Algoritmia para Problemas Difíciles
Práctica 2: Cobertura de vértices con pesos aproximado

José Daniel Subías Sarrato (NIA: 759533)
Fernando Peña Bes (NIA: 756012)

22 de enero de 2021

1. Descripción del problema

En ciencias de la computación existe un amplio número de problemas que son clasificados en grupos en función de su complejidad temporal. Algunos de estos son resolubles en un tiempo razonable, es por esto que se clasifican como problemas de tipo polinomial. No obstante, existen muchos problemas que no son resolubles en un periodo de tiempo aceptable. Estos problemas carecen de un algoritmo eficiente que los resuelva de una manera rápida y se clasifican como no polinomiales, más comúnmente conocidos como NP. Que estos problemas no tengan una metodología de resolución eficiente es realmente problemático, puesto que muchos tienen un gran número de aplicaciones en numerosos campos como la biología, la medicina... Es por esta razón que se han desarrollado técnicas como heurísticas o algoritmos aproximados para atacar los problemas NP, intentando así hallar en un tiempo razonable una solución subóptima que se aproxime en la medida de lo posible a la óptima.

En esta práctica se van a implementar dos algoritmos para resolver de forma aproximada el problema de cobertura de vértices con pesos (wVC). Este pertenece a la clase de problemas NP, por lo que habrá que abordar su resolución mediante algoritmos aproximados.

El problema de cobertura de vértices con pesos es una generalización del problema de cobertura de vértices, y tiene la siguiente definición: La entrada es un grafo $G = (V, E)$ donde V es el conjunto de vértices, E el conjunto de aristas y cada vértice tiene asociado un peso positivo $w(v)$. Se define como cobertura a un conjunto U que es subconjunto de V y que para cada arista (u, v) de G se cumple que $u \in U \vee v \in U$. Como salida, el algoritmo retorna el U de menor tamaño, definiendo este tamaño como $\sum_{v \in U} w(v)$.

Se van a implementar dos aproximaciones del problema de wVC: una relajación del problema de programación lineal equivalente, visto en las clases de teoría [1], y el algoritmo del *pricing method* [2]. Ambos algoritmos son una 2-aproximación.

Con el objetivo de comparar las prestaciones entre ambas implementaciones. Se hará un análisis de costes temporales en el que se compararán los dos algoritmos utilizados. Para ello se generará un pequeño conjunto de instancias con distintos tamaños de entradas, intentando cubrir el máximo número de casos posible.

2. Representación del grafo

Puesto que se trata de un problema de grafo, como el visto en la práctica anterior. Fue necesaria la definición de un tipo de dato grafo, sobre el que se trabajaría en las implementaciones de los algoritmos que explican en apartado posteriores. Aprovechando que se decidió hacer una implementación en el lenguaje de programación Python, se optó por utilizar las estructuras de listas que ofrece dicho lenguaje, ya que con ellas podía representarse un grafo de forma sencilla. Con esto podía crearse un grafo conforme se leía el fichero, donde estaba almacenada su información. De este modo el grafo consistía en una lista con dos listas, una para las aristas y otra para los vértices. La primera de ellas guardaba dos enteros que representaban los dos extremos de cada arista. Mientras que la segunda consistía en un vector de números reales $\{w_0, w_1 \dots w_{n-1}\}$ donde el índice representaba i representaba el nodo y la componente w_i el peso de dicho nodo.

3. Programación lineal

Tal y como se ha visto en clase de teoría el problema de wVC puede representarse, como un problema de de programación lineal. Para lograr tal representación, podemos asignar a cada vértice v una variable $x(v) = \{0, 1\}$, de modo que v pertenece al cubrimiento si y solo si $x(v) = 1$. La condición de que para cada arista (u, v) el grafo, uno de sus extremos debe estar en el cubrimiento puede expresarse del siguiente modo.

$$x(u) + x(v) \geq 1 \tag{1}$$

La expresión de esta ecuación indica que para cada par de vértices que conforman una arista, la suma de los dos valores debe ser como mínimo igual a 1, garantizando así que al menos uno de los dos vértices esta en el cubrimiento. De este, modo disponemos de un programa ILP (*Integer Linear Programming*) que tratará de minimizar la siguiente función objetivo.

$$min \sum_{\forall v \in V} w(v)x(v) \tag{2}$$

Como restricciones del propio problema se plantea la ecuación 1 y el echo de que cada variable $x(v)$ tome los valores 1 o 0. No obstante, esta última restricción limita bastante la resolución del problema y el estudio de la aproximación al óptimo. Esto se debe a que se trata de una restricción que da cambios bruscos, en el valor de las variables al cambiar solamente de 0 a 1 o viceversa. Es por esta razón que se plantea la siguiente relajación de la restricción al problema.

$$0 \leq x(v) \leq 1 \forall v \in V \tag{3}$$

Con esto se consigue un problema de programación lineal que es mas fácil resolver con algoritmos como el símplex. Además, como el problema original se trata de un caso particular del problema relajado, el óptimo de la relajación es una cota inferior del óptimo original.

Una vez solucionado el problema mediante programación lineal para la asignación del valor a las variables $x(v)$, puede plantearse un redondeo para construir la cobertura. Puesto que ahora nuestros vértices tienen asignada una variable $0 \leq \hat{x}(v) \leq 1$, pueden seleccionarse los vértices de la cobertura añadiendo todo los que cumplan con $\hat{x} \geq 1/2$. Como resultado queda el siguiente algoritmo.

```
1: function Aprox-Min-Weight-VC(G = (V, E))
2:     U ← ∅
3:     x̂ ← ProgramacionLineal(G)
4:     for v ∈ V do
5:         if x̂(v) ≥ 1/2 then
6:             U ∪ {v}
7:     return U
```

Este algoritmo resultante se trata de una 2-aproximación del problema original, tal y como se ha explicado en las clases de teoría de la asignatura. La implementación del algoritmo puede verse en el fichero `vertex_cover.py` en la función `MinWeightVC()`. Para la implementación de la resolución del problema de programación lineal, se ha utilizado la herramienta `pywraplp` del conjunto de herramientas python de `ortools` [3].

4. Pricing Method

Como segunda implementación para la resolución del problema de wVC y tal como se pide en el enunciado, se pretende utilizar el algoritmo de *pricing method*. Para poder resolver el problema original mediante *pricing method*, se debe partir del problema dual del problema de programación lineal explicado en el apartado anterior. Dicho problema dual viene definido por la siguiente función objetivo.

$$max \sum_{\forall e \in E} y(e) \tag{4}$$

Como se puede ver, ahora no se asignan variables $x(v)$ a los nodos, sino variables $y(e)$ a las aristas del grafo. Esto viene por que ahora cada aristas "pagará" un determinado "precio" $y(e)$ por ser cubierta. De este modo, ahora se busca la forma de obtener la mayor recaudación de aristas como sea posibles. Por esta razón se quiere maximizar el sumatorio de la ecuación 4. Por otro lado, para cada variable $y(e)$ deben cumplirse las siguientes restricciones:

$$\sum_{\forall e=(i,j) \in E} y(e) \leq w_i \tag{5}$$

$$0 \leq y(e) \tag{6}$$

Con la restricción de la ecuación 5 se consigue que la suma del precio que pagan las aristas que salen de un vértice no supere su peso, mientras que con la restricción de la ecuación 6 se indica que el precio no puede ser negativo. Aplicando las relaciones entre el problema de PL original y el dual se puede llegar a la conclusión de que:

$$\sum_{\forall e \in E} y(e) \leq w(U), \tag{7}$$

donde $w(U)$ representa el peso de la cobertura mínima.

Puesto que alguna solución y del dual da como solución una cota inferior al problema original, no es necesario resolver en su totalidad el problema dual. No obstante, la solución y debe ser fácil de convertir a una cobertura de vértices U, que no debe estar lejos de la solución óptima. Por estas razones podemos encontrar una solución de forma rápida, sin necesidad de resolver el problema de programación lineal.

En base a dicha dualidad puede plantearse la siguiente definición de lo que se considera como vértice *tight* para cualquier vértice i del grafo:

$$\sum_{\forall e=(i,j)\in E} y(e) = w_i \tag{8}$$

Esto implica que un vértice e considerado *tight* cuando el total de los precios de sus aristas, es igual al peso de dicho vértice. Con esto puede aplicarse el método de *pricing method* para resolver el problema de manera aproximada:

```
1: function PRICING-METHOD-VC(G = (V, E))
2:     U ← ∅
3:     for e ∈ E do
4:         y(e) = 0
5:     while existe una arista e = (i, j) tal que ni i ni j son tight do
6:         INCREMENTAR(y(e))
7:     for v ∈ V do
8:         if v es tight then
9:             U ∪ {v}
10:    return U
```

La función INCREMENTAR aumenta el precio de la arista e lo máximo posible respetando la ecuación 5, que para cualquier arista (i, j) en cualquier momento del algoritmo es el siguiente:

$$\text{mín}\left\{ w_i - \sum_{\forall e=(i,x)\in E} y(e), \quad w_j - \sum_{\forall e=(j,y)\in E} y(e) \right\} \tag{9}$$

Una observación importante es que una vez que cuando se incrementa el precio de una arista, alguno de los dos vértices con los que está conectada se vuelve *tight*, por lo que esa arista no podrá volverse a elegir en el bucle del algoritmo. Esto significa que es posible implementar el *pricing method* recorriendo cada arista una vez únicamente, haciendo que el coste del algoritmo sea lineal en el número de aristas.

En el fichero `vertex_cover.py` se encuentra la función `PricingMethod`, que implementa el *pricing method* con coste lineal.

5. Pruebas

Una vez implementados ambos algoritmos, se realizaron pruebas tanto de prestaciones como de tiempo. Todas las pruebas se han realizado en `lab000`.

Los grafos de pruebas se generaron de manera aleatoria, variando el número de vértices y su densidad.

5.1. Comparación de prestaciones

Para comparar las prestaciones, se han realizado varias pruebas comparando el peso mínimo devuelto por las aproximaciones con el peso óptimo.

Sabemos que las dos algoritmos son 2-aproximaciones, pero para comprobar experimentalmente cómo de buenas son las soluciones obtenidas, el programa de PL relajada se adaptó para que resolviera el problema original, utilizando programación lineal entera. De esta forma, el programa ILP resuelve completamente wVC y siempre devuelve la respuesta óptima.

Observamos que ILP es intratable, ya que es capaz de resolver wVC. Esto significa que algunas instancias pueden tardar mucho tiempo en resolverse, y por este motivo sólo se han hecho pruebas con grafos de hasta 200 vértices (unos 12 minutos de ejecución en los peores casos).

En las gráficas de la Figura 1 se recopilan los resultados obtenidos.

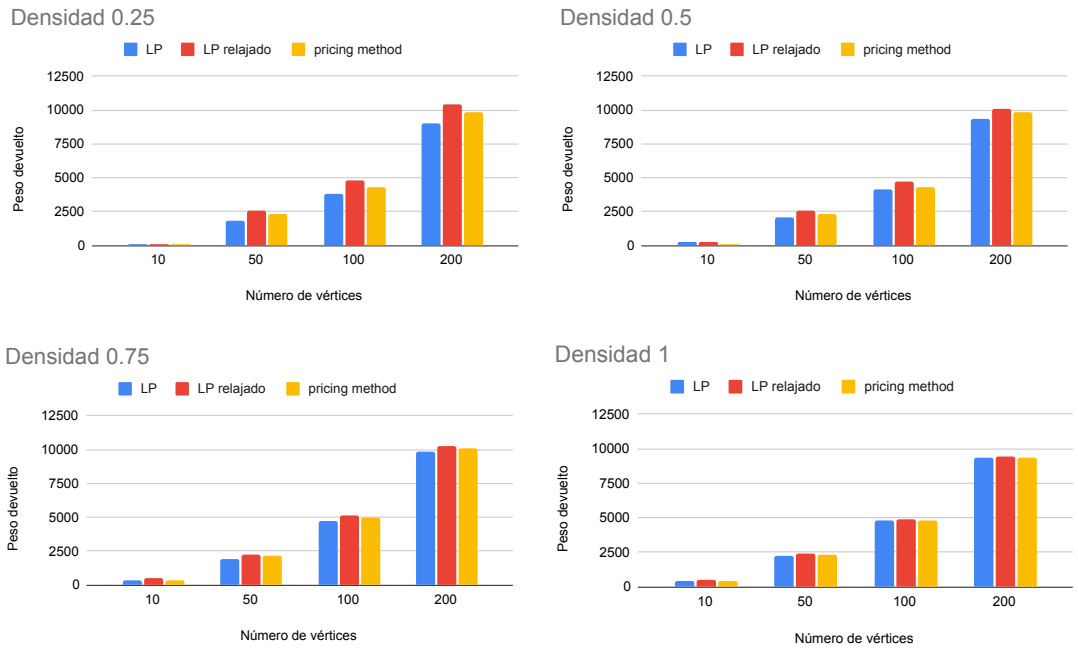

Figura 1: Comparación de prestaciones.

La mayor diferencia de las aproximaciones se ha producido en los grafos con 50 vértices y densidad para ambas, la relajación de PL ha realizado una aproximación de 1,43 y el *pricing method* ha alcanzado un 1,27. En las gráficas se puede ver que el *pricing method* devuelve una mejor aproximación que la relajación de PL en todos los casos. También se ha observado que la diferencia entre las aproximaciones y el la solución óptima tiende a decrementar conforme aumenta la densidad del grafo.

5.2. Comparación de tiempo de ejecución

En estas pruebas se compara el tiempo de ejecución de las dos aproximaciones. En la Figura 2 se muestran las medidas para la relajación de PL, y en la Figura 3 las de *pricing method*.

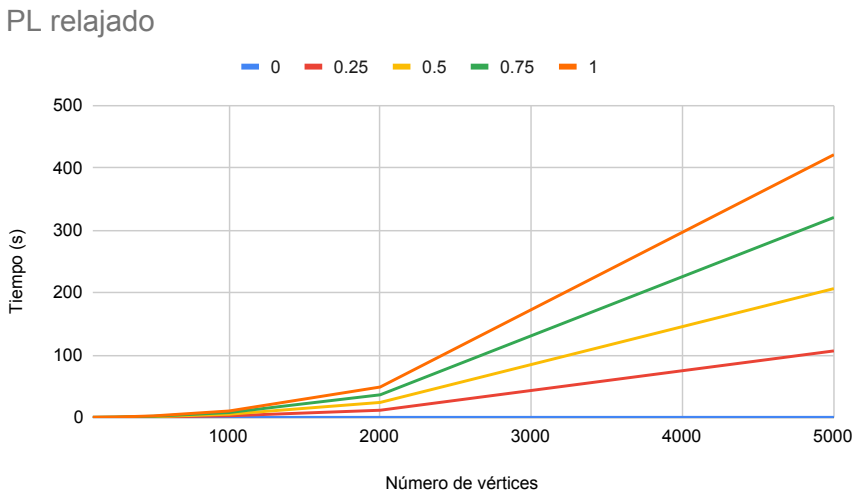

Figura 2: Tiempo de ejecución de la relajación de PL.

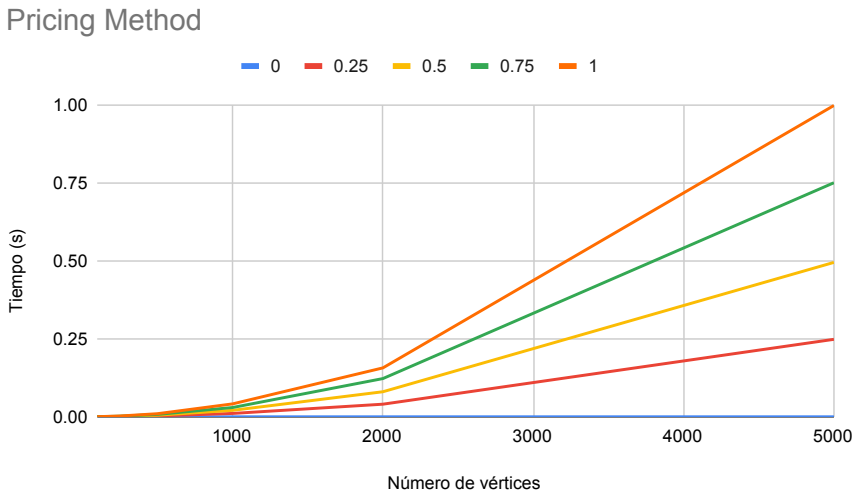

Figura 3: Tiempo de ejecución del *pricing method*.

En los dos casos el tiempo de ejecución crece conforme se aumenta el número de vértices y la densidad del grafo. Pero lo que más destaca es que el *pricing method* es siempre mucho más rápido. Esto se debe a que no resuelve completamente el problema de PL, su coste es lineal en el número de aristas como se ha comentado en el apartado anterior. El módulo utilizado de programación lineal para resolver la relajación usa el algoritmo símplex, cuyo coste en el peor caso es factorial, aunque en el caso medio (que es el que aparece en la mayoría de aplicaciones prácticas) es polinomial.

6. Conclusiones

En esta práctica se han implementado dos aproximaciones para el problema intratable de cobertura de vértices con pesos. Se han comparado experimentalmente ambas aproximaciones y se ha llegado a la conclusión de que el *pricing method* es mejor que la relajación de PL, siempre devuelve una mejor aproximación y su tiempo de ejecución es mucho menor en todos los casos al tener coste lineal en el número de aristas.

7. Reparto de trabajo

José Daniel Subías Sarrato

Horas invertidas: 8

Tareas realizadas:

- Implementación del algoritmo de programación lineal.
- Implementación del algoritmo del *pricing method*.
- Redacción del informe.

Fernando Peña Bes

Horas invertidas: 10

Tareas realizadas:

- Implementación de la generación de los datos.
- Generación de datos.
- Implementación del algoritmo de programación lineal entera.
- Implementación del algoritmo del *pricing method*.
- Diseño y ejecución de pruebas.
- Redacción del informe.

Referencias

[1] Transparencias de la asignatura *Algoritmia para problemas difíciles*. `http://webdiis.unizar.es/asignaturas/APD/wp/wp-content/uploads/2013/09/181123AproximadosI.pdf`.

[2] Transparencias para resolver cobertura de vértices con *pricing method*. `http://web.cs.iastate.edu/~cs511/handout10/Approx_VC.pdf`.

[3] Documentacion de la herramienta `pywraplp`. `https://developers.google.com/optimization/lp/glop`.