LA COMPLEJIDAD PARAMÉTRICA DE MINAR GRAFOS 2, RESULTADOS POSITIVOS

J. ANDRÉS MONTOYA.

Abstract. In this paper we analyze the parameterized complexity of graph mining tasks, specifically we analyze the parameterized complexity of the listing problem consistent in: Given an input graph G, list the frequent subgraphs of G of a given size. In this paper we prove some upper bounds for suitable restrictions of this problem.

RESUMEN. En este artículo analizamos la complejidad paramétrica de algunos problemas típicos en minería de grafos, específicamente nosotros analizamos la complejidad paramétrica del problema de listado consistente en: Dado G un grafo-input, liste todos los subgrafos frecuentes de G de un tamaño dado. En el artículo se prueban cotas superiores para algunas restricciones adecuadas del problema.

A Mamadimitriou e Hijodimitriou

1. Introducción

Este escrito es resultado de un proyecto de investigación que se ha fijado dos objetivos fundamentales, los cuales aunque opuestos son complementarios. El primer objetivo consiste en analizar la dificultad intrínseca de minar grafos, entendiendo esta tarea computacional como la tarea consistente en listar todos los patrones frecuentes y todos los patrones excepcionales de un grafo dado. En este escrito atacamos el segundo objetivo consistente en identificar restricciones del problema general que admiten soluciones eficientes en el sentido de la complejidad paramétrica.

1.1. Complejidad paramétrica.

La Complejidad paramétrica intenta analizar la complejidad de lenguajes que admiten una parametrización natural. Considere por ejemplo el problema del *Clique*.

Problema 1 (*Clique*)**.** .

- *Input:* (G, k)*, donde* G *es un grafo y* $k \in \mathbb{N}$*.*
- *Problema: Decida si* G *contiene un grafo completo de tamaño* k*, (i.e. un* k*-clique).*

Es un hecho bien conocido que el problema del *Clique* es NP-completo (vea [11]). Quien revise cuidadosamente la prueba de NP-completez del problema del *Clique*, notará que en ella es necesario considerar instancias (G, k) , en las que k es aproximadamente igual al tamaño de G. En las aplicaciones del problema, como por ejemplo búsqueda en una base de datos de k-personas relacionadas dos a dos, el grafo, i.e. la base de datos, es inmenso mientras que k es pequeño. Es natural

Key words and phrases. Máquinas de Turing, clases de complejidad, complejidad paramétrica, algoritmos eficientes, costos computacionales.

pensar que el análisis clásico del problema del *Clique*, que lleva a la prueba de su NP-completez y por lo tanto de su intratabilidad, es un análisis inadecuado porque ignora la naturaleza bidimensional de las instancias del problema, compuestas usualmente por una componente inmensa, el input propiamente dicho, y una componente relativamente pequeña, el parámetro. Considere la siguiente parametrización del problema del *Clique*.

Problema 2 ($p - Clique$)**.** .

- *Input:* (G, k)*, donde* G *es un grafo y* $k \in \mathbb{N}$*.*
- *Parámetro:* k*.*
- *Problema: Decida si* G *contiene un grafo completo de tamaño* k*, (i.e. un* k*-clique).*

Dado L un problema paramétrico, diremos que L es computable en tiempo fpt si y solo si existe un algorimo $\mathcal{M}$ que decide L y tal que el tiempo de cómputo de $\mathcal{M}$ en una instancia (x, k) está acotado por $f(k)\, p(|x|)$, donde f es una función computable y p es un polinomio.

La complejidad paramétrica intenta realizar un análisis paramétrico de los problemas parametrizables, como lo es por ejemplo el problema del *Clique*, proponiendo para ello una noción relajada de tratabilidad. Mientras que la noción canónica, (clásica), de tratabilidad es computabilidad en tiempo polinomial, la noción de tratabilidad en el contexto paramétrico es computabilidad en tiempo fpt. La complejidad paramétrica propone estudiar una noción paramétrica de reducción, a la que llamaremos p-Turing reducibilidad. La nueva noción de reducción permite a su vez definir nuevas clases de complejidad, las cuales permiten medir la complejidad (paramétrica) de los lenguajes parametrizables. Una clase paramétrica análoga a NP es la clase $W[1]$ (vea [7]) y un análogo paramétrico de la clase $\#P$ es la clase $\#W[1]$.

Nota 1. *El problema del Clique sigue siendo intratable desde el punto de vista paramétrico, p-Clique es* $W[1]$ *completo. Esto no quiere decir que todo problema intratable en el sentido clásico sigue siendo intratable en el sentido paramétrico, (si este fuera el caso ¿tendría sentido la complejidad paramétrica?), así por ejemplo el problema del cubrimiento de vértices (este problema parametrizable es usualmente llamado Vertex Cover (vea por ejemplo* [11]*)), es* NP *completo pero computable en tiempo* fpt*, esto es, intratable desde el punto de vista clásico pero tratable desde el punto de vista paramétrico.*

1.2. **Minería de grafos.** Intentaremos definir que es Minería de Datos en términos del tipo de problemas que se estudian en esta área de las Ciencias de la Computación. Sea X un objeto combinatorio como por ejemplo una base de datos o un grafo. Minar X significa extraer tanta información acerca de X como sea posible. El tipo de información que se quiere extraer puede ser de múltiples tipos, puede ser información estructural, (por ejemplo, una lista de axiomas que describen completamente la estructura), o puede ser información estadística, (que tipos de subestructura aparecen frecuentemente en X). En la práctica la minería de datos se ha concentrado en la extracción de información estadística y de manera mas específica se ha concentrado en el siguiente tipo de problema.

Problema 3 (Minería de datos, genérico)**.** .

- *Input: Una estructura X.*
- *Problema: Liste todas las subestructuras frecuentes y todos las estructuras infrecuentes en X.*

El problema genérico anteriormente descrito no admite mayores análisis, dado que su definición involucra varios términos que requieren ser precisados. Por ahora es suficiente notar que, en la práctica, el objeto X suele ser inmenso mientras que las subestructuras que se pide listar suelen ser relativamente pequeñas. El parágrafo anterior indica que, las diferentes variaciones del problema genérico de minería de datos que ocurren en la práctica, son problemas que admiten una parametrización natural. Es por ello que consideramos que la teoría de la complejidad paramétrica es la teoría que debemos usar si pretendemos analizar la complejidad computacional de este tipo de problemas.

La minería de grafos es una subárea de la minería de datos en la que el tipo de objetos combinatorios a minar, (el X del parágrafo anterior), son grafos. Esto (que en principio puede parecer una restricción demasiado fuerte) es solo una pequeña restricción, dado que la riqueza expresiva de los grafos nos permite codificar una amplia variedad de estructuras combinatorias como grafos.

1.3. **Organización del escrito.** El escrito consta de cuatro secciones incluyendo la introducción. En la sección dos se introducen los conceptos básicos, a saber, se revisan los conceptos fundamentales de la complejidad paramétrica, los conceptos fundamentales de arboricidad de grafos y de la lógica monádica de segundo orden. Adicionalmente se define el problema paramétrico a ser analizado, el cual llamamos $Gap - GM$ y el cual es una abstracción del problema de minar grafos en busca de patrones frecuentes.

En la sección tres empezamos discutiendo un resultado de Montoya [10] el cual afirma que $Gap - GM$ es equivalente al problema de aproximar una función en $\#W\,[1]$ dentro del rango 2 y como consecuencia $Gap - GM$ es intratable. Este resultado nos motiva a considerar restricciones del problema, en este escrito consideramos restricciones de $Gap - GM$ a clases de arboricidad acotada y se demuestra que estas restricciones son tratables en el sentido paramétrico. Finalmente en la sección 4 se proponen algunos problemas y se enuncian algunas conclusiones.

2. Preliminares

En esta sección se presentan los conceptos técnicos básicos que serán utilizados a lo largo del escrito. Adicionalmente se introduce un problema paramétrico que corresponde en buena medida al problema de minar grafos. Al final de la sección se introduce el concepto de arboricidad de un grafo y se discuten algunas de sus propiedades fundamentales.

2.1. **Complejidad paramétrica.** En esta sección introduciremos los conceptos básicos de la complejidad paramétrica.

Notación 1. $\{0,1\}^*$ *es el conjunto de las palabras de longitud finita en el vocabulario* $\{0,1\}$.

Definición 1. *Un lenguaje o problema paramétrico es un subconjunto de* $\{0,1\}^* \times \mathbb{N}$.

Ejemplo 1 (Evaluación de bases de datos)**.** .

- *Input:* (D, α)*, donde* D *es una base de datos y* α *una consulta.*
- *Parámetro:* $|\alpha|$*, donde* $|\alpha|$ *es el tamaño de la consulta, esto es, la longitud de la fórmula que expresa la consulta* α *en un lenguaje apropiado.*
- *Problema: Decida si* D *tiene la propiedad expresada por* α.

Nota 2. *Las instancias de todos los problemas que se considerarán en este escrito, así como las instancias del problema antes definido, son parejas constituidas por un número natural y un objeto combinatorio, como por ejemplo un grafo. Esto parece contradecir la definición de problema paramétrico que exige que las instancias de un tal problema estén constituidas por una palabra binaria y un número natural. Lo que sucede es que todo objeto combinatorio, (grafos, números, álgebras finitas, redes, nudos), puede ser codificado eficientemente como una palabra binaria. Nosotros usaremos una convención usual en Teoría de la Computación y en Complejidad Computacional, a saber, consideraremos las instancias de nuestros problemas como lo que son, (esto es, si son grafos como grafos si son matrices como matrices), y no nos preocuparemos por hacer explícita la codificación como palabras binarias que sabemos que existe.*

Definición 2. *Dado* L *un problema paramétrico, diremos que* L *es decidible en tiempo* fpt *si y solo si existe un algoritmo* $\mathbb{M}$ *tal que con input* (x,k)*:*

(1) $\mathbb{M}$ *decide correctamente si* (x,k) *pertenece a* L.
(2) *El tiempo de cómputo de* $\mathbb{M}$ *esta acotado por* $f(k)\,p(|x|)$*, donde* f *es una función computable,* p *es un polinomio y* $|x|$ *denota el tamaño de* x.

La clase $\mathcal{FPT}$ es la clase de todos los problemas decidibles en tiempo fpt. La clase $\mathcal{FPT}$ es el análogo paramétrico de la clase P, esto es, la clase $\mathcal{FPT}$ esta constituida por los problemas tratables en el sentido paramétrico.

Definición 3 (p-Turing reducibilidad)**.** *Dados* L *y* Σ *dos lenguajes paramétricos, diremos que* L *es p-Turing reducible a* Σ*, (en símbolos* $L \prec_T \Sigma$*), si y solo si existe un algoritmo* $\mathbb{M}$ *con acceso a un oráculo para* Σ *y tal que, con input* (x,k) :

(1) $\mathbb{M}$ *decide correctamente si el input* (x,k) *pertenece o no pertenece al lenguaje* L.
(2) *El tiempo de cómputo de* $\mathbb{M}$ *está acotado por* $f(k)\,p(|x|)$*, donde* f *es una función computable y* p *es un polinomio.*

(3) *Si durante la computación* $\mathbb{M}$ *consulta al oráculo acerca de la instancia* (y, r)*, entonces* $r \leq f(k)$.

La noción de p-Turing reducibilidad es el equivalente paramétrico de la noción clásica de Turing reducibilidad y al igual que esta nos permitirá: comparar diferentes problemas de acuerdo con su dificultad intrínseca; definir nuevas clases de complejidad, (paramétricas); clasificar los problemas paramétricos de acuerdo con su complejidad.

Definición 4. *Dados* L *y* Σ *dos lenguajes paramétricos, diremos que* L *es fpt reducible a* Σ*, (en símbolos* $L \prec_{fpt} \Sigma$*), si y solo si existe una p-Turing reducción de* L *en* Σ *tal que el oráculo es consultado una única vez en cada computación.*

La clase paramétrica que introduciremos a continuación es el análogo paramétrico de NP.

Definición 5. $W[1] = \{L : L$ *es un lenguaje paramétrico y* $L \prec_{fpt} p - Clique\}$.

(1) L *es* $W[1]$ *duro si y solo si* $p - Clique \prec_T L$.
(2) L *es* $W[1]$ *completo si y solo si* L *es* $W[1]$ *duro y* $L \in W[1]$.

En lo que sigue introduciremos una caracterización en términos de máquinas de la clase $W[1]$.

Definición 6 ($W[1]$-PRAM)**.** *Una* $W[1]$*-PRAM* $\mathbb{M}$ *es una máquina no determinística de acceso aleatorio tal que, con input* (x, k).

(1) *Realiza a lo mas* $f(k)\, p(|x|)$ *pasos. A lo mas* $f(k) \log(|x|)$ *de ellos no determinísticos, donde* f *es una función computable y* p *un polinomio.*
(2) *Usa a lo mas* $f(k)\, p(|x|)$ *registros.*
(3) *En todo momento de la computación los números guardados en cada uno de los registros son menores que* $f(k)\, p(|x|)$.
(4) *Los pasos no determinísticos están entre los últimos* $h(k) \log(|x|)$ *pasos de la computación, donde* h *es una función computable.*

Teorema 1. $L \in W[1]$ *si y solo si existe una* $W[1]$*-PRAM que decide* L.

Para mas información acerca del teorema el lector puede consultar [7]. A continuación introduciremos un tipo adicional de reducciones que llamaremos reducciones probabilísticas.

Definición 7 (Reducciones probabilísticas)**.** *Dados* L *y* Σ *dos lenguajes paramétricos, diremos que* L *probabilísticamente reducible a* Σ*, (en símbolos* $L \prec_r \Sigma$*), si y solo si existe un algoritmo probabilístico* $\mathbb{M}$ *con acceso a un oráculo para* Σ *y tal que, con input* (x, k) :

(1) *El número de bits aleatorios utilizados por* $\mathbb{M}$ *está acotado por* $f(k) \log(|x|)$*, donde* f *es una función computable.*

(2) *El tiempo de cómputo de* $\mathbb{M}$ *está acotado por* $f(k)\,p(|x|)$*, donde* p *es un polinomio.*
(3) *Si durante la computación* $\mathbb{M}$ *consulta al oráculo acerca de la instancia* (y,r)*, entonces* $r \leq f(k)$.
(4) *Si* $(x,k) \in L$*, entonces* $\Pr_r[\mathbb{M}$ *acepte* $(x,k)] \geq \frac{3}{4}$*, la probabilidad es calculada sobre las escogencias aleatorias de* $\mathbb{M}$.
(5) *Si* $(x,k) \notin L$*, entonces* $\Pr_r[\mathbb{M}$ *acepte* $(x,k)] \leq \frac{1}{4}$.

Para finalizar introduciremos la noción de problema paramétrico de conteo y algunas nociones propias de la complejidad paramétrica de problemas de conteo.

Definición 8. *Un problema paramétrico de conteo es una función computable* $f : \{0,1\}^* \times \mathbb{N} \to \mathbb{N}$.

Ejemplo 2 $(p - \#Clique)$**.** .

- *Input:* (G,k), *donde* G *es un grafo y* $k \in \mathbb{N}$.
- *Parámetro:* k.
- *Problema: Calcule el número de* k*-Cliques en* G.

La noción de p-Turing reducibilidad entre problemas de conteo es la adaptación natural de la noción de p-Turiing reducibilidad al contexto de problemas de conteo.

Definición 9 (p-Turing reducibilidsd entre problemas de conteo)**.** *Dados* χ *y* ψ *dos problemas paramétricos de conteo, diremos que* χ *es* p*-Turing reducible a* ψ*, (en símbolos* $\chi \prec_T \psi$*), si y solo si existe un algoritmo* $\mathbb{M}$ *con acceso a un oráculo para* ψ *y tal que, con input* (x,k) :

(1) $\mathbb{M}$ *calcula correctamente* $\chi(x,k)$.
(2) *El tiempo de cómputo de* $\mathbb{M}$ *está acotado por* $f(k)\,p(|x|)$*, donde* f *es una función computable y* p *es un polinomio.*
(3) *Si durante la computación* $\mathbb{M}$ *consulta al oráculo acerca de la instancia* (y,r)*, entonces* $r \leq f(k)$.

La noción de reducción entre problemas de conteo nos permite definir clases de problemas de conteo. A continuación introduciremos la clase $\#W[1]$ la cual es el análogo paramétrico de $\#P$ (ver [6]).

Definición 10. $\#W[1] = \left\{ \begin{array}{c} \psi : \psi \text{ es un problema paramétrico de conteo y} \\ \psi \prec_T p - \#Clique \end{array} \right\}$.

Mucha mas información acerca de la teoría de la complejidad paramétrica puede ser encontrada en [5] y [7].

2.2. **El problema de minar grafos, primeras observaciones y definición del problema.** El objetivo de esta sección es presentar una definición formal del problema que queremos analizar.

Notación 2. .

(1) *Dado S un conjunto* $[S]^2 = \{A \subset S : |A| = 2\}$.
(2) *Dado $n \in \mathbb{N}$, $[n]$ denotará el conjunto* $\{1, 2, ..., n\}$.
(3) *Dado $k \in \mathbb{N}$, K_k es el grafo completo con k vértices.*

Recuerde primero que un grafo G es un par $(V(G), E(G))$, donde $V(G)$ es un conjunto, el conjunto de vértices de G, y $E(G) \subset [V(G)]^2$, $E(G)$ será llamado el conjunto de aristas del grafo G. En lo que sigue, dado G un grafo, si $|V(G)| = N$, asumiremos que $V(G) = [n]$. El tamaño de G será igual a $|V(G)|$, en ocasiones usaremos el símbolo $|G|$ para denotar el tamaño de G.

Dados G y H dos grafos, diremos que G y H son *isomorfos* si y solo si existe una biyección $g : V(G) \to V(H)$ tal que para todo par de vértices $v, w \in V(G)$, $\{v, w\} \in E(G)$ si y solo si $\{g(v), g(w)\} \in E(H)$. Dados G y H, usaremos el símbolo $G \simeq H$ para indicar que G y H son isomorfos.

Dados G y H dos grafos, *el número de ocurrrencias de H como subgrafo de G* es igual a

$$\left|\left\{(W, F) : W \subset V(G) \ \& \ F \subset E \cap [W]^2 \ \& \ H \simeq (W, F)\right\}\right|$$

Dado $c \in \mathbb{N}$, diremos que H es *c-frecuente* en G si y solo si el número de ocurrencias de H como subgrafo de G es por lo menos $2c$. Por otro lado diremos que H es *c-excepcional* si y solo si el número de ocurrencias de H como subgrafo de G es a lo más c.

Problema 4 (Minería exacta)**.** .

- *Input: Un grafo G y dos números naturales c y k.*
- *Parámetro: k.*
- *Problema: Liste todos los subgrafos c-frecuentes y todos los subgrafos c-excepcionales de G.*

El problema anteriormente definido es precisamente el problema que quisieramos analizar. Lo primero que podemos notar es que el problema de minería exacta es al menos tan difícil como el problema $p - \#Clique$, el cual es bien sabido es $\#W[1]$ completo (vea [6]). Para verificar la anterior observación es suficiente considerar el siguiente argumento.

Sea (G, k) una instancia de $p - \#Clique$, para contar el número de k-cliques en G es suficiente calcular $c \in \mathbb{N}$ tal que, K_k es c-excepcional pero no es $(c-1)$-excepcional. Note simplemente que $c - 1$ es igual al número de $k - cliques$ en G. Para terminar es suficiente anotar que es posible calcular tal c, en tiempo polinomial en $|G|$, usando búsqueda binaria.

El parágrafo anterior indica que el problema de minería exacta es duro para $\#W[1]$. Un problema duro para una clase de problemas de conteo como $\#W[1]$ es un problema muy difícil. En el contexto clásico se sabe que $\#P$ contiene la jerarquía polinomial, este es el famoso teorema de Toda [12], esto quiere decir que si L es duro para $\#P$, entonces L es mas difícil que todo problema en la jerarquía polinomial y por lo tanto ω veces mas difícil que todo problema en NP, (si la jerarquía polinomial no colapsa). Aunque no se conoce un análogo del teorema de Toda en el contexto paramétrico, se conjetura que $\#W[1]$ contiene toda la jerarquía W (ver [6]). Así pues, si la conjetura es cierta, el problema de minería exacta es ω veces mas difícil que todo problema en $W[1]$, (si la jerarquía W no colapsa), lo que lo convierte en un problema extremadamente difícil mas allá de nuestras posibilidades de análisis.

El problema de minería exacta es difícil porque en el fondo él es un problema de conteo. Cuando se tiene que lidiar con un problema de conteo se tienen dos alternativas, intentar resolverlo de manera exacta, lo cual no suele ser posible, o intentar resolverlo de manera aproximada. En algunos casos es posible resolver eficientemente la versión aproximada de un problema de conteo aunque la versión exacta sea intratable (para detalles ver [1]). Es por ello que proponemos considerar la siguiente versión, aproximada, del problema de minar un grafo.

Problema 5 (Minería aproximada)**.** .

- *Input: Un grafo G y dos números naturales c y k.*
- *Parámetro: k.*
- *Problema: Calcule dos listas L_F y L_E de manera tal que L_F contenga todos los grafos c-frecuentes de tamaño k en G y L_E contenga todos los grafos c-excepcionales.*

Un algoritmo que resuelva el problema es un algoritmo que con input (G, k, c) produce dos listas disyuntas. La primera lista es una lista que contiene todos los subgrafos c-frecuentes de G de tamaño k y que puede contener otros grafos además de los c-frecuentes. La segunda lista, por su parte, ha de contener todos los subgrafos c-excepcionales de $\mathcal{G}$ de tamaño k y, al igual que la primera lista puede contener errores, esto es, grafos que no son c-excepcionales. Sea $\mathcal{GRAPHS}(k)$ una lista que contiene todos los grafos de tamaño k salvo isomorfismo. Lo primero que podemos notar es que el tamaño de $\mathcal{GRAPHS}(k)$ depende únicamente de k. Es por ello que, el problema de listado minería aproximada, es equivalente al siguiente $Gap - problem$.

Problema 6 ($Gap - GM$)**.** .

- *Input: (G, H, k, c). G y H son grafos, $|H| = k$ y $k, c \in \mathbb{N}$.*
- *Parámetro: k.*
- *Instancias SI: (G, H, k, c) es una instancia SI si y solo si H es c-frecuente.*
- *Instancias NO: (G, H, k, c) es una instancia NO si y solo si H es c-excepcional.*

Un algoritmo $\mathbb{M}$ que resuelva el problema $Gap - GM$. es un algoritmo que:

(1) A toda instancia SI, $\mathbb{M}$ la clasifica como un SI.

(2) A toda instancia NO, $\mathbb{M}$ la clasifica como un NO.
(3) Puede errar en toda otra instancia.

El que los problemas $Gap - GM$ y minería aproximada sean equivalentes nos permite analizar $Gap - GM$ en lugar de minería aproximada

2.3. **Arboricidad de grafos.** El concepto de Arboricidad de grafos, (Treewidth), es uno de los conceptos en teoría de grafos con mayores aplicaciones algorítmicas. La arboricidad de un grafo G permite medir que tan lejos está G de ser un árbol y a su vez permite medir la posibilidad de usar en G algoritmos específicos de árboles. Si G no es un árbol, no es posible en principio usar en G algoritmos de árboles. Lo que sucede es que asociada a la arboricidad de G aparece una construcción combinatoria que llamaremos el *Árbol de Courcelle de* G. $\mathbb{T}_{C,G}$ el árbol de Courcelle de G, es un árbol que puede ser utilizado para resolver preguntas acerca de G usando algoritmos para árboles sobre $\mathbb{T}_{C,G}$.

Dado G un grafo, una descomposición de G de orden w es un par $\left(\mathcal{T}, (\overrightarrow{a_t})_{t\in\mathcal{T}}\right)$, donde $\mathcal{T}$ es un árbol y para todo $t \in \mathcal{T}$, $\overrightarrow{a_t}$ es una tupla de vértices de G de longitud a lo más $w+1$. Adicionalmente se satisfacen las siguientes condiciones:

(1) Para todo $v \in V(G)$, el conjunto $\{t \in \mathcal{T} : v \text{ ocurre en } \overrightarrow{a_t}\}$ es no vacío y conexo.
(2) Para todo $\{v, w\} \in E(G)$, existe $t \in \mathcal{T}$ tal que v y w ocurren en $\overrightarrow{a_t}$.
(3) Todo vértice t de $\mathcal{T}$ satisface la siguiente condición: O t es un *join* o t es un *separador*. Si t es un separador, t tiene un único sucesor, llamémoslo s, tal que $\overrightarrow{a_t}$ es una subtupla de $\overrightarrow{a_s}$. Por otro lado si t es un join, t tiene dos sucesores, llamémoslos s y u, tales que $\overrightarrow{a_t} = \overrightarrow{a_s} \cup \overrightarrow{a_u}$.

Definición 11. .

(1) *Dado G un grafo, la arboricidad de G es igual a*

$$\min\{r \in \mathbb{N}: \textit{ existe una descomposición de } G \textit{ de orden } r\}$$

(2) *Dada $\mathcal{K}$ una clase de grafos, $\mathcal{K}$ es una clase de arboricidad acotada por r si y solo si para todo $G \in \mathcal{K}$, la arboricidad de G es a lo más r.*

Ejemplo 3. *Si G es un árbol, la arboricidad de G es* 1. *Si $\mathcal{K}$ es la clase de todos los bosques, $\mathcal{K}$ es una clase de arboricidad acotada por* 1.

El siguiente teorema es fundamental en la teoría de arboricidad de grafos dado que el asegura que es posible calcular descomposiciones adecuadas en tiempo fpt.

Teorema 2. *Existe un algoritmo $\mathbb{M}$ que, con input un grafo G, calcula una descomposición de orden w, donde w es la arboricidad de G. Además el tiempo de cómputo de este algoritmo, en el input G, está acotado por $2^{p(w)}|G|$, donde p es un polinomio.*

Para una prueba el lector puede consultar [5].

2.4. **Lógica monádica de segundo orden.** La lógica monádica de segundo orden es una lógica de gran poder expresivo, (existen problemas NP-completos expresables en esta lógica). La lógica monádica de segundo orden es una lógica de segundo orden en la que todas las variables de segundo orden son monádicas. Veamos una definición precisa de esta lógica. Primero definiremos los términos. Sea τ un vocabulario, los τ términos los definimos recursivamente de la siguiente manera:

(1) Si x es una variable de primer orden, x es un término de primer orden.
(2) Si X es una variable monádica de segundo orden X es un término de segundo orden.
(3) Si f es una función r-aria en el vocabulario τ y $t_1, ..., t_r$ son términos de primer orden, entonces $f(t_1, ..., t_r)$ es un término de primer orden.

Definidos los términos podemos pasar a definir las fórmulas atómicas:

(1) Si t, t^* son términos de primer orden, $t = t^*$ es una fórmula atómica.
(2) Si T, T^* son términos de segundo orden, $T = T^*$ es una fórmula atómica.
(3) Si T es un término de segundo orden y t es un término de primer orden, entonces $T(t)$ es una fórmula atómica.

Definidas las fórmulas atómicas, podemos pasar a definir las fórmulas:

(1) Si α es una fórmula atómica, α es una fórmula.
(2) Si α y β son fórmulas, entonces $\alpha \wedge \beta$, $\alpha \vee \beta$ y $\sim \alpha$ son fórmulas.
(3) Si $\alpha(x, X)$ es una fórmula, entonces $\exists x \alpha(x, X)$, $\exists X \alpha(x, X)$, $\forall x \alpha(x, X)$ y $\forall X \alpha(x, X)$ son fórmulas.

Una fórmula de la lógica monádica de segundo orden será llamada una MSO fórmula. Dada G una τ-estructura y dada α una MSO fórmula en el vocabulario τ, $G \models \alpha$ es la noción de satisfacción de Tarski para la lógica de segundo orden restringida a MSO fórmulas.

Veamos un ejemplo de un problema NP completo expresable en la lógica de segundo orden. El problema $3 - COL$ es el problema defindo por:

Problema 7 $(3 - COL)$**.** *3-coloreabilidad de grafos*

- *Input: Un grafo G.*
- *Problema: Decida si G es 3-coloreable.*

El problema $3 - COL$ es NP completo (vea [11]), sea $\varphi_{3-COL}(X_1, X_2, X_3, x, y)$ la MSO fórmula

$$\exists X_1 X_2 X_3 \left(\begin{array}{c} (\forall x (X_1(x) \vee X_2(x) \vee X_3(x))) \wedge \\ \left(\forall x \forall y \left(\bigwedge_{i \leq 3} (X_i(x) \wedge E(x, y)) \Rightarrow \sim (X_i(y)) \right) \right) \end{array} \right)$$

Es fácil verificar que, para todo grafo G, $G \models \varphi_{3-COL}(X_1, X_2, X_3, x, y)$ si y solo si G es tres coloreable.

Dado que existen problemas NP-completos expresables en la lógica monádica de segundo orden, el problema de verificación $MSO-Modelchecking$ es NP completo, donde $MSO - Modelchecking$ es el problema definido por

Problema 8 ($MSO - Modelchecking$)**.** .

- *Input:* (G, φ)*, donde* G *es un grafo y* φ *es una* MSO *fórmula sin variables libres.*
- *Problema: Decida si* $G \models \varphi$.

Es importante anotar que este mismo problema restringido a árboles es fácil, en particular, si fijamos φ una MSO fórmula sin variables libres, i.e. una sentencia. Observe que el siguiente problema es computable en tiempo lineal.

Problema 9 (φ-verificación (árboles))**.** .

- *Input: Un arbol* $\mathcal{T}$.
- *Problema: Decida si* $\mathcal{T} \models \varphi$.

Así pues, la verificación de MSO fórmulas sobre grafos es en general un problema difícil, mientras que la verificación de MSO fórmulas sobre árboles es un problema fácil. El concepto de arboricidad de grafos nos permitirá trasladar los resultados positivos acerca de árboles, i.e. la existencia de verificadores eficientes para MSO fórmulas, al contexto más general de clases de grafos de arboricidad acotada. Esta aplicación del concepto de arboricidad la usaremos fuertemente en la última sección de este escrito, lo que nos permitirá resolver el problema de minería exacta restringido a clases de arboricidad acotada.

3. Minería de grafos y conteo aproximado

En la sección anterior hemos introducido el problema $Gap - GM$ (problema 6) el cual corresponde en buena medida al problema de minar grafos de manera aproximada, lo que nos proponemos hacer a continuación es analizar este problema

3.1. Conteo aproximado y minado aproximado.

En investigaciones previas (ver [10]) se probó que existe una profunda relación entre el problema $Gap - GM$ y el problema de aproximar un problema de conteo en $\#W$ [1] dentro del rango 2. Veamos primero algunas definiciones.

Definición 12. *Dado* χ *un problema de conteo parametrizado y dado* $c \gneqq 1$*, un* $fpt-algoritmo$ $\mathbb{M}$ *aproxima* χ *dentro del rango* c *si y solo si para toda instancia* (x, k) *de*χ

$$\chi(x,k)\frac{1}{c} \leq \mathbb{M}(x,k) \leq \chi(x,k)\, c$$

donde $\mathbb{M}(x,k)$ *es el output de* $\mathbb{M}$ *en el input* (x,k).

Considere el siguiente problema de conteo.

Problema 10 ($p - \#EMB$)**.** .

- *Instancia:* (G, H, k)*, G y H son grafos,* $|H| = k$ *y* $k \in \mathbb{N}$.

- *Parámetro: k.*
- *Problema: Calcule el número de ocurrencias de H en G.*

Lema 1. *El problema $p-\#EMB$ es $\#W[1]$ completo.*

Proof. Note simplemente que $p-\#Clique$ esta contenido en $p-\#EMB$ □

Considere ahora el siguiente problema de conteo aproximado

Problema 11 ($p-\#_2EMB$)**.** .

- *Instancia: (G,H,k), G y H son grafos, $|H|=k$.*
- *Parámetro: k.*
- *Problema: Aproxime $|\{R\subset G: R\simeq H\}|$ dentro del rango 2.*

Teorema 3. *Los problemas $Gap-GM$ y $p-\#_2EMB$ son p-Turing equivalentes.*

Para una prueba del teorema el lector puede consultar Montoya, [10].

3.2. **Estudiando restricciones.** Del problema $Gap-GM$ podemos considerar diferentes restricciones. En algunas aplicaciones podemos suponer que el grafo input G no es un grafo arbitrario sino que pertenece a una clase especial de grafos. En otras aplicaciones podemos suponer que el grafo parámetro H no es un grafo arbitrario sino que pertenece a alguna clase especial de grafos. Dada $\mathcal{K}$ una clase de grafos, podemos considerar dos restricciones de $Gap-GM$ determinadas por $\mathcal{K}$.

Problema 12 ($Gap-GM(\mathcal{K})$)**.** .

- (G,H,k,c). *G y H son grafos, $|H|=k$, $G\in\mathcal{K}$ y $c\in\mathbb{N}$.*
- *Parámetro: k.*
- *Instancias SI: (G,H,k,c) es una instancia SI si y solo si H es c-frecuente.*
- *Instancias NO: (G,H,k,c) es una instancia NO si y solo si H es c-excepcional.*

Problema 13 ($Gap-GM[\mathcal{K}]$)**.** .

- (G,H,k,c). *G y H son grafos, $|H|=k$, $H\in\mathcal{K}$ y $c\in\mathbb{N}$.*
- *Parámetro: k.*
- *Instancias SI: (G,H,k,c) es una instancia SI si y solo si H es c-frecuente.*
- *Instancias NO: (G,H,k,c) es una instancia NO si y solo si H es c-excepcional.*

Un problema de importancia en la práctica y aún por resolver es el problema de *cartografiar* la intratabilidad del problema $Gap-GM$, esto es:

(1) Caracterizar las clases $\mathcal{K}$ para las cuales $Gap-GM(\mathcal{K})$ es tratable.
(2) Caracterizar las clases $\mathcal{K}$ para las cuales $Gap-GM[\mathcal{K}]$ es tratable.

Una manera, que consideramos viable, de enfrentar este problema es explotar la relación existente entre $Gap - GM$ y $p - \#EMB$. Dada $\mathcal{K}$ una clase de grafos, podemos definir dos restricciones de $p - \#EMB$ determinadas por $\mathcal{K}$.

Problema 14 ($p - \#_2EMB(\mathcal{K})$)**.** .

- *Instancia:* (G, H, k), *G y H son grafos,* $|H| = k$ *y* $G \in \mathcal{K}$.
- *Parámetro:* k.
- *Problema: Aproxime* $|\{R \subset G : R \simeq H\}|$ *dentro del rango 2.*

Problema 15 ($p - EMB[\mathcal{K}]$)**.** .

- *Instancia:* (G, H, k), *G y H son grafos,* $|H| = k$ *y* $H \in \mathcal{K}$.
- *Parámetro:* k.
- *Problema: Aproxime* $|\{R \subset G : R \simeq H\}|$ *dentro del rango 2.*

Es importante anotar que, en las reducciones presentadas en Montoya, [10], los grafos input y grafos parámetro no se transforman, esto es, en las reducciones presentadas si se parte, por ejemplo, de una instancia $((G, H, k, c))$ de $Gap-GM$ y $G, H \in \mathcal{K}$, entonces todas las instancias calculadas en la reducción de $Gap-GM$ en $p-EMB$ son de la forma (G^*, H^*, k^*) con $G^*, H^* \in \mathcal{K}$. Esto nos permite afirmar que:

(1) Para toda clase $\mathcal{K}$, $Gap - GM(\mathcal{K})$ es tratable si y solo si $p - \#_2EMB(\mathcal{K})$ es tratable.
(2) Para toda clase $\mathcal{K}$, $Gap - GM[\mathcal{K}]$ es tratable si y solo si $p - \#_2EMB[K]$ es tratable.

Esto a su vez nos permite reducir el problema de cartografiar la tratabilidad de $Gap-GM$ a cartografiar la tratabilidad de aproximar $p-\#EMB$ dentro del rango 2. Lo interesante del cuento es que, aunque no se ha cartografiado la tratabilidad de $p-\#EMB$, existe trabajo (ver por ejemplo [9] o [3]) en esta dirección. Cerraremos este escrito proponiendo una conjetura, la cual si fuera resuelta de manera positiva, nos permitiría resolver el problema antes propuesto.

Conjetura 1. .

(1) $p - \#_2EMB(\mathcal{K})$ *es tratable si y solo si* $\mathcal{K}$ *es una clase de arboricidad acotada.*
(2) $p - \#_2EMB[\mathcal{K}]$ *es tratable si y solo si* $\mathcal{K}$ *es una clase de arboricidad acotada.*

3.3. **Minando grafos en clases de arboricidad acotada.** En esta sección estudiaremos una dirección de la conjetura enunciada al final de la sección anterior, específicamente probaremos que $p - \#_2EMB(\mathcal{K})$ y $p - \#_2EMB[\mathcal{K}]$ son tratables si $\mathcal{K}$ es una clase de arboricidad acotada.

3.3.1. *Resolviendo $p-\#_2EMB[\mathcal{K}]$ en clases de arboricidad acotada.* En esta subsección probaremos que, si $\mathcal{K}$ es una clase de arboricidad acotada, los problemas: $p-\#_2EMB[\mathcal{K}]$, $Gap-GM[\mathcal{K}]$ y minería aproximada restringida a $\mathcal{K}$ son computables en tiempo fpt de manera probabilística.

Problema 16 ($\#HOM[\mathcal{K}]$)**.** .

- *Input:* (G,H)*, donde* G *y* H *son grafos con el mismo conjunto de etiquetas y* $H\in\mathcal{K}$*.*
- *Problema: Calcule* $\#HOM(G,H):=|\{f:V(H)\to V(G):f \text{ es un homomorfismo}\}|$.

Lema 2. *Si* K *es una clase de arboricidad acotada el problema* $\#HOM[\mathcal{K}]$ *puede ser resuelto en tiempo polinomial en* $|H|\,|G|$.

Proof. Sea w la arboricidad de $\mathcal{K}$ y sea (G,H) una instancia del problema $\#HOM[\mathcal{K}]$. Sea $\left(\mathcal{T},\ (\overrightarrow{a_t})_{t\in\mathcal{T}}\right)$ una descomposición de H de orden w.

Dado $t\in\mathcal{T}$, H_t es igual al subgrafo inducido de H con universo

$$\{v\in V(H):\text{Existe } t_0\in\mathcal{T}_t \text{ tal que } v \text{ ocurre en } \overrightarrow{a_{t_0}}\}$$

Dado $t\in\mathcal{T}$ tal que $\overrightarrow{a_t}$ es una tupla de longitud $r\leq w+1$ y dada $\overrightarrow{b}$ una r-tupla de elementos de G, $\#_{\overrightarrow{b}}(t)$ es igual al número de homomorfismos de H_t en G que envía la tupla $\overrightarrow{a_t}$ en la tupla $\overrightarrow{b}$, i.e. para todo $i\leq r$, $a_{t_i}=b_i$. Note que si r_o es la raíz de $\mathcal{T}$, entonces H_{r_0} es igual a H, note también que si $\overrightarrow{a_t}$ es una tupla de longitud α, entonces $\#HOM(G,H)=\sum_{\overrightarrow{b}\in G^\alpha}\#_{\overrightarrow{b}}(r_0)$. La idea del algoritmo consiste en utilizar una estrategia de programación dinámica para calcular $\#_{\overrightarrow{b}}(t)$ para cada $t\in\mathcal{T}$ y cada tupla $\overrightarrow{b}$ de la longitud adecuada. Note que esto es suficiente para resolver el problema, dado que si logramos calcular $\#_{\overrightarrow{b}}(r_0)$ para todo $\overrightarrow{b}\in G^\alpha$, podemos calcular $\sum_{\overrightarrow{b}\in G^\alpha}\#_{\overrightarrow{b}}(r_0)$ en tiempo $|G|^\alpha$.

Dado $t\in\mathcal{T}$, $\alpha(t)$ es igual a la longitud de la tupla $\overrightarrow{a_t}$.

Como el algoritmo es un algoritmo de programación dinámica, empezamos por las hojas. Dada t una hoja de $\mathcal{T}$, es posible calcular $\#_{\overrightarrow{b}}(t)$ en tiempo $O(w)$ para toda tupla $\overrightarrow{b}$ de longitud $\alpha(t)$. Esto implica que el tiempo de cómputo necesario para calcular la tabla de t es $O(|G|^w)$, donde la tabla de t es el conjunto de valores

$$\left\{\#_{\overrightarrow{b}}(t):\overrightarrow{b}\text{ es una tupla de longitud }\alpha(t)\right\}$$

Sea t un vértice de $\mathcal{T}$, supongamos que hemos logrado calcular $\#_{\overrightarrow{b}}(s)$ para todo $s\in\mathcal{T}_t$ y para toda tupla $\overrightarrow{b}$ de longitud $\alpha(t)$. Consideramos dos casos, a saber: el caso 1 es cuando t es un separador y el caso 2 cuando t es un join.

Caso 1: Supongamos que t es un separador y que s es su único hijo, supongamos además que $\alpha(t)\leq w+1$ y que $\overrightarrow{a_t}$ es una subtupla de $\overrightarrow{a_s}$. Sea $\overrightarrow{b}$ una tupla de elementos de G de longitud $\alpha(t)$, note que $\#_{\overrightarrow{b}}(t)=\sum_{\overrightarrow{b}\sqsubset\overrightarrow{c}}\#_{\overrightarrow{c}}(s)$. Lo anterior nos permite asegurar que el tiempo de cómputo necesario para calcular la tabla de t es $O(|G|^w)$, suponiendo calculadas la tabla de su sucesor.

Caso 2: Supongamos que t es un join y que s,u son sus dos hijos, supondremos además que $\overrightarrow{a_t}=\overrightarrow{a_s}\cup\overrightarrow{a_u}$. Sea $\overrightarrow{b}$ una tupla de longitud $\alpha(t)$ y sean $\overrightarrow{b_s}$ y $\overrightarrow{b_u}$ tuplas tales que $\overrightarrow{b}=\overrightarrow{b_s}\cup\overrightarrow{b_u}$. Note que $\#_{\overrightarrow{b}}(t)=\#_{\overrightarrow{b}_s}(s)\,\#_{\overrightarrow{b}_u}(u)$. Lo anterior implica que el tiempo de cómputo necesario para calcular la tabla de t es $O(|G|^w)$.

En el peor de los casos posibles, para calcular la tabla de r_0 necesitamos tiempo de cómputo $O\left(|\mathcal{T}|\,|G|^w\right)$, además calculada la tabla de r_0 podemos calcular el valor de $\#HOM\left(G,H\right)$ en tiempo $O\left(|G|^w\right)$. Finalmente recuerde que, dado H un grafo del cual se sabe que tiene arboricidad a lo más w, es posible calcular una descomposición $\left(\mathcal{T}\,,\ (\overrightarrow{a_t})_{t\in\mathcal{T}}\right)$, de orden w, en tiempo $2^{p(w)}\,|H|$, lo que de paso implica que $|\mathcal{T}|$ es menor o igual que $2^{p(w)}\,|H|$. Podemos entonces concluir que el tiempo de cómputo del algoritmo es $O\left(2^{p(w)}\,|H|\,|G|^w\right)$, lo que implica que el problema $\#HOM\left[\mathcal{K}\right]$ puede ser resuelto en tiempo $O\left(\left(|H|\,|G|\right)^w\right)$. □

Nota 3. *Dalmau y Jonsson en* [3] *probaron algo más fuerte, a saber:*

(1) *Si $\mathcal{K}$ es una clase de arboricidad acotada, entonces $\#HOM\left[\mathcal{K}\right]\in P$.*
(2) *Si $\mathcal{K}$ es una clase de arboricidad no acotada la versión paramétrica de $\#HOM\left[\mathcal{K}\right]$, (Dada una instancia (G,H), el parámetro es $|H|$), es $\#W\left[1\right]$ completo.*

Un grafo etiquetado es una tripla (G,λ,Σ), donde G es un grafo, Σ es un conjunto, (el conjunto de etiquetas), y λ es una función de $V\left(G\right)$ en Σ. Dados (G,λ,Σ) y (H,ν,Σ) dos grafos etiquetados con el mismo conjunto de etiquetas, un homomorfismo de (G,λ,Σ) en (H,ν,Σ) es una función g de $V\left(G\right)$ en $V\left(H\right)$ tal que para todo $u\in V\left(G\right)$, $\lambda\left(u\right)=\nu\left(g\left(u\right)\right)$.

Sea $\mathcal{K}$ una clase de arboricidad acotada. Considere el siguiente problema

Problema 17 (Embebimientos-saturados[$\mathcal{K}$])**.** .

- *Input: Dos grafos etiquetados $(G,\lambda,V\left(H\right))$ y $(H,id,V\left(H\right))$, id es la función identidad de $V\left(H\right)$ en $V\left(H\right)$.*
- *Problema: Calcule el número de embebimientos de H en G.*

Proposición 1. *Una función g de $V\left(H\right)$ en $V\left(G\right)$ es un homomorfismo de $(H,id,V\left(H\right))$ en $(G,\lambda,V\left(H\right))$ si y solo si es un embebimiento.*

Corolario 1. *El problema Embebimientos-saturados[$\mathcal{K}$], problema 17 puede ser resuelto en tiempo polinomial en $|H|\,|G|$.*

Lema 3 (Cotas de Chernoff)**.** *Sea S una suma de varibles aleatorias indicadoras mutuamente independientes, sea ρ igual al valor esperado de S. Entonces para todo $\epsilon\gneqq 0$.*

$$\Pr\left(|S-\rho|\gneqq\epsilon\rho\right)\lneqq 2\exp\left(-c\left(\epsilon\right)\rho\right)$$

donde $c\left(\epsilon\right)=\min\left\{\left(1+\epsilon\right)\ln\left(1+\epsilon\right)-\epsilon,\frac{\epsilon^2}{2}\right\}$.

Una prueba puede ser consultada en Alon and Spencer, [2].

Definición 13. *Sea ψ un problema de conteo paramétrico. Un fptras es un algoritmo probabilístico $\mathbb{M}$ tal que, para toda instancia (x,k) de ψ y para todo $l \in \mathbb{N}$, $\mathbb{M}$ calcula un número z tal que*

$$\Pr\left(\left(1-\frac{1}{l}\right)\psi(x,k) \leq z \leq \left(1-\frac{1}{l}\right)\psi(x,k)\right) \geq \frac{3}{4}$$

Además el tiempo de cómputo de $\mathbb{M}$ está acotado por $f(k)\,p(|x|)$, donde f es una función computable y p es un polinomio.

Note que un *fptras* para ψ permite calcular de manera probabilística aproximaciones para ψ dentro del rango 2, i.e. dada $\mathcal{K}$ una clase de grafos un *fptras* para $p-\#EMB[\mathcal{K}]$ nos permite resolver probabilísticamente el problema $p-\#_2EMB[\mathcal{K}]$.

Teorema 4. *Si $\mathcal{K}$ es una clase de arboricidad acotada, existe un fptras $\mathbb{M}$ para $p-\#EMB[\mathcal{K}]$.*

Proof. Sea (G,H) una instancia de $p-\#EMB[\mathcal{K}]$, el algoritmo $\mathbb{M}$ repite lo siguiente r veces, (r será determinado más adelante): $\mathbb{M}$ aleatoriamente escoge un etiquetado $\lambda : V(G) \to V(H)$ y calcula el número de embebimientos de $(H, id, V(H))$ en $(G, \lambda, V(H))$. Dado $i \in \{1,...,r\}$ y dado z_i el número de embebiminetos calculado en la i-esima iteración $\mathbb{M}$ retorna $z = \sum_{i\leq r} z_i$.

Sean $k = |H|$, $n = |G|$, $p_k = \frac{k^k}{k!r}$, $\epsilon = \frac{1}{l}$, $c = c(\epsilon)$ y $r = \left\lceil \frac{k\ln(8n)}{cp_k} \right\rceil$.

Suponga que el algoritmo escoge la secuencia de etiquetados $\lambda_1,...,\lambda_r$. Sea g un embebimiento de H en G y sea X_i^g la variable indicadora del evento: *g es un embebimiento de $(H, id, V(H))$ en $(G, \lambda_i, V(H))$*. Note que $\Pr(X_i^g = 1) = p_k$. Si S^g es la suma $\sum_{i\leq r} X_i^g$, su valor esperado $E[S^g] = \mu$ es igual a $p_k r$. El lema de Chernoff implica que

$$\Pr\left(|S^g - \mu| \gtrneqq \epsilon\mu\right) \lneqq 2\exp(-c\mu)$$

Esto a su vez implica que $\Pr(|S^g - \mu| \gtrneqq \epsilon\mu \text{ para algun } g)$ es a lo mas $\frac{1}{4}$, dado que

$$\begin{aligned}\Pr(|S^g - \mu| \gtrneqq \epsilon\mu \text{ para algun } g) &\leq \\ \sum_g \Pr(|S^g - \mu| \gtrneqq \epsilon\mu) &\leq \\ n^k 2\exp(-cp_k r) &\leq \frac{1}{4}\end{aligned}$$

Entonces con probabilidad al menos $\frac{3}{4}$ se tiene que para todo g

$$\mu(1-\epsilon) \leq S^g \leq \mu(1+\epsilon)$$

Note que para todo $i \leq r$ se tiene que $z_i = \sum_g X_i^g$, esto implica que $\sum_{i\leq r} z_i = \sum_g S^g$. Tenemos entonces que

$$z = \frac{k^k}{k!r}\sum_{i\leq r} z_i = \left(\frac{1}{\mu}\right)\sum_g S^g$$

Entonces con probabilidad al menos $\frac{3}{4}$

$$\sum_g (1-\epsilon) \leq z \leq \sum_g (1+\epsilon)$$

Y dado que ϵ es igual a $\frac{1}{l}$ podemos concluir que

$$\#EMB(G,H)\left(1-\frac{1}{l}\right) \leq z \leq \#EMB(G,H)\left(1+\frac{1}{l}\right)$$

□

Corolario 2. *Si $\mathcal{K}$ es una clase de arboricidad acotada, existe un algoritmo probabilístico $\mathbb{M}$ tal que dada (G,H) una instancia de $Gap-GM[\mathcal{K}]$, la probabilidad de que $\mathbb{M}$ clasifique correctamente la instancia (G,H) es mayor o igual que $\frac{3}{4}$. Además el tiempo de cómputo de $\mathbb{M}$ está acotado por $f(|H|)p(|G|)$, donde f es una función computable y p es un polinomio.*

Corolario 3. *Si $\mathcal{K}$ es una clase de arboricidad acotada, existe un algoritmo probabilístico $\mathbb{M}$ tal que dados G un grafo y $k,c \in \mathbb{N}$, $\mathbb{M}$ calcula dos listas L_F, L_E constituidas por grafos tamaño k y tales que:*

- $\Pr(L_F$ *contiene todos los subgrafos c-frecuentes de G pertenecientes a $\mathcal{K}$)* $\geq \frac{3}{4}$.
- $\Pr(L_E$ *contiene todos los subgrafos c-excepcionales de G pertenecientes a $\mathcal{K}$)* $\geq \frac{3}{4}$.

Además el tiempo de cómputo de $\mathbb{M}$ está acotado por $f(k)p(|x|)$, donde f es una función computable y p es un polinomio.

3.3.2. *Resolviendo $p-\#EMB(\mathcal{K})$ en clases de arboricidad acotada.* En esta subsección probaremos que, si $\mathcal{K}$ es una clase de arboricidad acotada, entonces los problemas $p-\#EMB(\mathcal{K})$ y minería exacta restringida a $\mathcal{K}$ son computables en tiempo fpt. Para ello necesitaremos usar un famoso teorema de Courcelle, que enunciamos a continuación. Antes debemos introducir algunos conceptos.

En esta sección introduciremos una manera no estándar de mirar los grafos. Un grafo G puede ser visto como una tupla (A_G, V_G, E_G, I_g), donde A_G es un conjunto, el conjunto de vértices y aristas del grafo G, V_G y E_G son predicados unarios correspondientes a el conjunto de vértices y al conjunto de aristas del grafo G y finalmente I_G es una relación binaria, la relación de incidencia, determinada por: $I_G(v,e)$ si y solo si v es un vértice, e es una arista y v es un extremo de e. Adicionalmente debe satisfacerse la siguiente condición: Para todo $e \in E_G$, existen exactamente dos elementos v,w de V_G tales que $I_G(v,e)$ y $I_G(w,e)$.

Dado $G=(A_G,V_G,E_G,I_G)$ un subgrafo de G es un par (V_0,E_0) tal que:

(1) $V_0 \subset V_G$.
(2) $E_0 \subset E_G$.
(3) Para todo $e \in E_0$ si $I(v,e)$, entonces $v \in V_0$.

Sea $\mathcal{K}$ una clase de arboricidad acotada por w, sea $G \in \mathcal{K}$ y sea $\left(\mathcal{T},\ (\overrightarrow{a_t})_{t\in\mathcal{T}}\right)$ una descomposición de G de orden w. Podemos suponer, sin pérdida de generalidad, que para todo $t \in \mathcal{T}$, $\alpha(t)=w$. En lo que sigue definiremos *El árbol de Courcelle* de la descomposición $\left(\mathcal{T},\ (\overrightarrow{a_t})_{t\in\mathcal{T}}\right)$. El árbol de Courcelle de la descomposición $\left(\mathcal{T},\ (\overrightarrow{a_t})_{t\in\mathcal{T}}\right)$ es un árbol etiquetado $\mathbb{T}_C=(\mathcal{T},\lambda,\Sigma)$, en el cual, usando las etiquetas, se codifica tanta información acerca de G como es posible, para de esta manera

poder verificar, usando $(\mathcal{T},\ \lambda, \Sigma)$, afirmaciones acerca de G expresables en la lógica monádica de segundo orden. La razón por la cual esto es útil, es por que existen algoritmos de tiempo lineal, basados en autómatas de árbol, que permiten verificar fórmulas monádicas de segundo orden sobre árboles, i.e. el árbol de Courcelle nos permite reducir el problema de verificar MSO fórmulas sobre H, lo cual puede ser muy difícil, a verificar MSO fórmulas sobre un árbol, lo cual es fácil, ¡es posible en tiempo lineal!

El árbol de Courcelle del par $\left(G, \left(\mathcal{T},\ (\overrightarrow{a_t})_{t\in\mathcal{T}}\right)\right)$ es el árbol etiquetado $\mathbb{T}_C = (\mathcal{T},\ \lambda, \Sigma)$ definido por:

Para todo $t \in \mathcal{T}$, la etiqueta de t, $\lambda(t)$, es la 5-tupla $(\lambda_1(t), ..., \lambda_5(t))$:

(1) $\lambda_1(t) = \{i \leq w+1 : (\overrightarrow{a_t})_i \in V_G\}$.
(2) $\lambda_2(t) = \{i \leq w+1 : (\overrightarrow{a_t})_i \in E_G\}$.
(3) $\lambda_3(t) = \left\{(i,j) \in [w+1]^2 : I\left((\overrightarrow{a_t})_i, (\overrightarrow{a_t})_j\right)\right\}$.
(4) $\lambda_4(t) = \left\{(i,j) \in [w+1]^2 : (\overrightarrow{a_t})_i = (\overrightarrow{a_t})_j\right\}$.
(5) Si t no es la raíz de $\mathcal{T}$ y s es el padre de t, entonces

$$\lambda_5(t) = \left\{(i,j) \in [w+1]^2 : (\overrightarrow{a_t})_i = (\overrightarrow{a_s})_j\right\}$$

Si t es la raíz de $\mathcal{T}$, $\lambda_5(t) = \varnothing$.

Finalmente tenemos que Σ es igual a $2^{[w+1]} \times 2^{[w+1]} \times 2^{[w+1]^2} \times 2^{[w+1]^2} \times 2^{[w+1]^2}$.

Es posible codificar los subconjuntos de G, y en particular los subgrafos de G, como subestructuras de $\mathbb{T}_C$. Dado $S \subset A_G$ y dado $i \leq w+1$, $U_i(S) = \{t \in \mathcal{T} : (\overrightarrow{a_t})_i \in S\}$. Además, $U(S) = (U_1(S), ..., U_{w+1}(S))$. Finalmente si $a \in A_G$, $U(a) = U(\{a\})$.

Lema 4. *Existen MSO fórmulas Subgraph* $(X_1, ..., X_{k+1})$, *Vertex* $(X_1, ..., X_{k+1})$ *y Edge* $(X_1, ..., X_{k+1})$ *tales que para toda* $w+1$*-tupla* $(U_1, ..., U_{k+1})$ *de subconjuntos de* $\mathcal{T}$ *se tiene que*

- $\mathbb{T}_C \models Subgraph(U_1, ..., U_{k+1})$ *si y solo si existe un subgrafo* (V_0, E_0) *de* G *tal que* $(U_1, ..., U_{k+1}) = U(V_0 \cup E_0)$.
- $\mathbb{T}_C \models Vertex(U_1, ..., U_{k+1})$ *si y solo si existe* $v \in V_G$ *tal que* $(U_1, ..., U_{k+1}) = U(v)$.
- $\mathbb{T}_C \models Edge(U_1, ..., U_{k+1})$ *si y solo si existe* $e \in V_G$ *tal que* $(U_1, ..., U_{k+1}) = U(e)$.

Para una prueba el lector puede consultar Flum and Grohe, [7].

Teorema 5 (Teorema de Courcelle)**.** *Dada* $\varphi(X_1, ..., X_p, y_1, ..., y_q)$ *una MSO fórmula en el vocabulario de la teoría de grafos, (i.e. en el vocabulario* (V, E, I)*). Es posible calcular de manera efectiva una MSO fórmula* $\varphi^*\left(\overline{X_1}, ..., \overline{X_p}, \overline{y_1}, ..., \overline{y_q}\right)$, *en el vocabulario de la estructura* $\mathbb{T}_C$, *tal que para todo* $S_1, ..., S_p \subset A_G$ *y para todo* $a_1, ..., a_q \in A_H$ *se tiene que:*

$$G \models \varphi(S_1, ..., S_p, a_1, ..., a_q)$$

si y solo si

$$\mathbb{T}_C \models \varphi^* \left(U\left(S_1\right), ..., U\left(S_p\right), U\left(a_1\right), ..., U\left(a_q\right)\right)$$

Para una prueba del teorema de Courcelle, (realmente de una versión más general del teorema) y para una prueba del teorema a continuación el lector puede consultar ídem, [7].

Teorema 6 (Conteo en árboles)**.** *Dado* $(\mathcal{A}, \nu, \Omega)$ *un árbol etiquetado, y dada* $\varphi\left(X_1, ..., X_p, y_1, ..., y_q\right)$ *una MSO fórmula en el vocabulario de* $(\mathcal{A}, \nu, \Omega)$*, es posible calcular en tiempo polinomial en* $|\mathcal{A}|$ *la cantidad:*

$$\left|\left\{\left(S_1, ..., S_p, a_1, ..., a_q\right) : (\mathcal{A}, \nu, \Omega) \models \varphi\left(S_1, ..., S_p, a_1, ..., a_q\right)\right\}\right|$$

Dado $H = \left(A_H, [k], \left\{e_1, ..., e_m\right\}, I_H\right)$ un grafo con k vértices y m aristas, existe una MSO fórmula $\varphi_H\left(x_1, ..., x_k, y_1, ..., y_m\right)$ tal que para todo grafo G, para todo $v_1, ..., v_k \in V_G$ y para todo $e_1, ..., e_m \in E_G$ se tiene que $G \models \varphi_H\left(v_1, ..., v_k, e_1, ..., e_m\right)$ si y solo si la función $f : [k] \to \left\{v_1, ..., v_k\right\}$ definida por $f\left(i\right) = v_i$, para todo $i \leq k$ es un isomorfismo de H en el subgrafo de G determinado por el par $\left(\left\{v_1, ..., v_k\right\}, \left\{e_1, ..., e_m\right\}\right)$. $\varphi_H\left(x_1, ..., x_k, y_1, ..., y_m\right)$ es la MSO fórmula dada por:

- $\varphi_1\left(x_1, ..., x_k\right) = \left(\bigwedge\limits_{i \leq k} V\left(x_i\right) \wedge \bigwedge\limits_{i \neq j} x_i \neq x_j \wedge \forall z \left(V\left(z\right) \Rightarrow \bigvee\limits_{i \leq k} z = x_i\right)\right)$.
- $\varphi_2\left(y_1, ..., y_m\right) = \left(\bigwedge\limits_{i \leq m} E\left(y_i\right) \wedge \bigwedge\limits_{i \neq j} y_i \neq y_j \wedge \forall z \left(E\left(z\right) \Rightarrow \bigvee\limits_{i \leq m} z = y_i\right)\right)$.
- $\varphi_3\left(x_1, ..., x_k, y_1, ..., y_m\right) = \bigwedge\limits_{I_H(i,j)} I\left(x_i, y_j\right) \wedge \bigwedge\limits_{\sim I_H(i,j)} \sim I\left(x_i, y_j\right)$.
- $\varphi_H\left(x_1, ..., x_k, y_1, ..., y_m\right) = \exists x_1, ..., x_k, y_1, ..., y_m \left(\varphi_1 \wedge \varphi_2 \wedge \varphi_3\right)$.

Corolario 4. $p - \#EMB\left(\mathcal{K}\right)$ *es computable en tiempo* fpt*.*

Proof. Sea $\mathcal{K}$ una clase de arboricidad acotada por w y sea (G, H) una instancia de $p - \#EMB\left(\mathcal{K}\right)$. Considere el siguiente algoritmo

(1) Calcule $\left(\mathcal{T}\ , (\overrightarrow{a_t})_{t \in \mathcal{T}}\right)$, donde $\left(\mathcal{T}\ , (\overrightarrow{a_t})_{t \in \mathcal{T}}\right)$ es una descomposición de G de orden w.
(2) Construya $\mathbb{T}_C = (\mathcal{T}, \lambda, \Sigma)$ el árbol de Courcelle de $\left(G, \left(\mathcal{T}\ , (\overrightarrow{a_t})_{t \in \mathcal{T}}\right)\right)$.
(3) Dada $\varphi_H\left(X\right)$ construya la fórmula $\varphi_H^*\left(X_1, ..., X_{k+1}\right) \wedge Subgraph\left(X_1, ..., X_{k+1}\right)$.
(4) Calcule $r = \left|\left\{\left(S_1, ..., S_{w+1}\right) : \mathbb{T}_C \models \varphi_H^*\left(S_1, ..., S_{k+1}\right) \wedge Subgraph\left(S_1, ..., S_{k+1}\right)\right\}\right|$.
(5) Calcule $\alpha = \left|\left\{f : V\left(H\right) \to V\left(H\right) : f \text{ es un automorfismo de } H\right\}\right|$.
(6) Retorne $r \cdot \alpha$.

Es fácil verificar que $r \cdot \alpha$ es igual a $\#EMB\left(G, H\right)$ y que el algoritmo es un algoritmo de tiempo fpt. □

El siguiente corolario indica que, para el caso en que $\mathcal{K}$ es una clase de arboricidad acotada, el problema de minar grafos en $\mathcal{K}$ es mas fácil de lo que podríamos haber supuesto, (a la luz de los resultados anteriores), específicamente el teorema asegura que es posible resolver en tiempo fpt el problema de minar grafos en $\mathcal{K}$, esto es, el problema de minería exacta restringido a $\mathcal{K}$ pertenece a $\mathcal{FPT}$.

Corolario 5. *El problema de minería exacta restringido a $\mathcal{K}$ es computable en tiempo fpt.*

4. Conclusiones y problemas

En este artículo hemos estudiado restricciones del problema $Gap - GM$ el cual se demostró, en Montoya, [10] que es intratable. En particular, se estudiaron restricciones del problema a clases de grafos de arboricidad acotada y se logró probar que estas restricciones son tratables, i.e. computables en tiempo fpt. Es importante señalar que las clases de grafos que son consideradas en la práctica; como por ejemplo las clases de los árboles, de los bosques, de los órdenes, de los grafos acíclicos; son clases de arboricidad acotada. Un problema que queda por resolver es el de cartografiar la tratabilidad de $Gap - GM$. Problema que creemos puede ser resuelto probando que la conjetura, enunciada en este escrito y en ídem, [10], es correcta.

Agradeciminentos: Dedicado a Mamadimitriou e Hijodimitriou. El autor agradece a la Universidad Industrial de Santander por brindarle las facilidades que hicieron posible la realización de este escrito. Esta investigación fue realizada con el auspicio de la VIE-UIS, proyecto 5153-2007.

Referencias

[2] N. Alon, J. Spencer. The Probabilistic Method. Wiley, 2a. edicion, 2000.
[1] V. Arvind and V. Raman. Approximation algorithms for some parameterized counting problems. In P. Bose and P. Morin, editors, *Proceedings of the 13th Annual International Symposium on Algorithms and Computation,* Volume 2518 of *Lecture Notes in Computer Science,* pages 453-464. Springer-Verlag, 2002.
[3] V. Dalmau and P. Jonsson. The complexity of counting homomorphism seen from the other side. *Theoretical Computer Science,* 329:315-323, 2004.
[4] R. Diestel. Graph Theory. Springer Verlag 2005.
[5] R.G. Downey and M.R. Fellows. Parameterized Complexity. Springer-Verlag, 1999.
[6] J. Flum and M. Grohe. The parameterized complexity of counting problems. *SIAM Journal of Computing,* 33(4):892-922, 2004.
[7] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer-Verlag, 2006.
[9] M. Grohe. The complexity of homomorphism and constraint satisfaction problem seen from the other side, *Journal of the ACM*, 54(1), —,2007.
[10] J. A. Montoya. La complejidad paramétrica de minar grafos I, resultados negativos. *Sometido, Revista Colombiana de Computación.*
[7] R. Motwani and P. Raghavan, Randomized Algorithms, Cambridge University Press, Boston MA, 1996.
[11] C.H. Papadimitriou. Computational Complexity. Addison-Wesley, 1994.
[12] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing,* 20(5):865-877, 1991.

Universidad Industrial de Santander
E-mail address: juamonto@uis.edu.co, amontoyaa@googlemail.com,