NP demostration

Alex Sánchez,Jorge Alberto Aspiolea,Carlos Manuel Gonzales

September 2024

1 Ejercicio

El problema 2 puede ser resumido a encontrar la menor cantidad de pasillos del laberinto que debemos eliminar (monstruos a matar , bueno, los mata Carlos) para eliminar los ciclos del laberinto , modelando el laberinto como un conjunto de salas conectadas por pasillos unidireccionales, podríamos tomar las salas como nodos y los pasillos como arcos de un Digrafo cíclico (si no hay ciclios no tiene sentido el ejercicio)

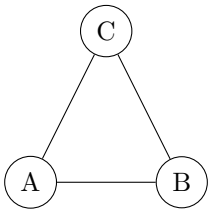
2 Comprobación en timepo Polinomial

Si queremos comprobar una solución ¡G,k¿ es decir si eliminando k arcos eliminamos los ciclos. Simplemente eliminamos esos k arcos y comprobamos que no queden ciclos en el grafo resultante. Usando algoritmos como Floyd Warshall O(V^3), verificando aristas de retroceso con Dfs O(V(G)+E(G)) etc). Por lo que verificar una solución es posible en tiempo polinomial.

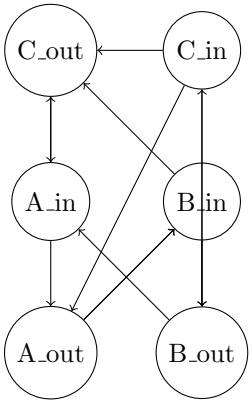
3 Reduciendo a Vertex Cover

Supongamos que tenemos un grafo G,convirtámoslo en un grafo dirigido. Siguiendo las siguientes reglas:

1. Por cada vertice $u \in V(G)$ creamos en Gf(grafo dirigido resultante) dos vértices u_in y u_out y conectamos u_in con u_out
2. Por cada arista $u, v \in E(G)$ conectamos u_out con v_in y v_out con u_in en Gf



figureGrafo G



figureGrafo resultante G_f

Esta transformación es posible hacerla en tiempo polinomial siguiendo las dos reglas anteriores por cada nodo , por lo que la transformación se haría en O(V(G))

Por cada vértice en G se construye un ciclo en Gf, asumimos que el grafo no tiene vertices aislados de tener estos sería imposible hacer vertex cover y no tendría sentido la reducción. Por tanto si existiera un algoritmo que resolviera este problema en una complejidad polinomial, se pudiera transformar a vertex Cover en una complejidad polinomial, por tanto se resolvería Vertex Cover en una complejidad polinomial , no puede pasar pq Vertex Cover está demostrado es NP-Hard. Por tanto el problema en cuestión es NP-Completo

Este problema es similar a un problema llamado Feedback arc set ya que ambos consisten en eliminar la minima cantidad de arcos en un digrafo aciclico tal que el resultado sea un DAG. Este problema fue demostrado NP por Karp.

4 Algoritmo Aproximados y heuristicas

El algoritmo GreedyFAS es una aproximación heurística para resolver el problema del feedback arc set (FAS), pero no garantiza que la solución sea óptima. Sin embargo, en la práctica, la solución obtenida puede estar razonablemente cerca del óptimo en muchos casos.

No da garantias teoricas de su cercania al resultado optimo real. Para cada vértice u en el grafo G, se define:

- $d^+(u)$: el número de arcos que salen de u (outdegree).
- $d^-(u)$: el número de arcos que entran en u (indegree).
- $\delta(u) = d^+(u) - d^-(u)$: una medida de cuán "fuente" o "sumidero" es el vértice.

Proceso

En cada iteración, el algoritmo elimina vértices de G siguiendo las siguientes reglas:

1. Elimina vértices que son sumideros ($d^+(u) = 0$) y prepéndelos a una secuencia s_2.

2. Elimina vértices que son fuentes ($d^-(u) = 0$) y añádelos al final de una secuencia s_1.

3. Si no hay fuentes ni sumideros, elimina el vértice u con el mayor valor de $\delta(u)$ y añádelo a s_1.

Resultado

Cuando se eliminan todos los vértices, se obtiene una secuencia $s = s_1 + s_2$, donde los arcos orientados de derecha a izquierda (backward arcs) forman un conjunto de arcos de retroalimentación.

La complejidad temporal del algoritmo es O(V + E), donde V es el número de nodos y E es el número de aristas en el grafo. Esto se debe a que cada nodo y cada arista se procesan una vez."

Algoritmo SimpleFAS

El algoritmo SimpleFAS se basa en un algoritmo muy simple de 2-aproximación para el problema de conjunto de arcos de retroalimentación mínimo (MAS). El proceso se describe a continuación:

Proceso

1. Primero, se fija una permutación arbitraria P de los vértices de G.

2. Luego, se construyen dos subgrafos L y R:

   - L contiene los arcos (u, v) donde $u < v$ en P.
   - R contiene los arcos (u, v) donde $u > v$ en P.

3. Después de esta construcción, tanto L como R son subgrafos acíclicos de G.

4. Al menos uno de ellos tiene un tamaño que es al menos la mitad del tamaño del máximo subgrafo acíclico. Por lo tanto, podemos devolver $m - \max(|L|, |R|)$ como el tamaño de un conjunto de arcos de retroalimentación para G.

Complejidad

La complejidad en tiempo del algoritmo SimpleFAS es $O(m + n)$, donde m es el número de arcos y n es el número de vértices en el grafo.

[colback=white!95!blue,colframe=blue!40!black,title=Demostración de que SimpleFAS es una 2-aproximación] Sea OPT el tamaño del conjunto mínimo de arcos de retroalimentación (la solución óptima). Queremos probar que el conjunto de retroalimentación producido por SimpleFAS es como máximo $2 \times OPT$.

Observaciones:

1. Los subgrafos L y R son acíclicos por construcción, ya que L contiene los arcos donde $u < v$ según la permutación P, y R donde $u > v$. Los grafos acíclicos no contienen ciclos.

2. El grafo G se descompone en los arcos de L y R. Al menos uno de estos subgrafos contiene al menos la mitad de los arcos de un subgrafo acíclico máximo. Es decir:

$$\max(|L|, |R|) \geq \frac{|A_{max}|}{2}$$

donde $|A_{max}|$ es el número de arcos en el subgrafo acíclico máximo.

3. La solución óptima OPT es tal que eliminar OPT arcos deja un subgrafo acíclico de tamaño $|E| - OPT$.

Cota superior:

El algoritmo SimpleFAS devuelve $|E| - \max(|L|, |R|)$. Dado que $\max(|L|, |R|) \geq \frac{|E|-OPT}{2}$, tenemos:

$$SimpleFAS \leq |E| - \frac{|E| - OPT}{2}$$

Simplificando:

$$SimpleFAS \leq \frac{|E| + OPT}{2}$$

Dado que $|E| \geq OPT$, obtenemos:

$$SimpleFAS \leq 2 \times OPT$$

Por lo tanto, SimpleFAS es una 2-aproximación.

[colback=white!95!blue,colframe=blue!40!black,title=Demostración de que BergerShorFAS es una 2-aproximación] Sea OPT el tamaño del conjunto mínimo de arcos de retroalimentación (la solución óptima). Queremos probar que el conjunto de retroalimentación producido por BergerShorFAS es como máximo $2 \times OPT$.

Observaciones:

1. Al procesar cada vértice v, se elige entre los arcos entrantes y salientes basándose en cuál conjunto tiene más arcos. Esto asegura que en cada paso se mantiene un subgrafo acíclico.

2. Supongamos que E es el conjunto de arcos del grafo original y E_0 es el conjunto de arcos seleccionados por el algoritmo. Los arcos que no están en E_0 forman el conjunto de arcos de retroalimentación. Es decir:

$$F = E \setminus E_0$$

3. Si el número de arcos en el subgrafo acíclico es al menos $(\frac{1}{2}+\Omega(\frac{1}{\sqrt{d_{max}}}))|E|$, donde d_{max} es el grado máximo de los vértices, entonces:

$$|E_0| \geq \left(\frac{1}{2} + \Omega\left(\frac{1}{\sqrt{d_{max}}}\right)\right)|E|$$

Cota superior:

La cantidad de arcos de retroalimentación devuelta por BergerShorFAS es:

$$|F| = |E| - |E_0|$$

Sustituyendo la desigualdad anterior, tenemos:

$$|F| \leq |E| - \left(\frac{1}{2} + \Omega\left(\frac{1}{\sqrt{d_{max}}}\right)\right)|E|$$

Simplificando:

$$|F| \leq \left(\frac{1}{2} - \Omega\left(\frac{1}{\sqrt{d_{max}}}\right)\right)|E|$$

Dado que $|E| \geq OPT$, se sigue que:

$$|F| \leq 2 \times OPT$$

Por lo tanto, BergerShorFAS es una 2-aproximación.

4.1 KiwiSort

[colback=white!95!blue,colframe=blue!40!black,title=Algoritmo KwikSortFAS] **Input:** Arreglo lineal A, vértice lo, vértice hi

Output: Un conjunto de arcos de retroalimentación para G

1. Si $lo < hi$ entonces: - Inicializar $lt \leftarrow lo$, $gt \leftarrow hi$, $i \leftarrow lo$. - Elegir un pivote aleatorio p en el rango $[lo, hi]$. - Mientras $i \leq gt$ hacer: - Si existe un arco (i, p): - Intercambiar lt con i. - Incrementar lt y i. - Sino, si existe un arco (p, i): - Intercambiar i con gt. - Decrementar gt. - Sino: - Incrementar i. 2. Llamar recursivamente a $KwikSortFAS(A, lo, lt-1)$. 3. Si se realizó al menos un intercambio: - Llamar recursivamente a $KwikSortFAS(A, lt, gt)$. 4. Llamar recursivamente a $KwikSortFAS(A, gt+1, hi)$.

Nota: El algoritmo utiliza un método de partición de 3 vías para ordenar, lo que permite manejar eficientemente los vértices desconectados.

[colback=white!95!blue,colframe=blue!40!black,title=Demostración de que KwikSortFAS es una 3-aproximación] Sea OPT el tamaño del conjunto mínimo de arcos de retroalimentación (la solución óptima). Queremos demostrar que el conjunto de retroalimentación producido por KwikSortFAS es como máximo $3 \times OPT$.

Observaciones:

1. El algoritmo utiliza un método de ordenamiento que intenta minimizar la cantidad de arcos de retroalimentación al organizar los vértices en un arreglo lineal favorable.

2. Al finalizar el ordenamiento, cada arco que va en dirección contraria al ordenamiento se cuenta como un arco de retroalimentación. Dado que el algoritmo organiza los vértices basándose en la existencia de arcos entre ellos, el número de arcos de retroalimentación generados es proporcional al desorden inicial de los arcos.

3. En el peor de los casos, el número de arcos de retroalimentación producidos por KwikSortFAS puede ser hasta el triple del número de arcos que se necesitarían eliminar para obtener un grafo acíclico. Esto se puede establecer formalmente como:

$$|F| \leq 3 \times OPT$$

Cota superior:

Al ordenar los vértices, si consideramos que un arreglo óptimo (o deseado) requeriría eliminar un número OPT de arcos para eliminar todos los ciclos, la naturaleza del algoritmo permite que, en el peor caso, hasta dos arcos adicionales puedan contribuir a la retroalimentación. Por lo tanto, se establece que:

$$|F| \leq |E| - |E_{final}| \leq 3 \times OPT$$

Por lo tanto, KwikSortFAS es una 3-aproximación.

5 bibliografia

https://vldb.org/pvldb/vol10/p133-simpson.pdf