el último thrackle que fue compatible con los demás y continúa con el siguiente del mismo tamaño del thrackle descartado.

El algoritmo termina, retornando un valor *true*, cuando encuentra un

thrackle con ocho aristas, dos con siete aristas y uno con seis aristas que son disjuntos a pares o bien, cuando no hay más thrackles que analizar; en este caso el algoritmo retorna un valor $false$.

Este algoritmo evalúa, en el peor caso, todas las combinaciones posibles de cuatro thrackles con tamaños 8,7,7 y 6 respectivamente.

En nuestra implementación, decidimos almacenar los datos de los thrackles en la memoria del cluster para que el acceso fuera mucho más rápido que leer repetitivamente los archivos creados por el algoritmo de búsqueda de thrackles con k aristas presentado en la sección 4.6.1.

Para las particiones guía presentadas en la tabla 4.10, en la algunos de los casos no fue necesario construir funciones que evaluaran toda la partición. Para el caso de $n = 9$ y las particiones $\{9, 8, 8, 8, 3\}$, $\{9, 8, 8, 7, 4\}$, $\{9, 8, 8, 6, 5\}$ el construimos un algoritmo que verifica la existencia de una colección de thrackles de tamaños nueve, ocho y ocho. Para las particiones $\{9, 8, 7, 7, 5\}$, $\{9, 8, 7, 6, 6\}$, $\{9, 7, 7, 7, 6\}$ construimos un algoritmo que verifica la existencia de una colección de thrackles de tamaños nueve, siete, siete, siete y seis. Para el resto de las particiones guía de $n = 9$ escribimos un algoritmo para cada una. Para las dos particiones guía de $n = 9$ construimos dos algoritmos.

Las funciones construidas permitieron dar el resultado del teorema 4.4.1 de la sección 4.4. Estas funciones pueden encontrarse en la siguiente liga:
`https://github.com/demaseme/programas_tesis_rev3/tree/master/cpps`.

Análisis de complejidad

La complejidad de este algoritmo está dado por el número de thrackles de cada tamaño establecido por la partición. Si existen k thrackles de tamaño 8, m thrackles de tamaño 7 y q thrackles de tamaño 6. El algoritmo realiza $O(k \cdot m^2 \cdot q)$ operaciones. Notese el exponente de m en la complejidad, recordemos que la partición guía es $\{8, 7, 7, 6\}$ esto implica que por cada thrackle de tamaño 7, verificaremos con cada uno de los otros thrackles de tamaño 7 y por ello debemos considerar $m \times m$ en la complejidad.

De manera general para una partición guía $P = \{a_1, a_2, \ldots, a_m\}$, si existen t_i thrackles de tamaño a_i, en el peor caso hay

$$(\binom{n}{2}^n t_i = \binom{\binom{n}{2}}{a_i} \tag{4.7}$$

thrackles de tamaño a_i. La ecuación 4.7 alcanza el máximo cuando $a_i = n$. Por lo que un algoritmo exhaustivo como el aquí presentado que usa como

partición guía una partición que tiene m elementos tiene, en el peor caso, una complejidad de

$$O\left(\binom{\binom{n}{2}}{n}^m\right) = O(n^{2nm})$$

.

Por otro lado 4.7 se minimiza cuando $a_i = 1$, por lo que el algoritmo tiene como mejor caso una complejidad de :

$$\omega\left(\binom{\binom{n}{2}}{1}^m\right) = \omega\left(\binom{n}{2}^m\right) = \omega(n^{2m})$$

.

4.6.6. Algoritmo para encontrar el anti-thickness de un dibujo de K_n

El algoritmo presentado en esta sección resuelve el siguiente problema:

> Problema: Dado un conjunto de n puntos en posición general, deseamos encontrar el anti-thickness de la gráfica geométrica completa inducida por el conjunto.
>
> Entrada: Un conjunto de n puntos en posición general.
>
> Salida: El anti-thickness de la gráfica completa inducida por el conjunto de puntos.

Este es un algoritmo exhaustivo y recursivo que usa la técnica de backtracking para encontrar el anti-thickness de un dibujo de K_n. El algoritmo realiza esta tarea buscando colecciones de thrackles cuyo tamaño sea mínimo, como en la definición 5 de anti-thickness de un dibujo presentado en la sección 2.2.

Este algoritmo construye, implicitamente, un árbol en el que cada nodo es un thrackle y un camino desde la raiz hasta algún nodo es una colección de thrackles. Una colección de thrackles es una descomposición si la intersección a pares de cada thrackle en la colección es vacía y si la unión de las aristas de todos los thrackles es igual al conjunto de aristas de K_n. Decimos que dos thrackles son compatibles si su intersección es vacía. En este árbol la raíz es el nivel cero, los hijos de la raiz son el nivel 1, los hijos de los hijos de la raiz son el nivel 2 y así sucesivamente. El algoritmo baja de nivel en el árbol

cuando encuentra un thrackle compatible con el resto de los thrackles que ya están en la colección.

El algoritmo busca el thrackle de mayor tamaño posible en cada llamada recursiva para intentar agregarlo a la colección. Para poder agregarlo debe ser disjunto con los thrackles que ya existan en la colección. El algoritmo termina cuando no puede encontrar más colecciones de thrackles que cubran la gráfica completa y que tenga tamaño mínimo o bien, cuando las colecciones que encuentra son más grandes que el anti-thickness encontrado actualmente.

Este algoritmo requiere que se lean los puntos y se induzca la gráfica completa así como la inicialización de la matriz de disyunción de la misma que mencionamos en la sección 4.6.1 del algoritmo de búsqueda de thrackles con k aristas.

En el algoritmo representamos un thrackle con un vector de enteros que representan las etiquetas de las aristas que inducen al thrackle. El vector de enteros tiene valores desde 0 hasta $\binom{n}{2} - 1$ ya que este es el rango de etiquetación de las aristas de K_n.

Presentamos el pseudcódigo en el algoritmo 8, donde $minAt$, $matrix$, $cols$, n y $coveredEdges$ son variables globales. En el pseudcódigo $minAt$ es un entero que representa el tamaño mínimo de la colección de thrackles encontrada hasta determinado momento, $matrix$ representa la matriz de disyunción previamente construida, $cols$ es un entero que representa el número de columnas en la matriz de disyunción y $cols = \binom{n}{2} - 1$ y $coveredEdges$ es un vector de enteros que indica cuáles aristas han sido cubiertas por alguna colección en determinado momento.

Procedemos a explicar las líneas de este algoritmo para entender mejor qué es lo que hace. En la línea 2 se verifica si el nivel actual en el árbol es mayor al mínimo encontrado hasta el momento. Si es así la llamada a la función termina. Esta verificación es útil cuando ya se encontró una descomposición de K_n de tamaño t, y el tamaño de la colección actual es mayor o igual a t. Ya que esa colección no mejorará el resultado actual de t, evitamos seguir por ese camino. La verificación de la línea 4 funciona para saber si la colección de thrackles ya cubre las aristas de la gráfica completa. Si la verificación es verdadera, quiere decir que se ha encontrado una descomposición. En la línea 5 se procede a evaluar si la descomposición es más pequeña que alguna encontrada anteriormente, si es así, se actualiza el valor de $minAt$ en la línea 6. Finalmente, en la línea 7, terminamos la llamada recursiva. El bloque `else` de la línea 8 se ejecuta cuando aún no se ha encontrado una descomposición. En la línea 9 declaramos un vector de enteros que nos servirá para almacenar

Algoritmo 8: Pseudcódigo del algoritmo que encuentra el anti-thickness de una gráfica completa inducida por un conjunto de puntos S.

```
1  función exhaustive_at (current_thrackle, LevelAt)
     Entrada: Un vector de enteros que representa las aristas de un
              thrackle, un entero que representa el tamaño de la
              colección de thrackles actual.
     Salida : El anti-thickness de la gráfica completa inducida por un
              conjunto de puntos.
2  if LevelAt ≥ minAt then
3      return
4  if coveredEdges.size() == cols then
5      if LevelAt < minAt then
6          minAt = LevelAt
7      return
8  else
9      Sea local_desc un vector de enteros.
10     while true do
11         Sea thrackle un vector de vector de enteros.
12         val ← next(local_desc, thrackle, current_thrackle, at)
13         if !val then
14             return
15         local_desc.push_back(thrackle)
16         coveredEdges ← coveredEdges ∪ thrackle
17         exhaustive_at(thrackle, LevelAt + 1)
18         coveredEdges ← coveredEdges ∩ thrackle
```

los hijos del nodo actual del árbol. El ciclo `while` de las líneas 10 a la 18 es en donde se hace la búsqueda de thrackles para la descomposición.

En la línea 11 declaramos un vector de enteros *thrackle*, en él, almacenaremos un thrackle compatible con el thrackle actual *current_thrackle*. En la línea 12 buscamos el thrackle compatible más cercano posible usando una función que busca, dada una colección de aristas, un thrackle compatible. Para nosotros, la noción de cercanía está basada en el ordenamiento de dos thrackles como mencionamos en la sección 2.1.2. Explicamos esta función más adelante. Si `next` encuentra un thrackle, lo almacena en la variable *thrackle* y val adquiere el valor de *true*, en caso contrario *val* tiene el valor de $false$. La línea 13 verifica si se encontró o no un thrackle compatible; si no se encontró, la llamada recursiva es terminada en la línea 14. La línea 15 agrega el vector *thrackle* a la lista de descendientes *local_desc*. Esto indica que *thrackle* es hijo de *current_thrackle* en el árbol. Esta información es usada por el la función `next`. La línea 16 actualiza la lista de aristas cubiertas hasta el momento. La linea 17 realiza la llamada recursiva para seguir bajando en el árbol y verificar si ya se ha cubierto a todas las aristas de K_n. La linea 18 quita de la lista de aristas cubiertas a las aristas cubiertas por *thrackle*. Esto permite realizar el backtracking en la siguiente iteración del ciclo `while`.

Para saber cuáles aristas han sido cubiertas por la colección de thrackles mantenemos registro de las aristas de cada thrackle haciendo la operación de unión: El vector *coveredEdges* contiene enteros desde 0 hasta $\binom{n}{2} - 1$. Cada vez que se trabaja con un thrackle compatible se hace la unión del vector *coveredEdges* con el contenido del vector que representa al thrackle.

En la figura 4.6 mostramos un ejemplo de un árbol que se forma cuando se ejecuta el algoritmo para la búsqueda del anti-thickness de algún dibujo de K_6, ilustramos un camino desde la raiz hasta un nodo T_3. En cada uno de los nodos se representa la información de los thrackles que conforman una descomposición de un dibujo de K_6. La gráfica K_6 tiene 15 aristas por lo que las etiquetas de las aristas de los thrackles van desde 0 hasta 15. En este ejemplo, una vez que se encontró la descomposición de tres thrackles T_1, T_2 y T_3 el valor de $minAt$ es igual a 3. Por ello el algoritmo evitará analizar colecciones de 3 o más thrackles. El algoritmo ignora estas colecciones al terminar prematuramente las llamadas recursivas en la línea 3 del pseudocódigo.

Una vez que se ha inicializado la matriz de disyunción podemos hacer la primer llamada recursiva a la función `exhaustive_at` usando como parámetros un vector de enteros vacío y $LevelAt = 0$.

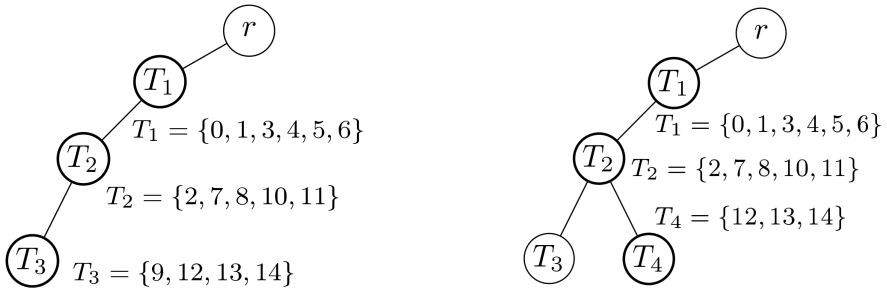


Figura 4.6: En la figura de la izquierda: una descomposición de algún dibujo de K_6 por tres thrackles T_1, T_2 y T_3. Cuando T_1 es analizado en el algoritmo, las aristas con etiquetas $0, 1, 3, 4, 5$ y 6 son agregadas al vector *coveredEdges* y después se hace la llamada recursiva con $current\_thrackle = T_1$ y $LevelAt = 1$, en esta llamada recursiva el algoritmo `next` encuentra a T_2 y lo agrega como parte de la colección al hacer la unión de sus aristas con el vector *coveredEdges*. Después se hace la llamada recursiva con $current\_thrackle = T_2$ y $LevelAt = 2$, en esta llamada recurisva el algoritmo `next` encuentra a T_3 y lo agrega como parte de la colección al hacer la unión de sus aristas con el vector *coveredEdges*. Después se hace la llamada recursiva y se detecta que todas las aristas han sido cubiertas por lo que se actualiza el valor de $minAt$ y se termina la última llamada recursiva. En la figura de la derecha: Después de terminar la llamada recursiva se descarta T_3 y `next` busca, a partir de T_2 el siguiente thrackle compatible. En este ejemplo, se encuentra a T_4 que es un thrackle con 3 aristas. En la colección de los tres thrackles T_1, T_2, T_4 falta la arista con etiqueta 9. El algoritmo intentaría buscar el próximo thrackle, de tamaño 1, que cubra esta arista en la siguiente llamada recursiva. Sin embargo, antes de buscar un thrackle comparará las variables $LevelAt$ y $minAt$. Como son iguales, evitará seguir buscando un thrackle y finalizará la llamada recursiva descartando así a T_4. Nótese que en este momento la instancia del vector $local\_desc$ tiene los siguientes elementos: $local\_desc = \{T_3, T_4\}$.

Algoritmo 9: Algoritmo para encontrar el siguiente thrackle compatible.

```
1  función next_at
     (current_thrackle, descendants, currentAt, found_thrackle)
     Entrada: El thrackle actual current_thrackle desde donde se quiere
              encontrar otro thrackle compatible, un vector de vectores
              de enteros que representa a los descendientes de
              current_thrackle, llamado descendants, el mejor tamaño
              actual de una descomposición de la gráfica completa
              actual currentAt y un vector vacío found_thrackle para
              almacenar el resultado. .
     Salida  : false si no se encontró un thrackle compatible; true si se
              encontró un thrackle compatible. El resultado es
              almacenado en found_thrackle.
2  Sea missing = {0, 1, 2, ..., (n choose 2) − 1} − coveredEdges.
3  q ← |missing|
4  if q > n and currentAt == 0 then
5  |  q ← n
6  end
7  if q > n and currentAt > 0 then
8  |  q ← n − 1
9  end
10 Sea s el tamaño del descendiente más pequeño i.
11 if  s < q and s > 0 then
12 |  q ← s
13 end
14 if  descendants ≠ ∅ then
15 |  starting_thrackle ←
   |    [descendants[i][0], descendants[i][1], ..., descendants[i][q − 1]]
16 end
17 else if current_thrackle es máximo o es vacío then
18 |  starting_thrackle ← [missing[0], missing[1], ..., missing[q − 1]]
19 end
20 else if current_thrackle no es máximo ni vacío y no hay
     descendientes then
21 |  starting_thrackle ← current_thrackle q ← |current_thrackle|
22 end
23 p ← 1
24 ce ← |coveredEdges|
25 while  qp + ce < cols do
26 |  p ← p + 1
27 end
28 if  currentAt + p ≥ minAt then
29 |  return false
30 end
```

```
32
33    Sea local un vector de enteros vacío.;
34    while true do
35        val ←find_next_thrackle(starting_thrackle, local);
36        while !val do
37            q ← q − 1;
38            if q == 0 then
39                return false
40            p ← 1;
41            ce ← |coveredEdges|;
42            while qp + ce < cols do
43                p ← p + 1
44            if currentAt + p ≥ minAt then
45                return false;
46            starting_thrackle ←
                [missing[0], missing[1], . . . , missing[q − 1]];
47        if coveredEdges ∩ local == ∅ then
48            if local ⊄ descendants then
49                found_thrackle ← local;
50                return true;
```

Ahora hablaremos acerca de la función `next` presentamos el pseudocódigo en el algoritmo 9. Esta función encuentra, dado un thrackle T de tamaño k otro thrackle compatible con T. Antes de empezar la búsqueda establece el tamaño q del thrackle a encontrar. Esto lo hace analizando el tamaño de los descendientes de T, el algoritmo toma el tamaño más pequeño de estos descedientes ya que, por la naturaleza del algoritmo, no tiene sentido buscar un thrackle más grande que el descendiente más pequeño. Una vez que se conoce el valor de q hay que elegir cuáles son las aristas en las que se iniciará la búsqueda del siguiente thrackle. Después se inicia una búsqueda secuencial y exhaustiva de un thrackle de tamaño q usando la misma técnica descrita en la sección 4.6.1 para encontrar un thrackle de un tamaño dado. La diferencia principal es la inicialización del vector C que en este caso no tendrá valores nulos sino los de las q etiquetas elegidas anteriormente.

Cuando la función `next` encuentra un thrackle M con q aristas se procede a verificar que la intersección del vector $coveredEdges$ y las etiquetas de M sea vacía. Esto es para asegurar que M sea disjunto con los otros thrackles de la colección.

Si la función `next` no encuentra un thrackle con q aristas, se repite la búsqueda de un thrackle con $q-1$ aristas. El tamaño mínimo de un thrackle es 1.

Si no se logra encontrar ningún thrackle compatible se retorna el valor de $false$. Si el thrackle encontrado es disjunto de los thrackles que ya están en la colección se retorna el valor de $true$.

Las primeras 30 líneas del pseudocódigo ilustran cómo se elige el tamaño del nuevo thrackle e inicializan el thrackle por donde se empezará la búsqueda. Las siguientes líneas describen el proceso de búsqueda de un thrackle compatible. La línea 35 llama al algoritmo de búsqueda que, como ya mencionamos, trabaja de la misma manera que el algoritmo para búsqueda de thrackles de tamaño k con la diferencia de que el vector inicial no está vacío sino que tiene los contenidos de $starting\_thrackle$.

El ciclo `while` de las líneas 34 a la 50 termina cuando sucede una de las tres situaciones siguientes

- El tamaño del thrackle compatible es 0. Esto indica que ya se ha intentado buscar thrackles compatibles con diferentes tamaños pero no se ha encontrado ninguno. Cada vez que el algoritmo no encuentra un thrackle compatible, se reduce el tamaño del thrackle a buscar en la línea 37.

- La comparación de la línea 44 es positiva. El segmento de código compuesto por las líneas 40 a la 45 hacen un cálculo para saber si vale la pena o no buscar un thrackle de tamaño q. Si se supone que en el mejor caso se necesitan p thrackles de tamaño q para cubrir a K_n entonces el tamaño final de la descomposición en ese caso en particular es $currentAt + p$. Si esto es mayor al tamaño más pequeño encontrado para una descomposición hasta el momento ($minAt$), entonces no vale la pena seguir por ese camino y se descarta esa búsqueda.

- Se ha encontrado un thrackle compatible, esto es que la intersección del thrackle encontrado y las aristas cubiertas hasta el momento es vacía y que además, el thrackle encontrado no es ninguno de los descendientes de *current_thrackle*. Esta verificación se hace en el `if` de las línea 47 a la 50.

Análisis de complejidad

Analizamos primero el costo de la función `exhaustive_at` descrita en el algoritmo 8. Las primeras 7 líneas tienen costo $O(1)$. Llamemos $O(\mathcal{XT})$ al costo de la línea 12, esto es el costo de la función `next`. El costo de agregar un elemento a un vector al final de este es constante por lo que la línea 15 tiene costo $O(1)$. Las operaciones de unión e intersección de conjuntos tienen costo lineal en el tamaño de los vectores de entrada, como el tamaño máximo del vector *coveredEdges* es $\binom{n}{2}$, las operaciones de las líneas 16 y 18 tienen costo $O(n^2)$. El costo total del algoritmo depende del número de veces que se llame recurisvamente a la función `exhaustive_at`. En el peor caso, en la primera llamada a la función tratamos de cubrir una gráfica con $\binom{n}{2}$ aristas y solo podemos cubrir una, es decir, encontramos un thrackle de tamaño uno, luego la llamada recursiva será para resolver un problema de $\binom{n}{2} - 1$ aristas y de nuevo, encontramos un thrackle de tamaño uno y así sucesivamente. Esto nos permite escribir una relación de recurrencia para la complejidad de este algoritmo. Sea $T(k)$ la complejidad, en el peor caso, del algoritmo `exhaustive_at` y $k = \binom{n}{2}$:

$$T(k) = T(k-1) + O(1) + O(k^2 \frac{n-1}{2}^2) + O(\mathcal{XT})$$

Podemos observar que como $k = \binom{n}{2}$ entonces $n = k\frac{n-1}{2}$. Esta relación de

recurrencia converge a

$$O(k^3n^2 + \mathcal{X}\mathcal{T} \cdot k) = O\left(\binom{n}{2}^3 n^2 + \binom{n}{2}\mathcal{X}\mathcal{T}\right) = O(n^8 + n^2\mathcal{X}\mathcal{T})$$

Ahora analizamos el costo de la función `next`. Hacemos esto contando cuántas veces se llamará a `next` desde un mismo nodo, para esto suponemos que el anti-thickness de la gráfica completa inducida por el dibujo es mayor que el nivel donde se encuentra el nodo. Dado un thrackle con m aristas donde $m < n$, debemos calcular cuántos thrackles son compatibles con él. Diagamos que solo se han cubierto m aristas, esto implica que quedan $\binom{n}{2} - m$ aristas por cubrir. En el peor caso hay $\binom{\binom{n}{2}-m}{m}$ thrackles de tamaño m compatibles, $\binom{\binom{n}{2}-m}{m-1}$ thrackles de tamaño $m-1$ compatibles, $\binom{\binom{n}{2}-m}{m-2}$ thrackles de tamaño $m-2$ compatibles y de esta misma manera hasta $\binom{\binom{n}{2}-m}{1}$ thrackles de tamaño 1 compatibles. Y cada uno será encontrado, no necesariamente de manera subsecuente, por la función `next` desde el mismo nodo. En total hay, en el peor caso

$$\sum_{i=1}^{m} \binom{\binom{n}{2} - m}{i}$$

thrackles compatibles. Como esa suma maximiza su valor cuando $i = m$, podemos acotarla por $O(n^{2m})$. Luego, el valor más grande que puede tener m es $n-1$ podemos acotar el costo de esta función por $O(n^{2(n-1)}) = O(n^{2n})$.

Finalmente el costo del algoritmo `exhaustive_at` es

$$O(n^8 + n^2n^{2n}) = O(n^8 + n^{2+2n}) = O(n^{2+2n}).$$

4.6.7. Algoritmo para dar el número de cruce de un thrackle máximo

Utilizamos este algoritmo para resolver el siguiente problema:
Este algoritmo fue diseñado para resolver el siguiente problema:

Problema: Dado un thrackle geométrico, calcular su número de cruce.

Entrada: Las aristas del thrackle representadas en un arreglo booleano.

Salida: El número de cruce del thrackle.

Calculamos el número de cruce de un thrackle como describimos en la sección 4.5. Mostramos el pseudocódigo del algoritmo para realizar los cálculos en el algoritmo 10.

Análisis de complejidad

Este algoritmo es secuencial. El ciclo de la línea 7, recorre un arreglo de tamaño $\binom{n}{2}$ lo que aporta $O(n^2)$ a la complejidad. Los ciclos anidados de las líneas 10 y 11 realizan $n+n-1+n-2+\cdots+1$ operaciones con complejidad $O(1)$, lo que aporta $O(n^2)$ a la complejidad del algoritmo. Finalmente, el ciclo de la línea 18 recorre un arreglo de tamaño n, lo que aporta $O(n)$ a la complejidad.

Así, la complejidad total del algoritmo es

$$O(n^2) + O(n^2) + O(n) = O(n^2)$$

.

Algoritmo 10: Algoritmo para calcular el número de cruce de un thrackle geométrico.

función CrossingNumberThrackle (E, n)
Entrada: Un arreglo booleano E, de tamaño $\binom{n}{2}$, que representa a las aristas del thrackle.
Salida : Un entero r igual al número de cruce del trackle inducido por E.

$i, j, c \leftarrow 0$
Sea $degs[n]$ un arreglo de enteros, de tamaño n
$th\_size \leftarrow 0$
$sum \leftarrow 0$
$r \leftarrow 0$
for $i = 0, \ldots, \binom{n}{2}$ **do**
  if $E[i] == 1$ **then**
    $th\_size \leftarrow th\_size + 1$
for $i = 0, \ldots, n$ **do**
  for $j = i + 1, \ldots, n$ **do**
    if $E[c]$ **then**
      $degs[i] \leftarrow degs[i] + 1$
      $degs[j] \leftarrow degs[j] + 1$
    $c \leftarrow c + 1$
$ans \leftarrow th\_size \cdot \frac{(th\_size-1)}{2}$
$sum \leftarrow 0$
for $i = 0, \ldots, n$ **do**
  $sum \leftarrow sum + degs[i] \cdot \frac{(degs[i]-1)}{2}$
$r \leftarrow ans - sum$
return r

Capítulo 5

Conclusiones y trabajo futuro

- Cómo caracterizamos los tipos de orden que no tienen thrackles máximos?
- Cómo caracterizamos los tipos de orden que tienen solo un thrackle máximo?

Apéndice A

Etiquetas de thrackles máximos

n	Tipo de Orden	Thrackles
8	2	36 32 29 25 2
		37 31 28 23 0
	54	32 29 26 16 2
9	12	101 96 93 68 33 2
	52	100 97 96 92 33 2
	54	82 78 75 52 33 2
	80	71 68 67 55 33 2
		72 68 67 55 33 2
		73 68 59 58 33 2
	696	79 76 73 40 22 2
	1080	37 33 32 29 15 2
	1287	57 55 44 35 24 3
		57 56 43 35 24 3
		57 55 50 35 24 3
		57 56 42 35 24 3
		57 56 44 35 24 3
		57 56 45 35 24 3
		57 56 48 35 24 3
		57 56 49 35 24 3
		57 56 50 35 24 3
		57 56 51 35 24 3
		60 56 39 35 24 3
		60 57 56 35 24 3
		61 56 39 35 24 3
		61 57 56 35 24 3
		62 55 39 35 24 3

n	Tipo de Orden	Thrackles
9	1287	62 56 39 35 24 3
		62 57 55 35 24 3
		62 57 56 35 24 3
		63 56 35 20 6 0
		63 56 35 20 7 6
		63 56 35 21 20 0
		63 56 35 22 20 0
		63 56 35 24 3 0
		63 56 35 24 4 0
		63 56 35 24 5 0
		63 56 35 24 6 0
		63 56 35 24 7 3
		63 56 35 24 7 4
		63 56 35 24 7 5
		63 56 35 24 7 6
		63 56 35 24 12 0
		63 56 35 24 13 0
		63 56 35 24 14 0
		63 56 35 24 15 0
		63 56 35 24 17 0
		63 56 35 24 18 0
		63 56 35 24 21 0
		63 56 35 24 22 0
		63 56 39 35 24 3
		63 57 56 35 24 3

n	Tipo de Orden	Thrackles
10	81	176 168 156 125 38 0
	1328	150 142 113 100 39 0
	2243	121 119 85 67 42 3
		128 115 82 51 39 0

Apéndice B

Números de cruce para thrackles máximos

n	Tipo de Orden	Thrackles	Números de Cruce
8	12	36 32 29 25 2	5 5 11 5 11
		37 31 28 23 0	11 5 5 11 5
	54	32 29 26 16 2	5 5 11 5 11
9	12	101 96 93 68 33 2	14 6 14 14 6 6
	52	100 97 96 92 33 2	14 6 14 6 6 6
	54	82 78 75 52 33 2	6 6 14 14 6 14
	80	71 68 67 55 33 2	14 6 15 6 6 14
		72 68 67 55 33 2	14 6 15 6 6 11
		73 68 59 58 33 2	6 6 17 6 6 14
	696	79 76 73 40 22 2	14 6 14 14 6 6
	1080	37 33 32 29 15 2	14 6 14 18 6 6
	1287	57 55 44 35 24 3	6 11 11 15 6 15
		57 56 43 35 24 3	6 11 14 15 6 15
		57 55 50 35 24 3	6 6 20 15 6 15
		57 56 42 35 24 3	6 6 17 15 6 15
		57 56 44 35 24 3	6 6 11 15 6 15
		57 56 45 35 24 3	6 6 21 15 6 15
		57 56 48 35 24 3	6 6 22 15 6 15
		57 56 49 35 24 3	6 6 20 15 6 15
		57 56 50 35 24 3	6 6 14 15 6 15
		57 56 51 35 24 3	6 6 22 15 6 15
		60 56 39 35 24 3	14 6 11 15 6 15
		60 57 56 35 24 3	14 6 11 15 6 15
		61 56 39 35 24 3	14 6 6 15 6 15
		61 57 56 35 24 3	11 6 11 15 6 15
		62 55 39 35 24 3	11 6 6 15 6 15

n	Tipo de Orden	Thrackles	Número de cruce
9	1287	62 56 39 35 24 3	6 6 11 15 6 15
		62 57 55 35 24 3	6 6 11 15 6 15
		62 57 56 35 24 3	6 6 6 15 6 15
		63 56 35 20 6 0	15 6 15 11 6 6
		63 56 35 20 7 6	15 6 15 11 11 6
		63 56 35 21 20 0	15 6 15 11 11 6
		63 56 35 22 20 0	15 6 15 14 11 6
		63 56 35 24 3 0	15 6 15 6 15 6
		63 56 35 24 4 0	15 6 15 6 14 6
		63 56 35 24 5 0	15 6 15 6 11 6
		63 56 35 24 6 0	15 6 15 6 6 6
		63 56 35 24 7 3	15 6 15 6 11 15
		63 56 35 24 7 4	15 6 15 6 11 14
		63 56 35 24 7 5	15 6 15 6 11 11
		63 56 35 24 7 6	15 6 15 6 11 6
		63 56 35 24 12 0	15 6 15 6 21 6
		63 56 35 24 13 0	15 6 15 6 22 6
		63 56 35 24 14 0	15 6 15 6 20 6
		63 56 35 24 15 0	15 6 15 6 22 6
		63 56 35 24 17 0	15 6 15 6 17 6
		63 56 35 24 18 0	15 6 15 6 20 6
		63 56 35 24 21 0	15 6 15 6 11 6
		63 56 35 24 22 0	15 6 15 6 14 6
		63 56 39 35 24 3	15 6 11 15 6 15
		63 57 56 35 24 3	15 6 6 15 6 15

n	Tipo de Orden	Thrackles	Número de cruce
10	81	176 168 156 125 38 0	19 7 19 7 19 7
	1328	150 142 113 100 39 0	19 7 19 7 19 7
	2243	121 119 85 67 42 3	7 19 7 19 7 19
		128 115 82 51 39 0	19 7 19 7 19 7

Bibliografía

Aichholzer, O., Aurenhammer, F., & Krasser, H. (2002). Enumerating order types for small point sets with applications. *Order*, *19*(3), 265–281.

Aichholzer, O., & Krasser, H. (2001). The point set order type data base: A collection of applications and results. In *CCCG*, vol. 1, (pp. 17–20).

Araujo, G., Dumitrescu, A., Hurtado, F., Noy, M., & Urrutia, J. (2005). On the chromatic number of some geometric type kneser graphs. *Comput. Geom.*, *32*(1), 59–69.

Chartrand, G., & Zhang, P. (2008). *Chromatic Graph Theory*. Chapman and Hall/CRC, 1 ed.

Dillencourt, M. B., Eppstein, D., & Hirschberg, D. S. (2004). Geometric thickness of complete graphs. In *Graph Algorithms And Applications 2*, (pp. 39–51). World Scientific.

Dujmovic, V., & Wood, D. R. (2017). Thickness and antithickness of graphs. *CoRR*, *abs/1708.04773*.

Fabila-Monroy, R., Hidalgo-Toscano, C., Leaños, J., & Lomelí-Haro, M. (2017). The chromatic number of the disjointness graph of the double chain. *arXiv preprint arXiv:1711.05425*.

Fabila-Monroy, R., Jonsson, J., Valtr, P., & Wood, D. R. (2018a). The exact chromatic number of the convex segment disjointness graph. *arXiv preprint arXiv:1804.01057*.

Fabila-Monroy, R., Jonsson, J., Valtr, P., & Wood, D. R. (2018b). The exact chromatic number of the convex segment disjointness graph. *arXiv preprint arXiv:1804.01057*, (p. 6).

Fulek, R., & Pach, J. (2011). A computational approach to conway's thrackle conjecture. *Computational Geometry*, *44*(6-7), 345–355.

Hurtado, F. (2009). Edge colouring geometric complete graphs. `http://www.openproblemgarden.org/op/edge_colouring_geometric_complete_graphs`.

Knuth, D. (2011a). *The art of computer programming:Volume 4A, Combinatorial Algorithms: Part 1, Fascicle 3A*. Addison-Wesley.
URL `http://www.cs.utsa.edu/~wagner/knuth/fasc3b.pdf`

Knuth, D. (2011b). *The art of computer programming:Volume 4A, Combinatorial Algorithms: Part 1, Fascicle 3B*. Addison-Wesley.
URL `http://www.cs.utsa.edu/~wagner/knuth/fasc3b.pdf`

Lara, D., & Rubio-Montiel, C. (2019). On crossing families of complete geometric graphs. *Acta Mathematica Hungarica*, *157*(2), 301–311.

Pach, J. (2013a). *The Beginnings of Geometric Graph Theory*, (pp. 465–484). Berlin, Heidelberg: Springer Berlin Heidelberg.
URL `https://doi.org/10.1007/978-3-642-39286-3_17`

Pach, J. (2013b). *The Beginnings of Geometric Graph Theory*, (p. 471). Berlin, Heidelberg: Springer Berlin Heidelberg.
URL `https://doi.org/10.1007/978-3-642-39286-3_17`

Pach, J., & Sterling, E. (2011). Conway's conjecture for monotone thrackles. *The American Mathematical Monthly*, *118*(6), 544–548.
URL `https://www.tandfonline.com/doi/abs/10.4169/amer.math.monthly.118.06.544`

Schaefer, M. (2018). *Crossing numbers of graphs*. Discrete Mathematics and Its Applications. CRC Press.