Boletín de Matemáticas
Nueva Serie, Volumen XIV No. 2 (2007), pp. 68—82

LA PERMANENTE COMPLEJIDAD DE LA PERMANENTE

J. ANDRÉS MONTOYA (*)

A Mamadimitriou e Hijodimitriou

Resumen. En este artículo estudiamos, a la luz de los famosos teoremas de Valiant y Toda, la complejidad computacional de calcular permanentes de matrices enteras.

Palabras claves. Máquinas de Turing, clases de complejidad, algoritmos eficientes, costos computacionales.

Abstract. In this paper we study, in the light of the famous theorems of Valiant and Toda, the computational complexity of calculating permanents of integer matrices.

Key words and phrases. Turing machines, complexity classes, efficient algorithms, computational costs.

2000 Mathematics Subject Classification: 90C60, 68Q25

1. Introducción

Los problemas de conteo suelen ser difíciles de resolver, es más, existen muy pocos problemas de conteo no triviales que hoy en día podemos resolver de manera eficiente. Es claro que contar es al menos tan difícil como decidir, por ello es pertinente considerar las siguientes preguntas:

¿Contar puede ser estrictamente más difícil que decidir?

¿Qué tanto más difícil puede ser contar que decidir?

En este breve artículo intentaremos presentar y discutir un par de respuestas a las preguntas anteriores. Específicamente presentaremos y analizaremos los teoremas de Valiant y Toda los cuales nos permiten entender qué tan difícil

(*) J. Andrés Montoya. Departamento de Matemáticas Universidad Industrial de Santander. E-mail: juamonto@uis.edu.co.

puede llegar a ser un problema de conteo. Los teoremas de Valiant y Toda también pueden ser usados para entender la complejidad intrínseca de evaluar ciertos funcionales algebraicos como el determinante o la permanente. De manera específica analizaremos la complejidad del problema consistente en calcular la permanente de matrices booleanas.

1.1. Organización del artículo. El artículo está organizado en ocho secciones incluyendo la introducción. En la sección dos se introducen los conceptos básicos de la complejidad computacional. En la sección tres se introducen los fundamentos de la complejidad de problemas de conteo. En esta sección se enuncian y comentan los famosos teoremas de Valiant [7] y Toda [5]. En la sección cuatro se estudia la relación existente entre los problemas de conteo tratables y el cálculo de determinantes de matrices enteras. En la sección cinco se introduce la permanente a partir del determinante, se mencionan sus similitudes y sus diferencias. En la sección seis se enuncia y analiza la hipótesis de Valiant [8] referente a la complejidad de la permanente. En la sección siete se analiza la complejidad de la permanente desde el punto de vista de la *Average case Complexity.* Finalmente en la sección ocho, a manera de conclusión, se hace un recuento de las preguntas más importantes tratadas en el escrito, se menciona y se insiste en que estas preguntas no tienen aún una respuesta satisfactoria.

2. Preliminares

Considere los siguientes problemas

Problema 1 ($CIRCSAT$)**.** .

- *Input: C un circuito booleano*
- *Problema: Decida si C es satisfactible, esto es, decida si existe un modelo para C*

Definición 1. *Dado G un grafo, un matching en G es una colección $\{e_1, ..., e_n\}$ de aristas tal que*

1. *Si $i \neq j$, e_i y e_j no tiene extremos comunes*
2. *Todo vértice de G es el extremo de alguna de las aristas en $\{e_1, ..., e_n\}$*

Problema 2 ($MATCHING$)**.** .

- *Input: G un grafo*
- *Problema: Decida si G tiene un matching*

Los dos problemas anteriores tiene en común el ser solubles en tiempo polinomial no determinista. ¿Qué quiere decir lo anterior? Considere los siguientes algoritmos

Algoritmo 1 $(CIRCSAT)$**.** .

Con input C, donde C es un circuito booleano con k inputs

1. *Adivine $v_1, ..., v_k$, donde $v_1, ..., v_k \in \{0, 1\}$*
2. *Verifique que el vector booleano $(v_1, ..., v_k)$ satisface el circuito C*
3. *Acepte si éste es el caso, en caso contrario rechace*

Algoritmo 2 $(MATCHING)$**.** .

Con input G

1. *Adivine un conjunto de aristas $\{e_1, ..., e_n\}$*
2. *Verifique que $\{e_1, ..., e_n\}$ es un matching.*
3. *Acepte si $\{e_1, ..., e_n\}$ es un matching, en caso contrario rechace*

Los dos algoritmos tienen muchas cosas en común, una de ellas que su tiempo de cómputo es polinomial, otra, quizás la más importante, es la sospechosa primera instrucción de cada uno de ellos. Es obvio que la instrucción *adivine algo* no puede ser una instrucción de un algoritmo en el sentido tradicional del término, es por ello que decimos que los dos algoritmos son no deterministas, porque no se determina la manera en que en el paso 1 la escogencia debe ser realizada y solo se le pide al algoritmo que adivine un objeto adecuado, un objeto solución, si es el caso que tales objetos (soluciones) existen.

El no determinismo incrementa nuestro poder de cómputo, (aunque no de manera ilimitada), los dos problemas anteriores, que pueden ser muy difíciles de resolver sin la ayuda del no determinismo, son fáciles de resolver con su ayuda, esto es así dado que para cada uno de estos problemas decidir si un input x pertenece al problema es equivalente a decidir si existe un objeto y (un modelo, un matching) de tamaño polinomial y tal que $(x, y) \in R$, donde R es una relación fácil de verificar. En los ejemplos anteriores la relación R corresponde a:

1. $(C, (v_1, ..., v_k)) \in R$ si y solo si para todo $(v_1, ..., v_k)$ es un modelo de C.
2. $(G, \{e_1, ..., e_n\}) \in R$ si y solo si $\{e_1, ..., e_n\}$ es un matching en G.

Note que en los dos casos anteriores la relación R es fácil de verificar, es decir, dada (x, y) podemos decidir rápidamente si (x, y) pertenece a R. Dado x y dado y tales que $(x, y) \in R$, diremos que y es un *certificado* para x, porque en cierto sentido y certifica que x pertenece al lenguaje en cuestión. Un problema L puede ser resuelto en tiempo polinomial no determinista si el problema determina una noción adecuada de certificado, una noción tal que dada cualquier

instancia x del problema, es fácil decidir si $x \in L$, conociendo un certificado para x.

Dado $n \in \mathbb{N}$, $\{0,1\}^n$ denotará al conjunto de las palabras de longitud n. Sea L un problema, R una relación y p un polinomio tales que:

1. $x \in L$ si y solo si existe $y \in \{0,1\}^{p(|x|)}$ tal que $(x,y) \in R$.
2. R es fácil de verificar.

De R diremos que es una relación p-balanceada y que el polinomio p es su módulo. El problema L es un problema genérico para la clase de problemas que pueden ser resueltos velozmente con la ayuda del no determinismo. Considere el siguiente algoritmo:

Algoritmo 3. .

Con input x

1. *Adivine* $y \in \{0,1\}^{p(|x|)}$
2. *Verifique que* $(x,y) \in R$
3. *Acepte x si $(x,y) \in R$, en caso contrario rechace x*

El tiempo de cómputo de este algoritmo está dado por $p(|x|)$, el tiempo requerido para adivinar y, mas el tiempo requerido para verificar que $(x,y) \in R$.

Por otro lado, el problema anterior puede ser resuelto de manera trivial sin el concurso del no determinismo, para ello podemos usar un algoritmo de fuerza bruta o de búsqueda exhaustiva. Sea $\mathbb{M}$ el algoritmo dado por:

Algoritmo 4. .

Con input x

1. *Liste los elementos de* $\{0,1\}^{p(|x|)}$
2. *Recorra la lista y para cada y en la lista decida si* $(x,y) \in R$
3. *Si existe y tal que $(x,y) \in R$ acepte x, en caso contrario rechace x*

El algoritmo $\mathbb{M}$ es correcto pero no es un buen algoritmo, el problema es que el espacio de búsqueda es demasiado grande, $\left|\{0,1\}^{p(|x|)}\right| = 2^{p(|x|)}$, por lo que el tiempo de cómputo de $\mathbb{M}$ no puede ser inferior a $2^{p(|x|)}$. Esto es, el tiempo de cómputo de $\mathbb{M}$ no es polinomial.

Todo algoritmo de búsqueda exhaustiva es ineficiente cuando el espacio de búsqueda es grande, lo que se requiere entonces son algoritmos que usen estrategias inteligentes de búsqueda, la estrategia más inteligente posible es el no

determinismo, (corresponde a una inteligencia omnimiscente), desafortunadamente es tan inteligente que no podemos implementarla computacionalmente.

Todo problema de la forma: *Dado x determine si x tiene un certificado*, puede ser resuelto eficientemente con la ayuda del no determinismo, es decir con la ayuda de una inteligencia omnimiscente. En algunas ocasiones el no determinismo no es indispensable. Existe por ejemplo un algoritmo polinomial determinista que resuelve el problema $MATCHING$ [4]. En otras ocasiones el no determinismo parece indispensable, éste parece ser el caso con el problema $CIRCSAT$ el cual es NP-completo, (lo que quiere decir que si para $CIRCSAT$ el no determinismo no es indispensable, entonces el no determinismo nunca es indispensable).

Definición 2. *NP es la clase de todos los problemas de decisión que pueden ser resueltos en tiempo polinomial no determinista*

3. Problemas de Conteo

Sea $L \in NP$ y sea R una relación p-balanceada tal que para todo $x \in \{0,1\}^*$ se tiene que $x \in L$ si y solo si $\exists y \in \{0,1\}^{p(|x|)} \left((x,y) \in R\right)$, donde p es el módulo de R. Dado x, el conjunto de R-soluciones para x es el conjunto S_x^R definido por $S_x^R := \left\{ y \in \{0,1\}^{p(|x|)} : (x,y) \in R \right\}$. Note que $x \in L$ si y solo si $S_x^R \neq \varnothing$. De lo anterior tenemos que el problema de decisión L es exactamente el siguiente problema

- *Input:* $x \in \{0,1\}^*$
- *Problema: Decida si* $S_x^R \neq \varnothing$

A partir de la relación R podemos definir un problema de conteo asociado a L al que denotaremos con el símbolo $\#L$. El problema $\#L$ es el problema de conteo definido por

- *Input:* $x \in \{0,1\}^*$
- *Problema: Calcule* $\left|S_x^R\right|$

Es claro que si podemos resolver el problema $\#L$, podemos entonces usar tal solución para resolver el problema L, esto es así porque si podemos calcular el tamaño de S_x^R, podemos entonces decidir si S_x^R es diferente de vacío. Lo anterior indica que los problemas de conteo son al menos tan difíciles como los correspondientes problemas de decisión. ¿Existe un problema de decisión, llamémoslo L, tal que el problema de conteo $\#L$ es estrictamente más difícil que el problema de decisión L? En las siguientes secciones estudiaremos una

respuesta parcial a esta pregunta, tal respuesta es el así llamado *Teorema de Valiant* [7].

Dado $\#L$ un problema de conteo y dada R una relación p-balanceada como en el párrafo anterior, con el símbolo $\#L(x)$ denotaremos a $\left|S_x^R\right|$.

3.1. Reducciones entre problemas de conteo. En esta sección introduciremos dos nociones de reducción entre problemas de conteo.

Definición 3 (Reducciones parsimoniosas)**.** *Sean $\#L$ y $\#S$ dos problemas de conteo. Diremos que $\#L$ es parsimoniosamente reducible a $\#S$ si y solo si existe un algoritmo $\mathbb{M}$ de tiempo polinomial tal que para toda instancia x de $\#L$, el algoritmo $\mathbb{M}$ calcula una instancia y de $\#S$ que satisface $\#L(x) = \#S(y)$.*

Definición 4. *Dado $L \in NP$, diremos que $\#L$ es $\#P$-completo si y solo si para todo $\#S \in NP$ se tiene que $\#S$ es parsimoniosamente reducible a $\#L$.*

Ejemplo 1. *$\#CIRCSAT$ es $\#P$-completo.*

La noción de reducción parsimoniosa es la adaptación natural al contexto de los problemas de conteo de la noción de reducción de Karp para problemas de decisión. De igual manera podemos adaptar al nuevo contexto la noción de reducción de Turing.

Definición 5. *Sean $\#L$ y $\#S$ dos problemas de conteo, diremos que $\#L$ es Turing-reducible a $\#S$ si y solo si existe un algoritmo $\mathbb{M}$ de tiempo polinomial, con acceso a un oráculo para $\#S$ y tal que con input x, $\mathbb{M}$ calcula $\#L(x)$.*

Definición 6. *Dado $L \in NP$, diremos que $\#L$ es $\#P$-Turing completo si y solo si para todo $\#S \in NP$ se tiene que $\#S$ es Turing-reducible a $\#L$.*

3.2. Los teoremas de Valiant y Toda. Retomemos la pregunta de la sección 3. ¿Existe un problema de decisión, llamémoslo L, tal que el problema de conteo $\#L$ es estrictamente más difícil que el respectivo problema de decisión?

Supongamos que $P \neq NP$. Recuerde que el problema $MATCHING$ pertenece a P y que por lo tanto no puede ser NP-duro. Lo anterior implica que $\#MATCHING$ no puede ser $\#P$-completo ya que si lo fuera entonces $MATCHING$ sería NP-completo y por lo tanto NP-duro. ¿Es posible que $\#MATCHING$ sea $\#P$-Turing completo? Note que, si éste es el caso, de ello obtenemos dos interesantes consecuencias:

1. Un ejemplo de un problema L que no es NP-duro pero tal que $\#L$ es NP-duro, dado que podemos usar un algoritmo que resuelva $\#L$ para resolver en tiempo polinomial todo problema en NP.

2. Un ejemplo de una reducción de Turing (en el contexto de problemas de conteo) que no puede ser simulada o reemplazada por una reducción de Karp (en el contexto de problemas de conteo, es decir, reemplazada por una reducción parsimoniosa).

Note que del ítem 1 obtenemos lo siguiente: Si $\#MATCHING$ es $\#P$-Turing completo, entonces la respuesta, a la pregunta central de esta sección, es afirmativa. El teorema de Valiant [7] es el punto de partida de la *counting complexity* (la teoría de complejidad de problemas de conteo), este teorema es la respuesta a nuestra pregunta y fue una gran sorpresa en el momento de su descubrimiento.

Teorema 1 (Teorema de Valiant)**.** *$\#MATCHING$ es $\#P$-Turing completo.*

Cerraremos esta sección enunciando y comentando un segundo teorema de gran importancia y profundidad, este es el Teorema de Toda. El Teorema de Toda [5] es una respuesta a la siguiente cuestión: Si existen problemas de conteo que son más difíciles que los problemas de decisión correspondientes. ¿Qué tanto más difícil puede ser un tal problema de conteo? ¿Qué tanto más difícil es contar que decidir?

Teorema 2 (Teorema de Toda)**.** *$\#MATCHING$ es más difícil que todo problema en la jerarquía polinomial*

El Teorema de Toda en cierto sentido afirma lo siguiente: Un problema de conteo puede ser ω-veces más difícil que un problema de decisión. La jerarquía polinomial es una jerarquía con ω niveles cuyo primer nivel es P, por lo tanto podemos decir, (si la jerarquía polinomial no colapsa), que $\#MATCHING$ es ω veces más difícil que $MATCHING$, dado que $MATCHING \in P$ y $\#MATCHING$ es más difícil que todo problema en la jerarquía polinomial.

4. Determinantes y conteo

En la sección anterior vimos que los problemas de conteo pueden ser muy difíciles. Son muy pocos los problemas de conteo que sabemos resolver eficientemente. A continuación veremos dos ejemplos de problemas de conteo con una solución eficiente. Estos dos ejemplos son de por sí paradigmáticos.

Uno de los pocos problemas de conteo que sabemos resolver de manera eficiente, es el de contar los subárboles generados de un grafo. Existe un algoritmo de tiempo polinomial que resuelve el problema, el algoritmo se remonta a Kirchhoff el famoso físico alemán ochocentista y es una consecuencia directa del, así llamado, *Teorema de la matriz de Kirchhoff.*

Definición 7. *Un árbol $T = (V, E)$ es un grafo conexo tal que $|E| = |V| - 1$.*

Definición 8. *Dado $G = (V, E)$ un grafo, un ciclo de longitud n en G, es una trayectoria $a_0, \ldots, a_n$ tal que a_0 es igual a a_n.*

Proposición 1. *Dado $G = (V, E)$ un grafo conexo, G es un árbol si y solo si G no contiene ciclos.*

Dado $G = (V, E)$ un grafo, un *subárbol generado* de G, es un subgrafo de G, digamos $H = (V, R)$, tal que

1. H es conexo
2. $|R| = |V| - 1$, (es decir, H es un árbol).

Problema 3 (Conteo de subárboles-generados)**.** .

- *Input: G un grafo*
- *Problema: Calcule N. Donde N es el número de los subgrafos generados de G*

Sea $G = ([n], E)$ un grafo con n vértices y sea $M(G)$ la matriz de adyacencia de G. Note que, para todo grafo G, las entradas diagonales de $M(G)$ son iguales a 0.

El *Laplaciano de G*, que denotaremos $L(G)$, es la matriz simétrica $[d_{ij}]_{i,j \leq n}$ definida por:

1. Dados $i \neq j$, se tiene que d_{ij} es igual a $-a_{ij}$
2. Para todo $i \leq n$, $d_{ii} = \deg(i)$, donde $\deg(i)$ es el grado en G del vértice i

Dada A una matriz de $n \times n$ y dado $i \leq n$, A_{ii} denota el i-ésimo *cofactor*, es decir, la matriz de $n - 1 \times n - 1$ obtenida de A eliminando la i-ésima fila y la i-ésima columna. Dado G, al número de subárboles-generados de G lo denotaremos con el símbolo $SAG(G)$.

Teorema 3 (Kirchhoff)**.** *Si G es un grafo con n vértices, entonces para todo $i \leq n$, se tiene que $SAG(G) = \det((L(G))_{ii})$.*

Es claro que el teorema de Kirchhoff determina un algoritmo para resolver el problema de contar subárboles-generados y lo mejor del cuento es que este algoritmo es eficiente ¿Por qué? El algoritmo es eficiente porque calcular determinantes es fácil, podemos calcular en tiempo $O(n^3)$, (usando eliminación gaussiana), para una matriz de tamaño $n \times n$.

El segundo problema que consideraremos es el de contar emparejamientos, (*matchings),* en grafos planos. Recuerde que dado $G = ([2n], E)$, un matching

en G es un subconjunto S de E tal que, para todo $v \in [2n]$, existe un único $e \in S$ para el cual $v \in e$. Dado G un grafo, $\#MATCHING(G)$ es el número de emparejamientos en G

Problema 4 ($\#MATCHING(planar)$)**.** .

- *Input:* $G = ([2n], E)$*, donde* G *es un grafo plano*
- *Problema: Calcule* $\#MATCHING(G)$

Dado $E_0 \subset E$ y dado $e \in E_0$, diremos que e es una *arista aislada* en E_0 si y solo si para todo $\varepsilon \in E_0$, se tiene que si $e \neq \varepsilon$, entonces $e \cap \varepsilon = \varnothing$. Dado Z un ciclo en G, diremos que Z es un *ciclo par* si y solo si la longitud de Z es un número par.

Proposición 2. *Dados* M_1, M_2 *dos emparejamientos de* G*,* $M_1 \cup M_2$ *es una unión disyunta de ciclos pares y de aristas aisladas en* $M_1 \cup M_2$*.*

Una *orientación de* G es un grafo dirigido $\overrightarrow{G} = ([2n], D)$ tal que, para toda arista $\{v, w\}$, se tiene que o $(v, w) \in D$ o $(w, v) \in D$, pero las parejas (v, w) y (w, v) no pertenecen a D de manera simultánea.

Una *traversa* de G es una trayectoria $a_0, \ldots, a_N$ que pasa por cada uno de los vértices de G. Dada $\overrightarrow{G}$ una orientación de G y dado C un ciclo en G, diremos que C es *imparmente orientado segun* $\overrightarrow{G}$ si y solo si toda traversa de G cruza un número impar de las aristas de C en sentido contrario a la orientación determinada por $\overrightarrow{G}$.

Definición 9. *Una orientación* $\overrightarrow{G}$ *de* G *es una orientación pfaffiana si y solo si, para todo par de emparejamientos* M_1, M_2*, todo ciclo en* $M_1 \cup M_2$ *es imparmente orientado segun* $\overrightarrow{G}$*.*

Dada $\overrightarrow{G} = ([2n], D)$ una orientación de G, la *matriz de adyacencia de la orientación*, que denotaremos $M\left(\overrightarrow{G}\right)$ es la matriz $[a_{ij}]_{i,j \leq 2n}$ definida por:

- Si $(i, j) \in D$, entonces $a_{ij} = 1$
- Si $(j, i) \in D$, entonces $a_{ij} = -1$
- Si $(i, j), (j, i) \notin D$, entonces $a_{ij} = 0$

Lema 1. *Dado un grafo plano* G*, podemos construir en tiempo polinomial en el tamaño de* G *una orientación pfaffiana* $\overrightarrow{G}$*.*

Teorema 4 (Kasteleyn)**.** *Dado* G *un grafo y dada* $\overrightarrow{G}$ *una orientación pfaffiana de* G*, se tiene que*

$$\#MATCHING(G) = \sqrt[2]{\det\left(M\left(\overrightarrow{G}\right)\right)}$$

Del teorema anterior podemos derivar un algoritmo de tiempo polinomial que resuelve el problema $\#MATCHING\,(planar)$. Note que nuevamente hemos resuelto el problema propuesto reduciéndolo al cálculo de un determinante.

4.1. Una pregunta de Jerrum. Valiant [8] observa que todos los problemas de conteo que hoy en día podemos resolver eficientemente, se han resuelto, como en los ejemplos anteriores, reduciendo el problema al cálculo de determinantes de matrices enteras. ¿Existe alguna buena razón para ello? (pregunta de Jerrum).

Sea $\mathcal{F} = (p_i)_{i\in\mathbb{N}}$ una familia de polinomios tal que para todo $i \in \mathbb{N}$, el polinomio p_i es un polinomio en las variables $\{X_1, ..., X_i\}$. Diremos que una tal familia es fácil si y solo si para todo $i \in \mathbb{N}$, el polinomio p_i puede ser evaluado usando un circuito algebraico de tamaño $g\,(i)$, donde g es un polinomio que no depende de i. Valiant [8], muestra que toda familia fácil es una *proyección,* (proyección en el sentido de Valiant [8]) de la familia $\mathcal{F}_{\det} := (q_i)$ definida por

1. Si i no es un cuadrado, q_i es el polinomio constante 1
2. Si $i = m^2$ y si identificamos el conjunto de variables $\{X_1, ..., X_i\}$ con el conjunto $\{X_{lh} : l, h \leq m\}$, entonces q_i es igual a
$$\sum_{\pi\in S_m} sign\,(\pi) \cdot \prod_{l\leq m} X_{l\pi(l)}$$

Note que la definición del polinomio q_{m^2} coincide con la definición del determinante de la matriz $[X_{lh}]_{l,h\leq m}$. Lo anterior le permite concluir a Valiant que el cálculo de determinantes es un problema universal para la clase de los *problemas de evaluación* [1] que se pueden resolver de manera eficiente, adicionalmente como todo problema de conteo es un problema de evaluación, podemos entonces concluir que el problema de calcular determinantes de matrices enteras es universal (en cierto sentido) para la clase de los problemas de conteo que admiten una solución eficiente. Los resultados de Valiant, aunque profundos, son tan solo una respuesta parcial a la pregunta antes formulada. Una segunda respuesta parcial a la pregunta de Jerrum puede ser encontrada en [6], Toda et al. muestran que el problema de calcular determinates de matrices enteras es completo para la clase $Gap\,(L)$ bajo reducciones log *space* [4].

5. De permanentes y determinantes

Sea $G = ([2n]\,, E)$ un grafo, diremos que G es un grafo *bipartito* si y solo si para todo $i \lneqq j \leq 2n$, si existe una arista entre los vértices i y j, entonces $i \in [n]$ y $j \in \{n + 1, \ldots, 2n\}$.

Ejemplo 2. *Dado $n \in \mathbb{N}$, el grafo $K_{nn} = ([2n], E_n)$ es un grafo bipartito, donde E_n es el conjunto $\{(i, n+j) : i, j \in [n]\}$. Dado $n \in \mathbb{N}$, K_{nn} es llamado el grafo bipartito completo con $2n$ vértices.*

Considere el siguiente problema de conteo:

Problema 5 ($\#MATCHING(bipartito)$)**.** .

- *Input:* $G := ([2n], E)$. *Donde G es un grafo bipartito*
- *Problema: Calcule $\#MATCHING(G)$. Donde $\#MATCHING(G)$ es el número de emparejamientos de G*

Es importante recordar que, aunque podemos resolver eficientemente el problema $MATCHING$, consistente en decidir, dado $G = ([2n], E)$, si G tiene al menos un emparejamiento, ([4], capítulo 1), el problema de conteo asociado a $MATCHING$, es decir $\#MATCHING$ no puede ser resuelto en tiempo polinomial, (a menos que cosas extraordinarias sucedan), dado que Valiant [7] y Toda [5] probaron que $\#MATCHING$ es más difícil que todo problema en la **jerarquía polinomial**.

Note que para todo grafo bipartito $G = ([2n], E)$, se tiene que

$$\#MATCHING(G) = \sum_{\pi \in S_n} \left(\prod_{i \leq n} a_{i\pi(i)} \right).$$

Dada $M = [a_{ij}]$ una matriz booleana de $n \times n$, la expresión $\sum_{\pi \in S_n} \left(\prod_{i \leq n} a_{i\pi(i)} \right)$ es llamada **la permanente** de M. A la permanente de la matriz M la denotaremos con el símbolo $perm(M)$. Recuerde que

$$\det(M) := \sum_{\pi \in S_n} sign(\pi) \left(\prod_{i \leq n} a_{i\pi(i)} \right).$$

Las definiciones de permanente y de determinante de una matriz booleana son casi idénticas, esto puede llevarnos a creer que, si podemos calcular eficientemente determinantes de matrices booleanas, podemos también calcular eficientemente sus permanentes. Recuerde que el método de eliminación gaussiana nos permite calcular determinantes en tiempo polinomial. ¿Podemos calcular permanentes en tiempo polinomial?

Si pudieramos calcular permanentes en tiempo polinomial, podríamos entonces resolver en tiempo polinomial el problema $\#MATCHING(bipartito)$ lo cual a su vez nos permitiría resolver eficientemente el problema $\#MATCHING$, pero esto es imposible a menos que la jerarquía polinomial colapse (¡altamente improbable!).

Conjetura 1 (Hipótesis de Valiant)**.** *Es imposible calcular permanentes de manera eficiente* [7].

6. ¿Podemos probar la hipótesis de Valiant?

En esta sección consideraremos la siguiente pregunta: ¿Podemos probar la hipótesis de Valiant? O mejor ¿Qué tan cerca estamos de poder probar la hipótesis de Valiant? Note que refutar la hipótesis de Valiant implica refutar $P \neq NP$. Existe cierto consenso dentro de la *complexity-theory community* respecto a que toda afirmación que implique $P \neq NP$ o $P = NP$ no podrá ser probada en el corto plazo.

¿Qué tan interesante es la hipótesis de Valiant? O mejor ¿Qué tanto interés existe dentro de la comunidad por encontrar una prueba de la hipótesis de Valiant? Al respecto nos permitiremos reseñar brevemente la charla plenaria presentada por Manindra Agrawal en el Congreso Internacional de Matemáticas Madrid 2006. La charla de Agrawal titulada *Determinant vs Permanent* [9] tuvo por tema los últimos avances en la búsqueda de una prueba para la hipótesis de Valiant. Los desarrollos recientes, estudiados por Agrawal en su charla, son dos, a saber, el trabajo de Mulmuley-Sohoni [3] y el trabajo de Impagliazzo-Kabanets [2].

La aproximación de Kabanets-Impagliazzo es llamada por Agrawal la *Derandomization approach.* Considere el siguiente problema computacional

Problema 6 (Identity Testing Problem)**.** .

- *Input:* $(\mathbb{F}, p, q)$*, donde* $\mathbb{F}$ *es un campo finito,* $p, q \in \mathbb{F}[X_1, ..., X_n]$
- *Problema: Decida si en* $\mathbb{F}$ *los polinomios* p *y* q *son iguales*

El *Identity testing problem* es un problema que puede ser resuelto en tiempo polinomial probabilístico [1] pero para el cual no se conoce un algoritmo determinístico de tiempo polinomial. Kabanets-Impagliazzo prueban el siguiente teorema

Teorema 5. *Si existe un algoritmo determínistico de tiempo subexponencial para el Identity testing problem entonces* $NEXPTIME \nsubseteq P/poly$ *o*

la hipótesis de Valiant es verdadera.

Es claro que el teorema anterior, y en general el trabajo de Kabanets-Impagliazzo, abre un camino prometedor que puede llevarnos a probar la hipótesis de Valiant, aún así es necesario anotar que el teorema anterior nos deja aún muy lejos de encontrar una prueba para dicha hipótesis.

Por otro lado Mulmuley-Sohoni han encontrado una manera completamente novedosa de acercarse a la conjetura de Valiant, ellos han mostrado cómo algunas herramientas de la geometría algebraica, específicamente de la teoría de invariantes geométricos, pueden arrojar luz sobre este problema. Es importante anotar que el camino que queda por recorrer, siguiendo la aproximación de Mulmuley-Sohoni, es aún un camino muy largo e intrincado.

7. La permante es dura casi siempre, (¿permanentemente?)

La corta sección a continuación es una pequeña incursión en el campo de la *average case complexity.* Como vimos en las secciones anteriores, el cálculo de permanentes es un problema difícil, (muy difícil si la jerarquía polinomial no colapsa). Nuestra noción de dificultad es obviamanete relativa o por decirlo de otra manera convencional, esto es así porque para desarrollar nuestros análisis y probar nuestros resultados hemos asumido por el camino una serie de convenciones, una de ellas por ejemplo es la de analizar la complejidad de un problema desde el punto de vista del peor caso posible, ¿Qué puede pasar si asumimos otra convención, como por ejemplo analizar la complejidad de un problema desde el punto de vista del caso promedio? ¿Qué puede pasar con el problema de calcular permanentes? La dureza del problema de calcular permanentes de matrices booleanas indica que existe una infinidad de instancias de este problema que son excesivamente difíciles como para ser resueltas en tiempo polinomial. Pero ¿Cuál es la medida de esta infinidad?

Note que $perm\left(\left[X_{ij}\right]_{i,j\leq n}\right)$ es un polinomio $p\left(X_{ij} : i,j \leq n\right)$ de grado n en las variables $\{X_{ij} : i,j \leq n\}$. Sean $A = [a_{ij}]$, B una matriz de $n \times n$ y $p_B(t)$ el polinomio en una variable definido por $p_B(t) := p(A + tB)$.

Si podemos calcular velozmente $p_B(t)$ en los $n+1$ puntos $\{A+B, A+2B, ..., A+(n+1)B\}$ podemos entonces, usando interpolación, calcular el valor de p_B en 0, o lo que es lo mismo podemos calcular el valor de p

en A. Que podamos calcular p_B en los puntos $\{A+B, A+2B, ..., A+(n+1)B\}$ dependerá de la elección de B. Si elegimos B al azar la probabilidad de elegir un buen B, es decir de elegir un B tal que $A+B, A+2B, ..., A+(n+1)B$ sean instancias fáciles de p dependerá de qué tantas instancias fáciles tiene p. En particular si p tiene muchas instancias fáciles lo anterior nos permite diseñar un algoritmo probabilístico para el cálculo de la permanente.

Teorema 6. *Si para todo n, existe $\mathbb{F}$, con $|\mathbb{F}| \geq n$, y tal que es posible calcular velozmente la $\mathbb{F}$-permanente de una fracción igual a $\left(1-\frac{1}{2n}\right)$ de las matrices de $n \times n$ con entradas en $\mathbb{F}$, es entonces posible calcular en tiempo polinomial probabilista la permanente de toda matriz con entradas en $\mathbb{F}$.*

Corolario 1. *Si la jerarquía polinomial no colapsa al segundo nivel, al menos una fracción igual a $\left(\frac{1}{2n}\right)$ de las matrices de $n \times n$, son instancias difíciles de la permanente.*

Demostración. En caso contrario todo problema en la jerarquía polinomial podría ser resuelto en tiempo polinomial probabilísitico, pero esto implicaría que la jerarquía polinomial colapsa al segundo nivel. □

El corolario anterior indica que (si la jerarquía polinomial no colapsa), si elegimos al azar una matriz entera M, la probabilidad de que calcular la permanente de M sea difícil es no despreciable. Podemos entonces concluir que desde el punto de vista del caso medio el problema de calcular permanentes de matrices enteras tampoco parece ser tratable.

8. Conclusiones

Más que presentando una lista de conclusiones quisiéramos cerrar este escrito indicando las preguntas más importantes que se consideraron en él. Tales preguntas fueron discutidas y analizadas sin que, para alguna de ellas, se haya llegado a obtener respuestas conclusivas.

- ¿Por qué es tan difícil contar?
- ¿Por qué todos los problemas de conteo que hoy en día podemos resolver eficientemente, los hemos resuelto reduciéndolos al cálculo de determinantes de matrices enteras?
- Calcular determinantes es fácil. ¿Cuál es la complejidad de calcular permanentes?

Referencias

[1] D Bovet, P Crescenzi. *Introduction to theory of Complexity.* Prentice Hall, 1994
[2] V. Kabanets, R Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. Proceedings ACM Symposium on Theory of Computing. 220-229, 1997
[3] K Mulmuley, M Sohoni. Geometric complexity theory, P vs NP and explicit obstructions. SIAM Journal on Computing, 31(2), 496-526, 2002
[4] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994
[5] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing,* 20(5):865-877, 1991
[6] T. Thierauf, S. Toda, O. Watanabe. On clousure properties of $GapP$. 4, 242-261, (1994)
[7] L. Valiant. The complexity of computing the permanent. *Theoretical computer Science,* 8:189-201, 1979
[8] L Valiant. Completeness classes in algebra. Proceedings ACM symposium on Theory of Computing, 249-261, 1979
[9] M. Agrawal. *Determinant vs Permanent,* conferencia plenaria IMU-2006

Recibido: Mayo de 2007. Aceptado para publicación: Noviembre de 2007