UIMP
Universidad Internacional
Menéndez Pelayo

Formalización del problema de la partición del grafo

Laura Rodríguez Navas
rodrigueznavas@posgrado.uimp.es

Junio 2020
Resolución de problemas con metaheurísticos

Resumen

Dentro del amplio campo de la Teoría de Grafos, en este trabajo se considera el problema de la partición del grafo: dado un grafo no dirigido cuyo número de vértices es par y cuyas aristas tienen asociado un peso, se trata de encontrar la partición del conjunto de vértices en dos subconjuntos de tamaño similar de manera que se minimice la suma de los pesos de aquellas aristas que unen vértices que están en diferentes conjuntos

En el capítulo inicial se presenta una breve introducción al problema y algunas nociones básicas. En el segundo capítulo, se proponen tres codificaciones para este problema, señalando sus aspectos más relevantes (positivos y negativos), además de ejemplos. Concretamente, las codificaciones son implementaciones en Python 3 de los algoritmos basados en técnicas metaheurísticas: Kernighan-Lin, Spectral Bisection y Multilevel Spectral Bisection. En el tercer capítulo, se concluye con una comparativa entre las diferentes codificaciones. Finalmente, en el último capítulo se presentan algunas conclusiones.

Las codificaciones se encuentran en el repositorio:

`https://github.com/lrodrin/masterAI/tree/master/A2`

Tabla de Contenidos

1. Introducción a la Partición de Grafos

1.1 Introducción

En los últimos años el problema de la partición de grafos ha sido ampliamente estudiado. El problema se deriva de situaciones del mundo real por lo que tiene aplicación en varias ramas de la ingeniería y de la investigación. La partición de grafos es una disciplina ubicada entre las ciencias computacionales y la matemática aplicada. El problema de la partición de grafos es un problema importante que tiene aplicaciones extensas en muchas áreas ya que se busca minimizar los costes o maximizar los beneficios a través de la correcta distribución de los recursos.

La primera aplicación del problema fue en la partición de componentes de un circuito en un conjunto de tarjetas que juntas realizaban tareas en un dispositivo electrónico. Las tarjetas tenían un tamaño limitado, de tal manera que el dispositivo no llegara a ser muy grande, y el número de elementos de cada tarjeta estaba restringido. Si el circuito era demasiado grande podía ser dividido en varias tarjetas las cuales estaban interconectadas, sin embargo, el coste de la interconexión era muy elevado por lo que el número de interconexiones debía ser minimizado.

La aplicación descrita fue presentada en [1], en la cual se define un algoritmo eficiente para encontrar buenas soluciones. En la aplicación, el circuito es asociado a un grafo y las tarjetas como subconjuntos de una partición. Los nodos del grafo son representados por los componentes electrónicos y las aristas forman las interconexiones entre los componentes y las tarjetas.

1.2 Descripción el problema

El problema de partición de grafos puede formularse como un problema de programación lineal entera (ver Definición 1).

Definición 1. *La programación lineal entera sirve para resolver los problemas de programación lineal en la que las variables tienen que tomar valores enteros.*

Los problemas de programación lineal entera generalmente están relacionados directamente a problemas combinatorios, esto genera que al momento de resolver los problemas de programación lineal entera se encuentren restricciones dado el coste computacional de resolverlos; por ese motivo se han desarrollado algoritmos que buscan soluciones de una forma más directa, la restricción de los mismos es que no se puede garantizar que la solución encontrada sea la óptima.

El problema de partición de grafos ha sido denominado como un problema NP-complete[2], lo que implica que las soluciones para él no pueden ser encontradas en tiempos razonables. Entonces, en lugar de encontrar la solución óptima para el problema, recurriremos a algoritmos que pueden no ofrecer la solución óptima pero que dan una buena solución al menos la mayor parte del tiempo.

2. Algoritmos basados en metaheurísticas

Debido a la *"intratabilidad"* del problema de la partición de un grafo, no existe un algoritmo concreto que permita obtener en tiempo polinómico una solución óptima a la partición de cualquier grafo. Es por ello por lo que, los algoritmos codificados para este trabajo se basan en la metaheurística, con el objetivo de obtener soluciones de buena calidad en tiempos computacionales aceptables, a pesar de que los algoritmos metaheurísticos no garantizan que se vaya a obtener una solución óptima al problema.

En el siguiente apartado se analizan los algoritmos Kernighan-Lin[1] (ver apartado 2.1), Specrtal Bisection (ver apartado 2.2) y Multilevel Spectral Bisection (ver apartado 2.3), tres algoritmos diseñados específicamente para la resolución del problema de la partición de grafos. Cualquiera de estos algoritmos proporciona una solución factible al problema, pudiendo ser esta óptima o no. Además de describir cada uno de los algoritmos detalladamente, las principales propiedades de cada uno de ellos, también se describirán algunos detalles sobre su codificación.

Para cada algoritmo analizado, contamos con un ejemplo de su codificación en el cual se ha utilizado un grafo inicial (ver Figura 2.1). El grafo inicial es un grafo no dirigido (ver Definición 2) de diez nodos con pesos en sus aristas.

Definición 2. *Un grafo no dirigido o grafo propiamente dicho es un grafo* $G = (V, E)$ *donde:* $V \neq \emptyset$ *y* $E \subseteq \{x \in \mathcal{P}(V) : |x| = 2\}$ *es un conjunto de pares no ordenados de elementos de* V*. Un par no ordenado es un conjunto de la forma* $\{a\ ,\ b\}$*, de manera que* $\{a,\ b\} = \{b,\ a\}$*. Para los grafos, estos conjuntos pertenecen al conjunto potencia de* V*, denotado* $\mathcal{P}(V)$*, y son de cardinalidad 2.*

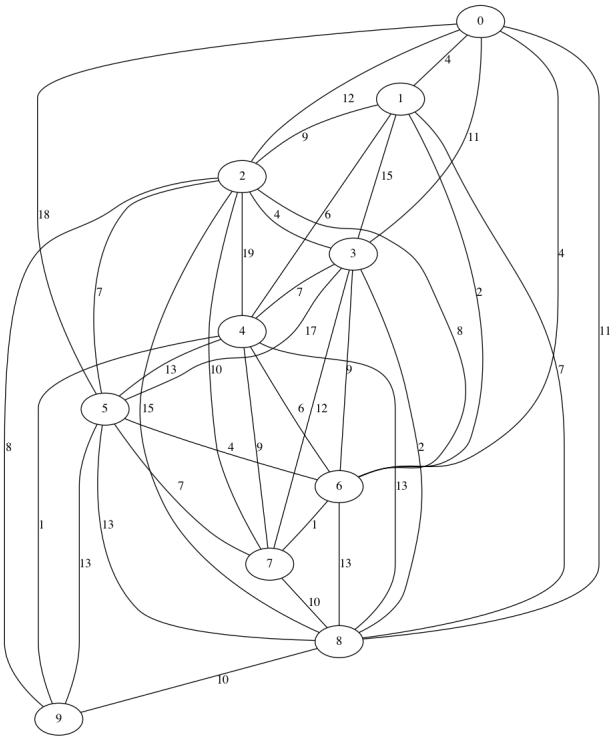

Figura 2.1: Grafo inicial.

2.1 Kernighan-Lin

El algoritmo Kernighan-Lin[1], a menudo abreviado como K-L, es uno de los primeros algoritmos de partición de grafos y fue desarrollado originalmente para optimizar la colocación de circuitos electrónicos en tarjetas de circuito impreso para minimizar el número de conexiones entre las tarjetas (circuitos VLSI[1][3]).

Las características principales del algoritmo es que es:

- **Iterativo**. Esto significa que el grafo inicialmente ya está particionado, pero la aplicación del algoritmo intentará mejorar u optimizar la partición inicial.
- **Voraz**. Esto significa que el algoritmo hará cambios si hay un beneficio inmediato sin considerar otras formas posibles de obtener una solución óptima. Para ello, elige la opción óptima en cada paso local con la esperanza de llegar a una solución general óptima.
- **Determinista** porque se obtendrá el mismo resultado cada vez que se aplique el algoritmo.

El algoritmo K-L no crea particiones, sino que las mejora iterativamente. La idea original era tomar una partición aleatoria y aplicarle Kernighan-Lin[1]. Esto se repetiría varias veces y se elegiría el mejor resultado. Mientras que para grafos pequeños esto ofrece resultados razonables, es bastante ineficiente para tamaños de grafos más grandes.

Hoy en día, el algoritmo se usa para mejorar las particiones encontradas por otros algoritmos, tratando de mejorar las particiones mediante el intercambio de nodos vecinos. Por lo tanto, complementa muy bien otro tipo de algoritmos como por ejemplo que realizan una partición espectral (ver sección 2.2).

2.1.1 Descripción

El objetivo del algoritmo es dividir el conjunto de vértices V de un grafo no dirigido en dos subconjuntos disjuntos (ver Definición 3) A y B de igual tamaño, de una manera que minimice la suma T de los pesos del subconjunto de aristas que cruzan de A a B. El algoritmo mantiene y mejora una partición, en cada pasada usando un algoritmo voraz (ver Definición 4), empareja los vértices de A con los vértices de B, de modo que al mover los vértices emparejados de una partición a la otra se maximiza la ganancia. Después de emparejar los vértices y maximizar la ganancia, realiza un subconjunto de los pares de los vértices elegidos para tener el mejor efecto sobre la calidad de la solución T. Dado un grafo con n vértices, cada paso del algoritmo se ejecuta en el tiempo $O(n^2 \log n)$.

Definición 3. *A y B son dos subconjuntos disjuntos si $A \cup B$ y $A \cap B \neq 0$.*

Definición 4. *Dado un conjunto finito C, un algoritmo voraz devuelve un conjunto S tal que $S \subseteq C$ y que además cumple con las restricciones del problema inicial. A cada conjunto S que satisfaga las restricciones y que logra que la función objetivo se minimice o maximice (según corresponda) diremos que S es una solución óptima.*

Más detalladamente, para cada $a \in A$, I_a es el coste interno de a, es decir, la suma de los pesos de las aristas entre a y otros nodos en A, y E_a es el coste externo de a, es decir, la suma de los pesos de las aristas entre a y los nodos en B. De manera similar, se define I_b y E_b para cada $b \in B$.

Por otra parte, la diferencia entre la suma de los costes externos e internos s es:

$$D_s = E_s - I_s$$

Entonces si a y b se intercambian, la ganancia es:

$$T_{antiguo} - T_{nuevo} = D_a + D_b - 2c_{a,b}$$

donde $c_{a,b}$ es el coste de las aristas posibles entre a y b.

Así, el algoritmo intenta encontrar una solución óptima de operaciones de intercambio entre los elementos de A y B que maximice la ganancia ($T_{antiguo} - T_{nuevo}$) y luego ejecuta las operaciones, produciendo una partición del grafo en A y B.

Con esto dos conceptos, podemos describir la primera iteración del algoritmo usando el pseudocódigo en [3].

```
function Kernighan-Lin(G(V, E)) is
 determine a balanced initial partition of the nodes into sets A and B

 do
  compute D values for all a in A and b in B
  let gv, av, and bv be empty lists
  for n := 1 to |V| / 2 do
    find a from A and b from B, such that gain is maximal
    remove a and b from further consideration in this pass
        add g to gv, a to av, and b to bv
        update D values for the elements of A = A \ a and B = B \ b
  end for
  find k which maximizes g_max, the sum of gv[1], ..., gv[k]
  if g_max > 0 then
    Exchange av[1], av[2], ..., av[k] with bv[1], bv[2], ..., bv[k]
  until (g_max <= 0)

return G(V, E)
```

Entonces, en cada iteración, el algoritmo K-L intercambia pares de vértices para maximizar la ganancia. Continúa haciéndolo hasta que se intercambian todos los vértices de la partición más pequeña. Una gran ventaja del algoritmo es que no se detiene incluso aceptando ganancias negativas, sigue esperando que las ganancias posteriores sean grandes y que el tamaño de la suma de los pesos de las aristas se reduzca más tarde hasta que se intercambian todos los vértices de la partición más pequeña. Esta capacidad de salir de los mínimos locales es una característica crucial.

C. Fiduccia y R. Mattheyses realizaron importantes avances prácticos en [4], quienes mejoraron el algoritmo de Kernighan-Lin[1] de tal manera que, cada partición se ejecuta en $O(n^2)$, en lugar de $O(n^2 \log n)$.

2.1.2 Ejemplo de codificación

Para la codificación del algoritmo se ha utilizado la libraría de Python: NetworkX. El algoritmo divide un grafo en dos bloques usando el algoritmo Kernighan-Lin[1]. Es decir, divide un grafo en dos conjuntos intercambiando iterativamente pares de nodos para reducir el peso de las aristas entre los dos conjuntos.

Por ejemplo en una ejecución de la codificación, la entrada es el grafo de la Figura 2.1. Y después de una partición inicial podemos obtener los conjuntos: A = {3, 4, 6, 8, 9} y B = {0, 1, 2 , 5, 7}. Estos conjuntos son del mismo tamaño y sus elementos están ordenados.

Las principales desventajas de este algoritmo son:

- Los resultados son aleatorios porque el algoritmo comienza con una partición aleatoria.

- Computacionalmente es un algoritmo lento.
- Solo se crean dos particiones del mismo tamaño.
- Las particiones tienen que tener el mismo tamaño para que el algoritmo no intente encontrar tamaños de partición óptimos que ya existan.
- No resuelve muy bien los problemas con aristas ponderadas.
- La solución dependerá en gran medida del primer intercambio.

2.2 Spectral Bisection

2.2.1 Descripción

2.2.2 Ejemplo de codificación

Después de la partición obtenemos los conjuntos: A = {1, 2, 3, 4, 6, 9} y B = {0, 8, 5, 7}.

Los conjuntos no son del mismo tamaño. No están ordenados.

2.3 Multilevel Spectral Bisection

El particionamiento de grafos por multilevel es un método moderno que reduce el tamaño de las particiones del grafo con la combinación de los vértices y las aristas sobre varios niveles, creando particiones del grafo cada vez más pequeñas y extensas con muchas variaciones y combinaciones de diferentes métodos.

2.3.1 Descripción

2.3.2 Ejemplo de codificación

METIS es un algoritmo de partición que se enfoca en minimizar el número de bordes de partición cruzados y distribuir la carga de trabajo de manera uniforme entre las particiones. Con METIS, los gráficos se dividen en tres fases. La primera fase es la fase de engrosamiento, la segunda la fase de partición y la tercera y última fase la fase de no engrosamiento (Figura X). La fase de particionamiento contiene particiones bi-particionadas y K-way. A diferencia de la partición K-way, la bi-partición se realiza de forma recursiva. Las subseccion a continuación contienen una explicación más extensa de estas diversas fases.

El grafo de ejemplo inicialmente, vemos que sigue siendo el mismo que en el primer ejemplo (ver Figura **??**).

Después de la partición obtenemos los conjuntos: A = {0, 1, 3, 7, 8} y B = {2, 4, 5, 6, 9}.

Los conjuntos tienen el mismo tamaño. Están ordenados.

3. Comparativa entre los diferentes algoritmos

Para finalizar, veremos una comparativa entre los algoritmos que hemos analizado en las secciones previas, tras ser aplicados sobre grafos aleatorios, también conocidos como grafos de Erdös y Renyi o grafos binomiales. Pero, en primer lugar, definimos el concepto de grafo aleatorio.

Se dice que un grafo es aleatorio si la presencia y colocación de sus aristas siguen una distribución aleatoria. Por tanto, un grafo aleatorio no puede diseñarse con ningún criterio concreto. La noción del modelo de un grafo aleatorio incluye el límite entre cada par de nodos con igual probabilidad, independientemente de los extremos. Concretamente del modelo se denomina: modelo de Erdös y Renyi. El nombre del modelo proviene de los matemáticos Paul Erdős y Alfréd Rényi[5], quienes lo presentaron por primera en 1959.

El modelo Erdös y Renyi (a veces nombrado en la literatura abreviado como modelo ER), es el método que se ha empleado en la generación de los grafos aleatorios. Para ello, un grafo se construye mediante la conexión de los nodos al azar. Cada arista se incluye en el grafo con una probabilidad de p independiente de las otras aristas del grafo. De manera equivalente, todos los grafos con n nodos y m aristas tienen la misma probabilidad de:

$$p^m(1-p)^{\binom{n}{2}-m}$$

Para la implementación de los grafos aleatorios se ha utilizado la librería de Python: NetworkX. Concretamente el método que se ha utilizado nos devuelve un grafo no dirigido, donde se elige cada una de las aristas posibles con probabilidad $p = 0.7$, según el número de nodos de entrada n. En particular, el caso p = 0.7 se corresponde con el caso en el que todos los grafos en n vértices se eligen con mayor probabilidad. Los pesos de las aristas también se han generado aleatoriamente dentro de un intervalo de pesos de 1 a 20.

Podemos observar la codificación utilizada a continuación, que se ejecuta en tiempo $O(n^2)$.

```
G = nx.erdos_renyi_graph(n, 0.7)

for u, v in G.edges():
if u != v:
G[u][v]['label'] = random.randrange(1, 20)
```

Finalmente, en la tabla siguiente, se presenta una comparación entre los algoritmos codificados sobre los grafos aleatorios de este trabajo en términos de tiempo computacional. Se muestra el tiempo (en segundos) en completar ejecuciones sobre conjuntos de grafos aleatorios para distintos números de vértices, n. Para ello se ha utilizado un MacBook Pro, con un procesador de cuatro núcleos de 2.8 GHz y con 16 GB de memoria RAM. Es de esperar que, a mayor número de vértices, los algoritmos tengan mayor tiempo de ejecución.

n	307	552	861
Kernighan-Lin	0.0117	0.0214	0.0451
Spectral Bisection	0.0021	0.0031	0.0060
Multilevel Spectral Bisection	0.0025	0.0031	0.0037

Podemos ver como los algoritmos Kernighan-Lin, Spectral Bisection y Multilevel Spectral Bisection tienen unos tiempos de ejecución similares cuando el número de vértices n es más pequeño. Fenómeno que cambia cuando casi se duplican el número de vértices.

Si nos fijamos con el algoritmo Kernighan-Lin, cuando el número de vértices inicialmente casi se dobla, el tiempo de ejecución también lo hace. Parece que el algoritmo va aumentando el tiempo de ejecución a medida que aumenta el número de vértices de manera proporcionada.

En cambio, el algoritmo Spectral Bisection obtiene unos resultados muy parecidos con Multilevel Spectral Bisection, pero cuando el número de vértices empieza a ser muy elevado, la eficiencia baja drásticamente. Que no es el caso del algoritmo Multilevel Spectral Bisection.

Como conclusión global a la comparación entre los algoritmos podemos decir que el algoritmo Multilevel Spectral Bisection es el más eficiente que se ha codificado, y el algoritmo Kernighan-Lin es el menos eficiente.

4. Conclusiones

Hemos utilizado el mismo grafo aleatorio para los tres ejemplos descritos de las codificaciones de los algoritmos.

El algoritmo Multilevel Spectral Bisection es el más eficiente que se ha codificado, y el algoritmo Kernighan-Lin es el menos eficiente.

Para la comparativa solo se ha tenido en cuenta el tiempo de ejecución de los algoritmos. No se ha añadido el tiempo de ejecución empleado en la creación de los grafos, antes y después de la partición.

Cada vez que ejecutamos un algoritmo las soluciones cambian. Así que, por eso, los hemos comparado por tiempo de ejecución. Ya que, al ser algoritmos metaheurísticos, todas las soluciones pueden ser óptimas y válidas. Eso quiere decir, que, aunque un algoritmo sea más eficiente, no obtendrá la solución óptima.

Como hemos visto, a menudo existe una relación entre el tiempo de ejecución y la calidad de la solución. Algunos algoritmos funcionan muy rápido pero solo encuentran una solución de medio calidad mientras que otros tardan mucho tiempo pero ofrecen soluciones excelentes e incluso otros se puede sintonizar entre ambos extremos.

La elección del tiempo frente a la calidad.

Las particiones pequeñas en general podrían ser más rápidas que un algoritmo más lento con mejor calidad particiones Pero si usamos la misma matriz (o diferentes matrices con el mismo gráfico) a menudo, el algoritmo más lento podría ser preferible. tamaño de corte Todo esto debería demostrar que no existe un mejor algoritmo único para todas las situaciones. y que los diferentes algoritmos descritos en los siguientes capítulos tienen todos sus aplicaciones.

Bibliografia

[1] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, 1970. 3, 4, 5, 6

[2] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990. 3

[3] C. P. Ravikumar and George W. Zobrist. *Parallel Methods for VLSI Layout Design*. Greenwood Publishing Group Inc., USA, 1995. 5, 6

[4] C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. In *19th Design Automation Conference, Las Vegas, NV, USA, 14-16 June, 1982*. IEEE, 1982. 6

[5] P. Erdős and A. Rényi. On random graphs. *Publicationes Mathematicae*, 6:290–297, 1959. 8