Parcial 3 – 2020-3
David Ricardo Bernal Alfonso
Análisis de algoritmos

1. Teoría general de intratabilidad (35)

En los problemas NP disponemos de un algoritmo de verificación para comprobar que una solución se ajusta a dicho problema. La verificación de si esta solución propuesta es satisfactoria o no, se realiza en tiempo polinomial $O(n^k)$ siendo n el tamaño de la solución $|S|$ y k el número de recorridos realizados por lo que la dificultad radica principalmente en encontrar o generar dicha solución. Como este aspecto se desconoce, es necesario probar todas las combinaciones o caminos posibles para generar una solución (tal como sucede con los problemas de fuerza bruta) lo cual requiere un tiempo exponencial que sería $O(2^n)$.

Por ejemplo, asumamos que tenemos un problema NP y lo ejecutamos en una maquia no determinista, por cada paso o iteración realizado existe dos opciones o caminos para elegir. Como debemos probar todos los caminos posibles, elegimos ambos en cada iteración lo cual significa que el número máximo de combinaciones está dado por 2^{n^k} y cada uno de ellos puede ser probado en tiempo polinomial $p(n^k)$ (tal y como se prueba las soluciones o certificados para un problema NP). Esto significa que el tiempo total es $O(2^{n^k})$ para un problema NP.

(196 Palabras) - https://www.contadordepalabras.com/

2. Solución de problemas difíciles (35)

Dominando una solución para conjunto dominante.

Actualmente, es muy común creer que los problemas NP no puedan resolverse en un tiempo polinómico. El hecho que de exista una baja probabilidad que un algoritmo con complejidad polinomial pueda ser ajustado o utilizado para resolver de forma óptima los problemas NP ha obligado tanto a los profesionales e investigadores en este campo a tratar de resolver estos problemas de manera eficiente con la esperanza de obtener algoritmos que posean un balance entre la calidad de la solución y el tiempo que toma llegar a dicha solución.

Dentro de la numerosa lista de problemas NP que actualmente son tema de investigación, se encuentra el problema del Conjunto Dominante (Dominating Set). Para un grafo G = (V, A) conformado por una lista de Vértices y una lista de Aristas, se establece que un conjunto dominante consiste en un subconjunto de vértices D que pertenece a V tal que cada vértice V es un vecino al menos de un vértice del subconjunto D. [1]

Antes de entrar en materia con la descripción y análisis del acercamiento que otorga una posible solución para nuestro problema, debemos describir de forma breve cuales son los métodos más comunes utilizados para la resolución general de problemas NP. Entre estos técnicos destacan: aproximación de la respuesta, parametrización del problema o por heurística.

Un algoritmo de aproximación devuelve una solución a un problema de optimización. Esta técnica no garantiza la mejor solución ya que su objetivo es acercarse lo más posible a la respuesta esperada [2]. Este tipo de algoritmos se utilizan normalmente cuando es difícil encontrar una solución óptima, pero también se pueden utilizar en algunas situaciones en las que se puede encontrar rápidamente una solución casi óptima y no se necesita una solución exacta.

En el caso de la parametrización para lidiar con la dificultad de encontrar una solución óptima a los estos problemas, acá se pregunta por cuál es el mejor tiempo de ejecución que podemos lograr con respecto a algún parámetro del problema, en lugar del tamaño de la instancia de entrada. Este parámetro normalmente se denota como k se toma como el tamaño de la solución por lo que puede tomar tiempos exponenciales de ejecución en k.

Por último, los algoritmos heurísticos se ejecutan más rápido que los algoritmos exactos o de aproximación y permiten calcular soluciones sub-optimas a la solución esperada (algunas de estas soluciones sub-optimas son inviables). En otras palabras, estos algoritmos se limitan a entregar una buena solución rápida a cambio de sacrificar la optimización, exactitud y precisión.

Teniendo en lo descrito anteriormente, uno de los acercamientos más comunes para nuestro problema es la aproximación a un conjunto dominante mínimo por medio de un algoritmo voraz [3] [4]. Para ello se establece que un vértice *v* cubre a un vértice *u* si los vértices *v* y *u* son adyacentes o si el vértice *v=u* y dichos vértices deben estar contenidos dentro un subgrafo D que debe cubrir todos los vértices en el grafo superior G.

En la estrategia principal de este acercamiento voraz para encontrar el conjunto mínimo dominante se inicia con un conjunto S vacío ($S = \emptyset$) y vamos agregando los vértices o nodos que cubran la mayor cantidad posible de los nodos vértices que previamente se encontraron o descubrieron al conjunto S hasta que S sea un conjunto dominante.

Para ello vamos a marcar los nodos que estamos visitando por colores y/o agregando al conjunto S. Los nodos que vamos agregando o se encuentren en el conjunto S se colorean de negro, los nodos que están cubiertos por algún nodo en S (es decir que existe una arista (v,u) tal que u es el nodo vecino de v) se colorean en gris y se colorea de blanco los nodos o vértices que no están cubiertos. Los nodos blancos que son vecinos directos o inmediatos de un nodo v (incluido el mismo

nodo v) serán determinados por la función *vecinos(v)* (determina el alcance del nodo v). El seudocódigo para el proceso descrito anteriormente seria el siguiente:

```
function conjuntoDominanteVoraz:
        S = []
        while (existeNodosBlancos) do:
                x = max(vecinos(v), vecinos(u))
                S.add(x)
        end while
end function
```

Si asumimos que tenemos un nodo v blanco y que sus vecinos $u_1,..,u_4$ son blancos, los 5 nodos son cambiados de color ya que v pasa a ser de color negro (dominante) y los otros nodos u pasan a ser gris (al ser cubiertos por el nodo v), y el costo de esto sería 5. Si el nodo v ya es gris, solo cambian de color sus nodos vecinos por lo que el costo disminuye. Cada vez que escogemos un nuevo nodo para agregar al conjunto dominante S, tiene costo 1.

Otro acercamiento partiendo del anterior seria reduciendo el alcance que tiene un nodo. Este alcance solo puede ser reducido si algunos de los nodos ubicados a una distancia máxima de 2 ya está incluido en el conjunto S, por lo que se establece que, si el alcance de un nodo v es mayor que el alcance de cualquier otro nodo a una distancia 2 como máximo, el algoritmo debe elegir v antes que cualquiera de los otros nodos.

¿Pero por qué interesa una solución para el problema de conjunto dominante?

El uso creciente de varios distintos dispositivos móviles ha generado redes de comunicación donde la topología se encuentra en constante cambio. Las redes vehiculares, las redes de sensores inalámbricas (WSN), redes de control y toleración de retrasos son algunos de los ejemplos de estas redes dinámicas. Los usuales modelos gráficos no suelen ser siempre la mejor solución para visualizar y analizar redes tan complejas y giganticas en algunos casos. Tener la capacidad de detectar en una gran red corporativa cuales son los nodos dominantes o puntos críticos que controlan dos o más servicios esenciales para el funcionamiento continuo e ininterrumpido de la red, y que además permiten la comunicación con otras redes.

Es aquí donde la importancia de abordar y explorar las soluciones que se han planteado alrededor de este problema radica en la gran variedad de aplicaciones mencionadas anteriormente y que giran alrededor de una solución como la que explicó a lo largo de este escrito.

3. **Reducción y equivalencia de problemas NP-completos (35)**
    a. **Problema NP Cometa.** Una cometa es un grafo con un número par de vértices, digamos 2n, en donde n vértices forman una clique y los otros n una cola, la cual es un camino que llega a uno de los vértices de la clique. Dado un grafo y un número g, el problema de la cometa consiste en decidir si el grafo cuenta con una cometa de tamaño 2g. Demuestre que este problema es NP-completo.

El certificado es el conjunto de vértices C de Clique de tamaño g ya que el tamaño de cometa es 2g (Clique+Cola). Verificó que el grafo G tiene una cola del mismo tamaño del certificado y luego compruebo si el certificado es Clique.

```
verificarCometa(Grafo G; Vertices C; meta g ):
        V=G.getVertices()
        cola=false
        for each v in C:
                if (tieneCola(v,g)):
                        cola=true
                        break
        if(cola):
                if(len(C)=g):
                        for each v in C:
                                for each c in C:
                                        if not(G.costoArista(v,c) < Inf):
                                                return false
                else:
                        return false
                return true
        else:
                return true
```

Reducción Clique a Cometa:

Para la reducción se tiene que dado un grafo G=(V,A) que tiene un subgrafo Clique (o un conjunto de vértices C), añadir una cola de tamaño g a cada vértice del grafo G. Luego se debe verificar si el nuevo grafo es Clique.

```
esSatisfactible(Grafo G, meta g):
        H = nuevoGrafo(G,g)
        return existeClique(H,g)

nuevoGrafo(Grafo G, meta g):
        V=G.getVertices()
        for each v in V:
                G.añadirCola(g)
        return G
```

b. **Problema NP Presupuesto.** Nos dan un conjunto de conjuntos {S1, S2, . . . Sn} y un presupuesto p. Nuestra misión consiste en encontrar un conjunto H, que tenga a lo sumo p elementos, en el cual $H \cap Si \neq \emptyset$ para $i \in 1..n$. Demuestre que este problema es NP-completo.

El certificado es el subconjunto H de tamaño b cuyos elementos pertenecen al conjunto S, y que contiene al menos un elemento de cada subconjunto S_i. Se debe comprobar por cada conjunto S si existe al menos un elemento coincidente con el conjunto H.

```
validarSubconjuntos(Conjunto S, Conjunto H, entero b):
    if (len(H) == b):
            for each s in S:
                    if (sonDisyuntos(s,H)):
                            return false
            return true
    else:
            return false
```

Referencias

[1] P. Harsha y S. Srinivasan, «Approximation Algorithms for NP-hard problems,» 2 Febrero 2010. [En línea]. Available: http://www.tcs.tifr.res.in/~prahladh/teaching/2009-10/limits/lectures/lec01.pdf. [Último acceso: 27 Noviembre 2020].

[2] R. Kannan y M. Karpinski, «Approximation Algorithms for NP-Hard Problems,» 24 Junio 2004. [En línea]. Available: https://www.researchgate.net/publication/200031635_Approximation_Algorithms_for_NP-Hard_Problems. [Último acceso: 27 Noviembre 2020].

[3] R. Rajaraman, T. Suel y L. Jia, «An Efficient Distributed Algorithm for Constructing Small Dominating Sets,» 2002. [En línea]. Available: http://engineering.nyu.edu/~suel/papers/domi.pdf. [Último acceso: 27 Noviembre 2020].

[4] R. Wattenhofer y F. Kuhn, «Constant-time distributed dominating set approximation,» Mayo 2005. [En línea]. Available: http://ac.informatik.uni-freiburg.de/publications/publications/podc03a.pdf. [Último acceso: 27 Noviembre 2020].

[5] «Chapter 7 - Dominant Set,» [En línea]. Available: http://ac.informatik.uni-freiburg.de/teaching/ss_12/netalg/lectures/chapter7.pdf. [Último acceso: 27 Noviembre 2020].

[6] D. Corneil y L. Stewart, «Dominating Sets in Perfect Graphs,» 2 Diciembre 1998. [En línea]. Available: https://core.ac.uk/download/pdf/82635697.pdf. [Último acceso: 27 Noviembre 2020].

[7] K. L. Ma, «In Solving the Dominating Set Problem: Group Theory Approah,» Mayo 1998. [En línea]. Available: https://spectrum.library.concordia.ca/478/1/NQ40311.pdf. [Último acceso: 27 Noviembre 2020].