Miscelánea Matemática **80** (2025) 87-102 SMM

**DOI:** https://doi.org/10.47234/mm.8007

# Conectar al mundo con eficiencia

Ana Paulina Figueroa
Departamento Académico de Matemáticas
ITAM
ana.figueroa@itam.mx
y
Julián Fresán-Figueroa
Departamento de Matemáticas Aplicadas y Sistemas
UAM-Cuajimalpa
jfresan@cua.uam.mx

## 1. Introducción

En un mundo interconectado, diseñar conexiones óptimas que garanticen que los individuos estén conectados entre sí, es esencial. Este es el caso de un problema que ha captado la atención de matemáticos y científicos de la computación durante las últimas décadas: encontrar el árbol generador de peso mínimo, una forma de conectar puntos de manera efectiva y económica.

Los primeros artículos que se conocen sobre este tema fueron publicados en checo, por Otakar Borůvka [1, 2], en 1926. En ellos se planteaba el siguiente problema.

> Sean $n$ puntos en el plano (o en el espacio) cuyas distancias mutuas son diferentes. El problema consiste en unirlos mediante una red de manera que:
> 1. Cualquier par de puntos esté unido directamente o por medio de otros puntos.
> 2. La longitud total de la red sea la más pequeña posible.

Borůvka abordó este problema motivado por la necesidad de optimizar la construcción de redes eléctricas en el suroeste de Moravia en Checoslovaquia [1]. Los dos artículos fueron traducidos en 2001 por Nešetřil et al. [11] al inglés, por la importancia del método ahora conocido como el

*Palabras clave*: teoría de las gráficas, árbol de peso mínimo, árbol de peso mínimo con restricciones de grados, teoría de las gráficas, conexidad, algoritmos, heurísticas.

[1] El proyecto para electrificar el sur de Moravia fue necesario a principios del Siglo XX, por lo que los editores de la revista en que se publicó el artículo lo encontraban interesante.

algoritmo de Borůvka, que fue uno de los primeros en proporcionar una solución sistemática para este tipo de problemas. En 1930, Jarník [7] propuso una simplificación y una variante al método de Borůvka, que posteriormente influiría en el desarrollo del algoritmo de Prim, probablemente el más conocido y el que se expone en la mayoría de los temarios de los cursos de teoría de las gráficas y optimización entre otros. Este método se enfoca en la construcción incremental de una solución al problema, añadiendo la arista de menor peso disponible que conecta un nuevo punto a la solución en construcción. El problema del árbol de peso mínimo, en el lenguaje matemático actual, se plantea sobre una gráfica completa [2] en donde cada arista tiene un peso. Encontrar la forma de unir a los vértices de dicha gráfica completa [3], se traduce en encontrar una subgráfica generadora $H$ [4], conexa [5] y que cumpla que la suma de los pesos de sus aristas sea la menor posible. Un resultado de cualquier curso introductorio de teoría de las gráficas es que $H$ es una subgráfica conexa generadora de peso mínimo si y solo si $H$ es un árbol [6] generador de peso mínimo, es decir, una subgráfica conexa, sin ciclos y de peso mínimo. Desde la planificación de redes de comunicación hasta el procesamiento de imágenes digitales, el problema del árbol generador de peso mínimo se encuentra en el corazón de numerosos desafíos del mundo real [5, 12, 13]. En el ámbito de las redes de comunicación, por ejemplo, encontrar un árbol generador de peso mínimo puede ayudar a minimizar el costo de transmitir datos entre diferentes puntos de una red. En logística, puede utilizarse para planificar rutas de transporte eficientes, minimizando el tiempo o los costos asociados con el transporte de bienes. Al tener tantos problemas cuya solución necesita de encontrar un árbol de peso mínimo, se comenzaron a proponer una oleada de nuevos algoritmos para encontrarlo en el menor tiempo posible. Hemos decidido exponer a profundidad, en la segunda sección de este trabajo, el algoritmo de Borůvka y el algoritmo

[2]Una gráfica es un objeto formado por dos conjuntos. Al primero conjunto se le conoce como conjunto de vértices o nodos y al segundo como conjunto de aristas y está formado con parejas no ordenadas de vértices. Una forma de interpretar a las aristas, es como una relación simétrica en el conjunto de vértices. Cuando la pareja de vértices $uv$ forman una arista, decimos que u y v son adyacentes y que la arista $uv$ incide con cada uno de sus vértices extremos $u$ y $v$.

[3]Una gráfica es completa si cada par de vértices son adyacentes.

[4]$H$ es una subgráfica de $G$ si los vértices (respectivamente aristas) de $H$ son vértices (respectivamente aristas) de $G$. Una subgráfica generadora de $G$ es una subgráfica que tiene como vértices a todos los de $G$.

[5]Una trayectoria es una gráfica formada por una sucesión de vértices adyacentes cuyas aristas son las parejas de vértices consecutivos. Una gráfica es conexa si cada par de vértices está conectado por una trayectoria.

[6]Un árbol tiene muchas definiciones equivalentes. Lo más común en los libros de texto es definir a un árbol como una gráfica conexa y acíclica; pero también es una gráfica conexa que al quitarle cualquier arista se desconecta (conexa minimal); una gráfica conexa que al ponerle cualquier arista se forma un ciclo (aciclica maximal); una gráfica acíclica con una arista menos que los vértices, etc.

de Kruskal. El primero porque, como señala Nešetřil el al. [11], está en la médula de los algoritmos más eficientes que se conocen hasta el momento; y el segundo, el llamado algoritmo de Kruskal [9], porque es una buena ilustración de un algoritmo ávido (glotón). Los algoritmos ávidos son una familia de algoritmos muy estudiados en la optimización combinatoria y se caracterizan por tomar la mejor decisión en cada paso sin importar el comportamiento en el futuro. A pesar de que la estrategia de tomar en cada paso la mejor decisión disponible no siempre garantiza el óptimo (un mínimo o un máximo), en el caso del problema del árbol de peso mínimo, el algoritmo de Kruskal si obtiene un árbol de peso mínimo.

No todos los problemas aplicados en donde se necesitan conectar vértices con el menor peso posible se resuelven con el árbol de peso mínimo, pues los requerimientos particulares de una aplicación pueden provocar que la solución obtenida por el algoritmo sea inútil. Por ejemplo, si necesitamos conectar a los puntos usando un árbol específico, como puede ser una trayectoria, el asunto se complica. El requisito de que el árbol sea una trayectoria que pase por todos los vértices y que tenga peso mínimo es un problema clásico: es el problema de encontrar una trayectoria hamiltoniana de peso mínimo [7]. Este problema, que también esta ligado con el del agente viajero[8], no siempre se puede resolver en una gráfica en general. En la tercera sección de este artículo presentaremos una variante del problema menos conocida, el árbol de peso mínimo con restricción en grados. Esta variación del problema es terriblemente difícil de calcular, por lo que se han desarrollado heurísticas para llegar a soluciones parciales. En la última sección describiremos brevemente el concepto de heurística y lo ilustraremos con una heurística para encontrar el árbol de peso mínimo con restricciones basada en el comportamiento de las hormigas.

## 2. Los algoritmos de Borůvka y Kruskal

A pesar de que han surgido otros algoritmos más avanzados para encontrar árboles generadores de peso mínimo, como mencionamos antes, el algoritmo de Borůvka sigue siendo relevante y estudiado tanto por su importancia histórica como por su utilidad en ciertos casos, especialmente en gráficas con muchos vértices y grado máximo pequeño [9],

[7] Una trayectoria hamiltoniana de $G$ es una trayectoria por la que pasan todos los vértices de $G$. Saber si una gráfica en general tiene una trayectoria hamiltoniana es un problema NP-completo.

[8] Un agente debe recorrer todas las ciudades (vértices) sin repetir ciudad y regresando a su destino con el menor costo posible.

[9] El grado de un vértice es el número de aristas a las cuales ese vértice es incidente.

donde brilla por su eficiencia. A continuación presentamos una descripción detallada de cómo funciona.

### 2.1 Algoritmo de Borůvka

En cada paso del algoritmo, se construye una subgráfica generadora $T$, que en el paso final resultará un árbol generador de peso mínimo de la gráfica $G$ con la que comenzamos. Para ilustrar el algoritmo, consideraremos la gráfica $G$ de la figura 1.

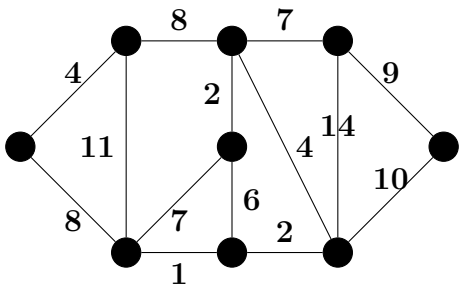


**Figura 1.** Grafica $G$.

Comenzaremos considerando a $T$ como la gráfica generadora sin aristas y por lo tanto cada vértice de la gráfica formará una componente conexa[10] de $T$ (figura 2).

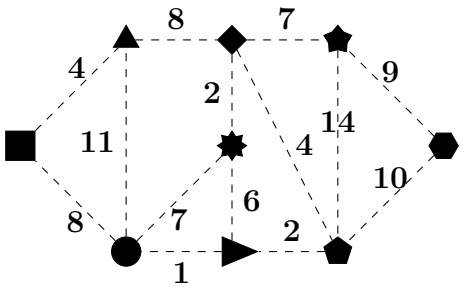


**Figura 2.** Componentes al inicio del paso 1.

Lo que sucede en este algoritmo es que en cada paso se disminuye el número de componentes conexas que tiene la subgráfica $T$. Como resultado, el algoritmo termina cuando $T$ tiene una sola componente conexa, es decir cuando $T$ es una subgráfica conexa de $G$. A continuación explicamos cómo se disminuye el número de componentes conexas. En el primer paso, para cada componente conexa (en este caso cada vértice)

[10]Una componente conexa de $G$ es una subgráfica conexa de $G$ y que es maximal con esa propiedad. De esta forma, las subgráficas conexas de $G$ forman una partición de los vértices de $G$.

tomaremos la arista de menor peso que la une con el resto de las componentes conexas. En el ejemplo, para la componente conexa formada por el vértice cuadrado hay dos opciones: una arista de peso 4 que la une con la componente del triángulo y una arista de peso 8 que la une con la componente del círculo. El algoritmo elige la arista de peso 4 que la une con la componente del triángulo. Hay tres opciones para elegir una arista que sale de la componente del vértice circular, con pesos 1, 7 y 8. En este caso hay que tomar la de peso 1. Cuando se han tomado todas las decisiones para elegir una arista entre una componente dada y las demás, se termina el primer paso del algoritmo y se obtiene, en nuestro ejemplo la subgráfica $T$ en la figura 3.

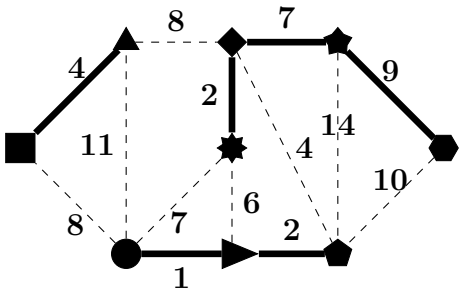


**Figura 3.** $T$ en el Paso 1.

Es sencillo ver que todas las aristas que se han seleccionado deberían de estar en un árbol de peso mínimo. Al inicio del paso dos se han creado nuevas componentes. En nuestro ejemplo la de cuadrados, la de estrellas y la de círculos (figura 4).

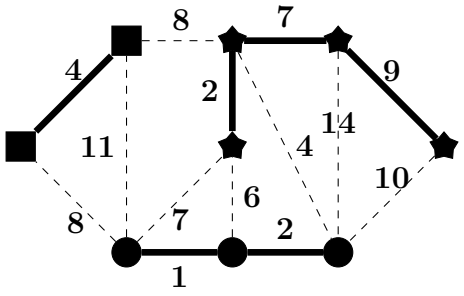


**Figura 4.** Nuevas componentes conexas al finalizar el Paso 1.

En el paso 2, se repite el proceso del paso 1 pero tomando en cuenta estas nuevas componentes. Como se observa en la figura 4, al procesar la componente de cuadrados seleccionamos una arista de menor peso. En nuestro caso seleccionamos la que corresponde es la arista de peso 8 que está entre la componente de cuadrados y la de estrellas (podríamos

tomar también la de peso 8 de la componente de cuadrados a la de círculos: cuando hay un empate, se puede decidir qué arista tomar con cualquier criterio). Al procesar la componente de estrellas, la arista de menor peso que la une con otra componente es la arista de peso 4, que está entre la componente de estrellas y la de círculos. Ya que se han seleccionado las aristas, se añaden a la subgráfica $T$ para formar la subgráfica $T$ del paso 2, como se observa en la figura 5.

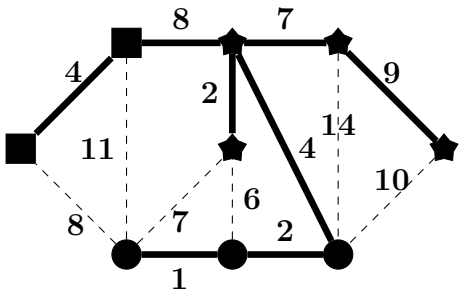


**Figura 5.** T del Paso 2.

Al final del paso 2, calculamos nuevamente las componentes conexas y se repite el proceso en caso de que $T$ tenga por lo menos dos componentes conexas. En cada paso en el que $T$ no es conexa, se disminuye la cantidad de componentes conexas agregando por lo menos una arista más entre dos componentes; pero cuando $T$ es conexa (tiene una componente conexa), termina el algoritmo. Como es el caso de $T$ en la figura 6 en donde hay una única componente de círculos.

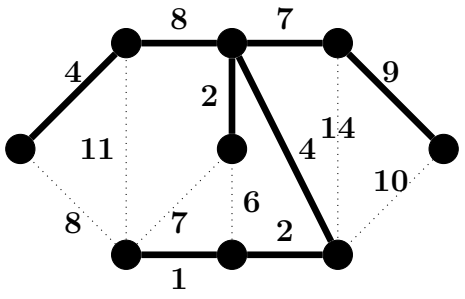


**Figura 6.** Árbol encontrado por el algoritmo de Borůvka.

Es claro que al disminuir las componentes conexas en cada paso, el algoritmo termina y se obtiene una subgráfica conexa $T$. Además $T$ no tiene ciclos porque solo puede haber una arista que se añade entre dos componentes en cada paso. Por lo que al final del algoritmo, $T$ es un árbol generador. Las aristas que hemos puesto hacen que el árbol $T$ que obtenemos al final sea de peso mínimo. La demostración se basa en que,

de lo contrario, existe un árbol, $T^*$, diferente de $T$, de peso mínimo; que entre todos los árboles de peso mínimo es el que se parece más a $T$ ($E(T) \cap E(T^*)$ es máxima) y que se distingue de $T$ en la etapa más tardía del algoritmo posible. Se usa la forma en que se eligen las aristas en cada paso, para demostrar que $T$ debe de pesar lo mismo o menos que $T^*$, por lo que $T$ es de peso mínimo.

El algoritmo de Borůvka es interesante en el contexto de la computación paralela debido a su naturaleza, que permite dividir el trabajo en subproblemas independientes que pueden resolverse en paralelo. Como en cada componente se elige la arista de peso mínimo de manera independiente, es posible que cada procesador maneje uno o varios componentes, buscando la arista de menor peso que conecte una componente con otra componente. De igual manera, la fusión de componentes se puede realizar en paralelo; los pares de componentes pueden ser fusionados de manera independiente y simultáneamente si no comparten aristas comunes. Por ello, el algoritmo de Borůvka en un algoritmo fácilmente paralelizable, lo que lo convierte en un algoritmo eficiente para encontrar árboles generadores de peso mínimo en gráficas muy grandes o en situaciones en las que hay que actualizar constantemente el árbol.

A continuación exponemos uno de los algoritmos clásicos que se desarrolla en un curso básico de teoría de las gráficas, el algoritmo de Kruskal, el cual tiene la particularidad de ser un algoritmo ávido.

### 2.2 Algoritmo de Kruskal

El algoritmo de Kruskal comienza ordenando todas las aristas de la gráfica de menor a mayor peso y como en el algoritmo de Borůvka, iremos construyendo una subgráfica $T$ generadora que al final será un árbol de peso mínimo. Para ilustrar el proceso usaremos la misma gráfica $G$ que en el algoritmo anterior (figura 1). Al principio se considera a $T$ como la subgráfica generadora con cero aristas como se aprecia en la figura 7.

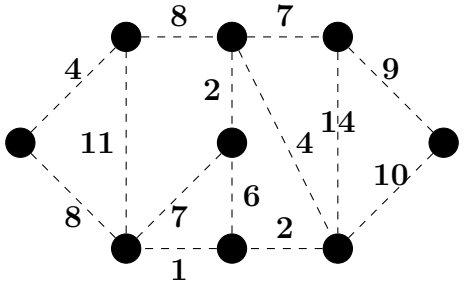


**Figura 7.** $T$ inicial.

En cada paso, a diferencia del algoritmo anterior, se decide una a una, según el orden creciente de pesos, si la arista correspondiente se añade o no a $T$. Una arista se añadirá a $T$ cuando no forme ciclos con las aristas de $T$. Por ejemplo, de la gráfica $G$ con la que comenzamos, en los primeros cinco pasos del algoritmo, se agregan a $T$ las aristas con pesos del 1 al 4, como en la figura 8.

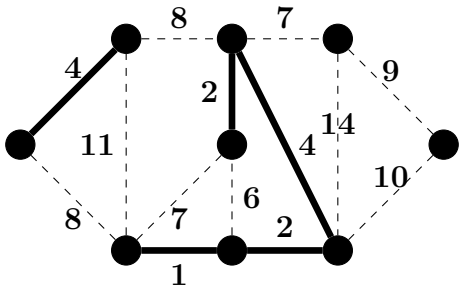


**Figura 8.** Primeros cinco pasos en el algoritmo.

En el ejemplo, la arista de peso 6, que es la que sigue en el orden creciente, no se puede añadir porque porque forma un ciclo con las aristas de $T$, como se puede observar en la figura 9. Algo similar sucede con una de las aristas de peso 7.

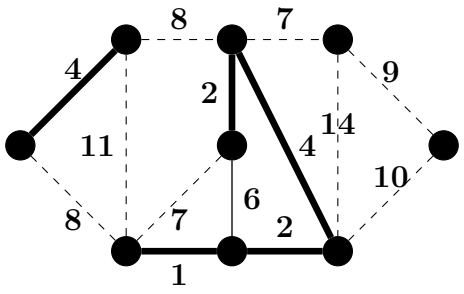


**Figura 9.** La arista de peso 6 no se puede añadir $T$.

Sin embargo, la otra arista de peso 7 puede añadirse a $T$ sin problemas. En este momento como se puede ver en la figura 10, hemos agregado siete aristas a $T$, por lo que estamos en el paso 8. En este paso, la arista candidata a ser agregada es la peso 8 que no pertenece a $T$. Pero no puede ser añadida porque $T$ dejaría de ser acíclica. De esta manera, en el paso 8 se debe de añadir la arista de peso 9, que es la de menor peso que no forma ciclos al ser añadida a $T$ (figura 10).

Verificar si una arista forma ciclos al añadirse a $T$ puede parecer complicado de implementar, sin embargo, una arista puede añadirse al $T$, sin formar ciclos si y solo si los vértices que conforman la arista están

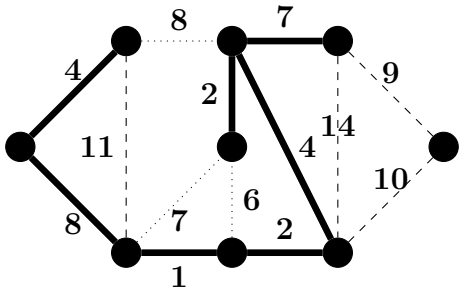


**Figura 10.** Paso 8.

en componentes conexas distintas. En el siguiente paso, las componentes que tienen a los vértices finales de la arista, se convierten en una sola componente. De forma que es muy sencillo guardar cuáles son las componentes en cada paso, y si las vamos guardando, la verificación de que una arista forme o no un ciclo es casi inmediata solo se trata de ver si sus extremos están o no en la misma componente conexa de $T$. En cualquier curso de teoría de las gráficas introductorio, se demuestra que $T$ es un árbol generador si y solamente si no tiene ciclos y tiene una arista menos que la cantidad de vértices de $G$. Supongamos que $n$ es el número de vértices de $G$. El algoritmo sigue evaluando añadir aristas a $T$ hasta que $T$ tenga $n-1$ aristas. Si esto no fuera posible, la gráfica tendría a lo más $n-2$ aristas, en cuyo caso $G$ no tendría un árbol de peso mínimo.

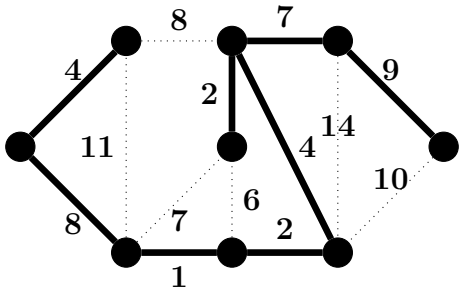


**Figura 11.** Árbol que se obtiene con el algoritmo de Kruskal.

Una vez que se han procesado todas las aristas o se han añadido $n-1$ aristas, el algoritmo termina encontrando de esta forma un árbol generador de peso mínimo en la gráfica (figura 11).

Observemos que los árboles que hemos encontrado con el algoritmo de Borůvka (figura 6) y el de Kruskal (figura 11) pesan lo mismo pero son distintos; lo que nos lleva a concluir que una gráfica puede tener más de un árbol de peso mínimo. Se puede demostrar que una condición suficiente para que el árbol generador de peso mínimo sea único es que

todos los pesos de sus aristas sean distintos. Una ventaja del algoritmo de Kruskal es que es particularmente eficiente cuando la gráfica es dispersa, es decir, cuando el número de aristas es relativamente pequeño en comparación con el número de vértices. Esto se debe a que la parte que mas lenta del algoritmo de Kruskal es el ordenamiento de aristas; si hay pocas aristas este algoritmo será muy rápido ya que no requiere de explorar todas las aristas de la gráfica. A pesar de que se podría pensar que las gráficas dispersas no son muy interesantes, existen muchas gráficas, como las redes de transporte entre muchas otras, que lo son.

## 3. Árboles con restricciones de grados

En una red de comunicación, como puede ser una red de *routers*, se busca transmitir datos de manera eficiente y confiable entre los dispositivos conectados. Entonces podemos plantear una gráfica en la que los vértices son los *routers* y el peso en las aristas refleja un retraso en la transmisión. En este problema, la manera más económica de que todos los *routers* puedan transmitir información entre ellos sin necesidad de mantener muchos canales de comunicación es encontrar un árbol de peso mínimo. Sin embargo, para solucionar este problema, son muy importantes las limitaciones de capacidad, ancho de banda, energía o recursos de procesamiento en cada *router*, dado que cada *router* puede tener una capacidad limitada para manejar conexiones simultáneas debido a restricciones de hardware o software. Esto se traduce en que se debe elegir una cantidad de aristas que salen de un vértice en función de su capacidad. Encontrar un árbol generador de peso mínimo entre todos los árboles que respetan las restricciones en los grados[11] de los vértices permite que la solución sea factible en términos prácticos pero que se siga optimizando el costo y la eficiencia, asegurando que se utilicen los recursos disponibles de manera óptima y evitando la congestión en los vértices. Es importante notar que estos árboles pueden no tener el mismo peso que un árbol de peso mínimo al que no se le pide ninguna restricción. Estas restricciones se interpretan como las desigualdades $d(x) \leq b(x)$, donde $b(x)$ es el máximo de conexiones que el vértice $x$ puede tener en un árbol generador.

Por otro lado, para las redes de alta demanda, es importante distribuir equitativamente la carga de tráfico entre las diferentes aristas y vértices para evitar congestiones y maximizar la eficiencia del sistema. Al tener múltiples aristas conectadas a cada vértice, se facilita la distribución de la carga de tráfico y se evita la congestión en *routers* individuales, lo que mejora el rendimiento general de la red. Esto se

[11] El número de aristas que salen de un vértice $v$ es el grado de $v$, denotado como $d(v)$.

refleja en que necesitaremos una restricción $a(x) \leq d(x)$, donde $a(x)$ es el mínimo de conexiones que debe de tener un vértice para que la carga se distribuya de manera efectiva. Notemos que en las soluciones para el árbol de peso mínimo dadas por los algoritmos como Kruskal y Borůvka no hay control sobre el grado debe tener cada vértice $x$ en un árbol de peso mínimo $T$ (como se puede observar en los árboles que hemos obtenido en la sección anterior en las figuras 6 y 11; un mismo vértice puede tener grados distintos en cada uno de esos árboles). Sin embargo hay problemas en los que, de manera natural, el grado de algún vértice $x$ en el árbol de peso mínimo necesita cumplir ciertas restricciones ($a(x) \leq d(x) \leq b(x)$).

Entonces el problema de encontrar un árbol de peso mínimo con restricciones en los grados, se plantea de la siguiente manera:

> **Problema de peso mínimo con restricciones en los grados:** Dada una gráfica $G$ donde cada arista $xy$ tiene un costo (asociado, usualmente a longitud o tiempo) $c_{xy}$, deseamos encontrar un árbol generador con el menor peso posible que satisfaga, $a(x) \leq d(x) \leq b(x)$ para cada vértice $x$ de $G$.

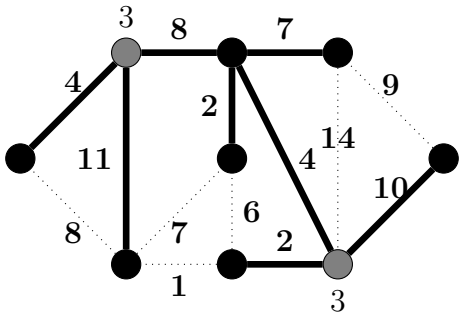


**Figura 12.** Árbol de peso mínimo con restricciones en los grados.

Consideremos nuevamente la gráfica $G$ de la figura 1. Si, como se ve en la figura 12, limitamos a los vértices grises a que tengan grado tres en el árbol que esperamos, la solución obtenida por un algoritmo que resuelva el problema en general, no necesariamente satisface las condiciones de grado (como se puede ver en las figuras 6 y 11 donde los vértices grises tienen grados uno o dos). Esto nos lleva a observar que los árboles generadores de peso mínimo con restricciones en los grados pueden tener un peso distinto a los árboles que no consideran esas restricciones. En la figura 12, podemos observar un árbol de peso mínimo cuyo peso es claramente mayor al de los árboles generadores de peso mínimo de $G$. De hecho, se puede demostrar que la diferencia entre el peso del árbol sin restricciones y con restricciones puede ser arbitrariamente grande.

Este problema restringido se planteo por primera vez en un artículo de Narula y Ho en 1980 [10]. En dicho artículo, los autores motivaron el problema con una red eléctrica y señalaron que encontrar un árbol de peso mínimo con restricciones en los grados tiene múltiples aplicaciones en otro tipo de problemas como en redes de transporte, comunicación, drenaje y manejo de aguas residuales. Modelaron también la búsqueda de este tipo de árboles como un problema de programación lineal entera y exploraron algunas técnicas para intentar buscar un algoritmo eficiente que lo solucionara. Sin embargo, la búsqueda de árboles de peso mínimo es esencialmente distinta a la búsqueda de árboles de peso mínimo con restricciones en los grados. Mientras que existen algoritmos eficientes que encuentran el árbol de peso mínimo [12], como Kruskal y Borůvka o Prim, el problema de encontrar un árbol de peso mínimo con restricciones en los grados es $NP$-completo[13] (no se tienen algoritmos eficientes y la esperanza de tenerlos es muy pequeña).

## 4. Heurísticas para el árbol de peso mínimo con restricciones en los grados

Cuando no se pueden encontrar algoritmos eficientes para solucionar un problema computacional se diseñan algoritmos heurísticos (también llamados heurísticas), que son una clase de algoritmos de búsqueda no exhaustiva y que desempeñan un papel fundamental en la resolución eficiente de problemas complejos en áreas como inteligencia artificial, optimización combinatoria y computación evolutiva. A diferencia de los algoritmos exactos, que garantizan una solución óptima, los algoritmos heurísticos adoptan un enfoque pragmático al explorar el espacio de búsqueda [14](en nuestro ejemplo, el espacio de búsqueda son todos los posibles árboles generadores en los que se satisfacen las restricciones de grado sin importar su peso) para encontrar una solución que sin ser necesariamente la óptima se calcule rápido y sea lo suficientemente buena en la práctica. Las heurísticas han tenido tantas aplicaciones que se han ido clasificando en tipos, como pueden ser las heurísticas de búsqueda local, las de búsqueda aleatoria, las de búsqueda tabú, etcétera. Incluso hay heurísticas basadas en comportamientos naturales como los algoritmos evolutivos, que están inspiradas en la teoría de la evolución, en

[12]La complejidad computacional de los algoritmos para encontrar el árbol de peso mínimo es $mlogn$ donde $m$ es el número de aristas de la gráfica y $n$ el número de vértices.

[13]Una prueba de que este problema es NP-completo se puede encontrar en el famoso libro de Garey, M, R. y Johnson D.S. [6] , que fue el primer libro escrito en el área de intratabilidad y problemas NP-completos y que contenía como anexo un compendio de los problemas NP-completos y sus reducciones conocidas hasta ese momento.

[14]Se le llama espacio de búsqueda al conjunto de todas las posibles soluciones aunque no sean óptimas.

donde se toman como población a algunas instancias del espacio de búsqueda y se obtienen nuevas poblaciones por selección, recombinación y mutación, generando así nuevas soluciones. Otras heurísticas se han diseñado inspiradas en la forma de actuar de grupos de animales como los enjambres de abejas o las colonias de hormigas.

Se han desarrollado una gran cantidad de heurísticas que intentan resolver de la mejor forma posible el problema del árbol generador de peso mínimo con restricciones en los grados [8]. A continuación exponemos una de dichas heurística, basada en el comportamiento de las hormigas y publicada en 2006 por Bui y Zrncic [4] para resolver un caso particular del problema de árboles de peso mínimo con restricciones en los grados. Esta heurística fue mejorada posteriormente en el 2011 por los mismos autores [3] pero la primera es más sencilla y probablemente ilustra mejor a las heurísticas de este estilo. Se comienza el algoritmo con una gráfica completa $G = (V, E)$ cuyas aristas tienen un peso no negativo y un entero $d \geq 2$. La salida deseable es un árbol de peso mínimo que cumpla que cualquier vértice tiene grado a lo más $d$. A pesar de que esta heurística no resuelve el caso general en el que todo vértice puede tener una restricción diferente, ni considera restricciones inferiores distintas a uno, el problema que plantea resolver sigue siendo $NP$-completo, como señalan sus autores.

### 4.1 Descripción del algoritmo basado en colonias de hormigas

El algoritmo se basa en el comportamiento de las colonias de hormigas para realizar trabajos colectivos como la creación de nidos o la búsqueda de alimentos y consta de una serie de fases. Cada fase tiene dos etapas: exploración y construcción. En la etapa de exploración de cada fase se utilizan hormigas para descubrir un conjunto de aristas candidatas a partir del cual, en la etapa de construcción, se construirá un árbol que cumpla con la restricción en los grados, tratando de usar dichas aristas. De esta manera, en cada fase se construye un árbol que cumple con las restricciones de grado. El árbol que encuentra el algoritmo es el que tenga menor costo entre todos los árboles que se construyeron en las distintas fases.

*Inicio.* El algoritmo inicia colocando una hormiga en cada vértice de la gráfica. Estas hormigas se irán moviendo por la gráfica de forma que cada vértice a lo largo del algoritmo puede o no tener hormigas y puede haber vértices que tengan varias hormigas. Por otra parte, a cada arista $uv$ de la gráfica se le asigna un nivel de feromonas $f(uv)$ de forma que las aristas de menor costo tengan el mayor nivel de feromonas.

*Exploración.* En la etapa de exploración las hormigas recorrerán las aristas de la gráfica. La decisión de si una hormiga debe pasar por una

arista incidente al vértice donde está parada dependerá del peso de la arista y de la información recopilada durante los recorridos realizados por otras hormigas. Para lograr esto, cada hormiga deja un rastro de feromonas cuando pasa por una arista que se añade al nivel de feromonas que ya tenía la arista. Este rastro de feromonas será mayor cuando el peso de la arista sea menor. Entonces la hormiga parada en un vértice decidirá al azar qué arista incidente a ese vértice debe recorrer cumpliéndose que, en la heurística, las aristas que tienen mayor nivel de feromonas tendrán una mayor probabilidad de ser recorridas.

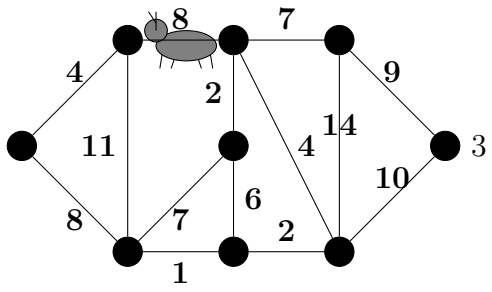


**Figura 13.** Las hormigas recorren las aristas en la etapa de exploración, eligiendo la arista que recorrerán con base en el peso y el nivel de feromonas de cada arista.

Por consiguiente, en cada paso de la etapa de exploración las hormigas se mueven a un vértice nuevo. Cuando llegan a un vértice que ya repitieron en la etapa, seleccionan una arista distinta a la que habían seleccionado antes; si no pueden hacerlo en 5 intentos entonces se quedan sin moverse.

Al final de la fase de exploración se actualizan los niveles de feromonas en cada arista, sumando la cantidad de feromona depositada por todas las hormigas durante la fase, pero teniendo en cuenta una disminución gradual de la feromona; con esto se evita que los niveles de feromonas en cada arista se vuelvan excesivamente altos para impedir sesgos no deseados.

*Construcción del árbol.* Cuando las hormigas terminan de moverse y las feromonas se han recalculado se seleccionan las primeras $n$ aristas con mayor nivel de feromonas y se ordenan de acuerdo a su peso. Con esto se forma una lista de aristas candidatas $C$. Se comienza a construir un árbol con un algoritmo muy parecido a Kruskal, añadiendo aristas del nivel de costo más bajo en $C$ al nivel más alto, de forma que no se formen ciclos y se respete el grado de cada vértice. Es probable que con las aristas candidatas $C$ no se complete un árbol. En este caso todas las demás aristas de la gráfica se ordenan por peso, de menor a mayor, y se van añadiendo para encontrar un árbol con las restricciones en los

grados. De esta manera el árbol construido es el árbol de la fase. Se compara su peso con los demás árboles construidos en otras fases y se conserva el que tenga el menor peso hasta el momento.

*Inicialización de feromonas de un nuevo ciclo.* Para comenzar con un nuevo ciclo de exploración y construcción se determinan nuevos niveles de feromonas de las aristas que ya no son los iniciales. Se les añadirá un factor de mejora, sobre todo a las aristas que están en el árbol de menor peso hasta el momento descubierto. Este factor de mejora se va acentuando cuando han pasado más ciclos del algoritmo porque al principio del algoritmo se espera que los árboles no sean tan buenos como al final, por lo que las aristas de los mejores árboles en las últimas fases deberían de ser fortalecidas con el factor de mejora. Sin embargo, para que el algoritmo no se estanque en un mínimo local, los niveles de feromona serán disminuidos cuando no se mejora el árbol en un cierto número dado de fases. El número de fases que se llevarán a cabo se decide experimentalmente. El árbol encontrado será el árbol de menor peso entre todos los árboles que se han encontrado en todas las fases que, por construcción cumple con los requisitos de los grados pero no necesariamente es óptimo.

Este algoritmo ilustra muy bien el sabor que tienen muchas de las heurísticas bioinspiradas para resolver problemas de gran dificultad computacional, como el de encontrar árboles de peso mínimo con restricciones en los grados. Es importante mencionar que no hay una única manera de diseñar un algoritmo heurístico inspirado en hormigas, se podrían hacer muchas modificaciones con las mismas ideas. El problema del árbol de peso mínimo con restricciones en los grados es solo una de las tantas variaciones del árbol de peso mínimo que surgen de la necesidad de aplicarlo. Revela que al añadir restricciones, surgidas desde las aplicaciones, los problemas generalmente se vuelven más difíciles de resolver, por lo que el uso de heurísticas se ha convertido en una herramienta de todos los días para atacar este tipo de problemas.

## Bibliografía

[1] O. Borůvka, «O jistém problému minimálním», *Práce Moravské přírodovědecké společnosti*, 1926, 37–58.

[2] ———, «Příspěvek k otázce ekonomické stavby elektrovodnỳch sítí», *Elektrotechnický obzor*, 1926, 153–154.

[3] T. N. Bui, X. Deng y C. M. Zrncic, «An improved ant-based algorithm for the degree-constrained minimum spanning tree problem», *IEEE Transactions on evolutionary Computation*, vol. 16, núm. 2, 2011, 266–278.

[4] T. N. Bui y C. M. Zrncic, «An ant-based algorithm for finding degree-constrained minimum spanning tree», en *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006, 11–18.

[5] P. F. Felzenszwalb y D. P. Huttenlocher, «Efficient graph-based image segmentation», *International journal of computer vision*, vol. 59, 2004, 167–181.
[6] M. R. Garey y D. S. Johnson, *Computers and intractability*, vol. 174, freeman San Francisco, 1979.
[7] V. Jarník, «O jistém problému minimálním.(Z dopisu panu O. Borůvkovi)», *Práce Moravské Přírodovědecké Společnosti*, 1930, 57–63.
[8] M. Krishnamoorthy, A. T. Ernst y Y. M. Sharaiha, «Comparison of algorithms for the degree constrained minimum spanning tree», *Journal of heuristics*, vol. 7, 2001, 587–611.
[9] J. B. Kruskal, «On the shortest spanning subtree of a graph and the traveling salesman problem», *Proceedings of the American Mathematical society*, vol. 7, núm. 1, 1956, 48–50.
[10] S. C. Narula y C. A. Ho, «Degree-constrained minimum spanning tree», *Computers & Operations Research*, vol. 7, núm. 4, 1980, 239–249.
[11] J. Nešetřil, E. Milková y H. Nešetřilová, «Otakar Borůvka on minimum spanning tree problem Translation of both the 1926 papers, comments, history», *Discrete mathematics*, vol. 233, núm. 1-3, 2001, 3–36.
[12] E. Tapia y R. Rojas, «Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance», en *International workshop on graphics recognition*, Springer, 2003, 329–340.
[13] C. T. Zahn, «Graph-theoretical methods for detecting and describing gestalt clusters», *IEEE Transactions on computers*, vol. 100, núm. 1, 1971, 68–86.