SISTEMAS DE PRUEBAS INTERACTIVAS

EDWARD OLMEDO PACHECO CASTILLO

UNIVERSIDAD DEL CAUCA
FACULTAD DE CIENCIAS NATURALES, EXACTAS Y DE LA EDUCACIÓN
DEPARTAMENTO DE MATEMÁTICAS
POPAYÁN
2004

SISTEMAS DE PRUEBAS INTERACTIVAS

EDWARD OLMEDO PACHECO CASTILLO

TRABAJO DE GRADO

En la modalidad de seminario de grado, presentado como requisito parcial para optar al título de matemático

Director

Esp. DIEGO RAMIRO CORREA CUENE

UNIVERSIDAD DEL CAUCA

FACULTAD DE CIENCIAS NATURALES, EXACTAS Y DE LA EDUCACIÓN

DEPARTAMENTO DE MATEMÁTICAS

POPAYÁN

2004

Nota de aceptación

_____

_____

_____

Director

_____

Especialista Diego Ramíro Correa Cuene

Comité evaludor

_____

Especialista Hebert Vivas

_____

Especialista Martha Romero

Fecha de sustentación: Popayán, 20 de agosto de 2004

A mi familia por todo el amor,
la comprensión y el apoyo que
siempre me han brindado

AGRADECIMIENTOS

El autor expresa su sincero agradecimiento a:

Mauricio Maca Chagüendo, profesor de matemáticas, jefe del departamento Matemáticas por su valiosa colabaración y orientación.
Diego Correa Cuene, profesor de Matemáticas y director de este seminario por su valiosa colaboración y oportunas sugerencias.
Hebert Vivas, profesor de Matemáticas y miembro del comité evaluador por su interés y oportunas sugerencias.
Martha Romero, profesora de Matemáticas y miembro del comité evaluador por sus oportunas sugerencias.
Carlos Trujillo, profesor de Matemáticas, por su colaboración en la realización de este trabajo.
Yudy Marcela Bolaños, profesora de Matemáticas, por su valiosa ayuda en la realización de este trabajo.
Maritza Herrera Florez, profesora de Matemáticas, por su valiosa ayuda en la realización de este trabajo.
A mi familia por su constante apoyo y motivación durante el transcurso de mis estudios.
A Ermila Zúñiga por su motivación y valiosa colaboración.
A mis compañeros de carrera, por su amistad y constante apoyo.

A la Universidad del Cauca, a todos los profesores del departamento de Matemáticas y demás personas que de alguna manera colaboraron con la realización del seminario de grado.

CONTENIDO

LISTA DE FIGURAS

RESUMEN

En este documento se presenta el informe del seminario de grado "Sistemas de Pruebas Interactivas,"realizado dentro del grupo de estudio y desarrollo investigativo en matemática aplicada en la línea de matemática computacional. En la primera parte se presentan algunos conceptos y ejemplos previos. Posteriormente, se realiza una descripción detallada, mediante ejemplos prácticos, de los sistemas de pruebas interactivas y también se muestran los resultados más importantes relacionados con las clases de complejidad y por último, se hace una introducción al teorema PCP, el cual es uno de los principales resultados encontrados en los últimos años, debido a que caracteriza de manera diferente a una de las principales clases de complejidad denominada NP.

INTRODUCCIÓN

A partir del modelo de computación teórico creado por Alan Turing y conocido como máquina de Turing, se ha desarrollado la teoría de la complejidad computacional. Mas adelante la computación eficiente fue definida en términos del tiempo polinomial con respecto a la longitud de la entrada y con ello, se definieron formalmente las principales clases de complejidad denominadas P y NP. Desde entonces y hasta el momento, sigue abierto el gran interrogante, ¿$P = NP$?

Con el fin de dar respuesta a este problema, catalogado como uno de los principales problemas matemáticos del milenio, se han realizado distintos trabajos y creado nuevas teorías, dentro de las cuales están los sistemas de pruebas interactivas originados a partir de las máquinas de Turing probabilísticas y desarrollados por Goldwasser, Mical, Rackoff y Babai.

Mediante las pruebas interactivas, es posible resolver problemas de teoría de grafos, teoría de números y aproximabilidad entre otros. Es por ello, que los sistemas de pruebas interactivas originan una nueva clase de complejidad denotada por IP.

Estos sistemas consisten de dos máquinas de Turing llamadas el *verificador* y el *probador*, las cuales realizan la computación de una misma entrada por medio del intercambio de mensajes. El probador tiene mayor capacidad,[1] por lo tanto, realiza la parte más difícil de la prueba y el verificador, determina si esto es correcto; de manera que, este último es quien acepta o rechaza la entrada. Como este sistema es probabilístico, es posible dis-

[1]Formalmente el probador es una máquina de Turing sin límite computacional, esto es, sus computaciones son ilimitadas tanto en tiempo como en espacio en memoria y se realizan eficientemente (tiempo polinomial).

minuir su error realizando varias computaciones sobre una misma entrada.

A pesar de que la teoría de los sistemas de pruebas interactivas es muy reciente, su capacidad ha permitido desarrollar varios de los más sorprendentes e importantes resultados conocidos hasta el momento en complejidad computacional y aproximabilidad, de tal manera que se ha llegado a extender la clase NP a partir de estos conceptos y se ha mostrado la equivalencia entre las clases IP y $PSPACE$.

Un concepto muy utilizado en las pruebas interactivas es la máquina de Turing con *oráculo*, debido a que sí el probador es una máquina de este tipo, se obtiene un sistema denominado *Comprobación Probabilística de Pruebas*, con el cual es posible caracterizar la clase NP de manera diferente, a este resultado se le conoce como el *Teorema* PCP. También, es posible mostrar la no aproximabilidad de algunos problemas teniendo como principal condición que $P \neq NP$.

Con este trabajo se pretende brindar un texto de consulta para aquellos que se interesen por el campo de la complejidad computacional, más específicamente, es una herramienta apropiada para quienes estén interesados en estudiar e investigar acerca de las pruebas interactivas. Además, presenta una introducción adecuada para la comprensión de uno de los principales resultados encontrados en los últimos años, el teorema PCP. Por otro lado, también se espera que este documento fortalezca el grupo de estudio y desarrollo investigativo en matemática aplicada, en la línea de matemática computacional, debido a que el grupo se encuentra trabajando en algoritmos de aproximación.

1. PRELIMINARES

Para lograr una mayor comprensión de los aspectos relacionados con los sistemas de pruebas interactivas, es necesario hacer una revisión previa. Para ello se darán las definiciones, teoremas y ejemplos que se estimen pertinentes.

1.1. ALFABETOS Y LENGUAJES

Definición 1.1. *Un* ***alfabeto*** *es un conjunto no vacío y finito* $\Sigma = \{\sigma_1, ..., \sigma_k\}$. *Un símbolo es un elemento del alfabeto. Una cadena es una tupla finita* $x = \langle \sigma_{i_1}, \sigma_{i_2}, ..., \sigma_{i_n} \rangle$ *de símbolos de* Σ. *La cadena vacía se denota por* e.

Definición 1.2. *El conjunto infinito de todas las cadenas sobre el alfabeto* Σ *se denotará por* Σ^*. *La longitud de una cadena* $x = \sigma_{i_1}, \sigma_{i_2}, ..., \sigma_{i_n}$ *es el número* n *de símbolos que contiene* x *y se nota* $|x|$. *De ahí, que la cadena vacía tiene longitud 0 y el número de cadenas de longitud* n *sobre un alfabeto de* k*-símbolos es igual a* k^n.

Definición 1.3. *Dado un alfabeto* Σ, *un* ***lenguaje*** L *sobre* Σ *es un subconjunto de* Σ^*. *El complemento de un lenguaje* L, *que se denota* L^c, *se define como* $L^c = \Sigma^* - L$.

Definición 1.4. *Una clase de complejidad* C, *es una colección de todos los lenguajes sobre el alfabeto* Σ *que satisfacen un predicado* π. *Es decir,*

$$C = \{L : L \subseteq \Sigma^* \wedge \pi(L)\}$$

El concepto de clase de complejidad se ilustrará con los siguientes ejemplos:

Ejemplo 1.

La clase τ de lenguajes, se denomina *tally* y se define como:

$$\tau = \{L : L \subseteq \Sigma^* \wedge \pi(L)\}$$

donde

$$\Sigma = \{0, 1\} \ \wedge \ \pi(L) = verdadero \longleftrightarrow L \subseteq 0^{*2}$$

es decir

$$\tau = \{\{0, 00, ...\}, \{00, 000, ..., \}, \{0, 000, ...\}, ...\}.$$

Luego τ es la clase conformada por todos los lenguajes del alfabeto Σ^*, cuyas cadenas solamente contienen el elemento cero.

Ejemplo 2.

La clase *Sparse* (Dispersa) denotada por S, está formada por aquellos lenguajes que son acotados y se define como:

$$S = \{L : L \subseteq \Sigma^* \wedge \pi(L)\}$$

donde

$$\Sigma = \{0, 1\} \, y \, \pi(L) = verdadero \longleftrightarrow \exists p \, \forall n \, (|L_n| \leq p(n))$$

donde p es un polinomio.

Por lo tanto, la clase S es aquella donde la longitud de sus lenguajes está acotada por un polinomio.

[2] 0^* denota cadenas de cualquier longitud conformadas únicamente con ceros.

1.2. MÁQUINAS DE TURING Y CLASES DE COMPLEJIDAD

Es de anotar que la teoría de la complejidad computacional está formalizada a partir del modelo de computación básico, denominado *Máquina de Turing*, que fue desarrollado por Alan Turing en 1936, de acuerdo a como él percibía el pensamiento matemático. Además, los sistemas de pruebas interactivos están definidos a partir de las máquinas de Turing no determinísticas, concepto que se define a continuación.

Definición 1.5. *Una máquina de Turing no determinística (NT) es una terna* (Q, S, δ) *donde:*

1. *Q es un conjunto finito, llamado el conjunto de estados, el cual contiene un estado especial q_0, llamado el estado inicial y tres estados de parada: el estado de aceptación q_Y, el estado de rechazo q_N, y el estado stop.*

2. *S se define como el conjunto finito de símbolos llamado el alfabeto de la máquina, el cual siempre contiene el símbolo blanco b.*

3. *δ es una relación definida de un subconjunto de $(Q - \{q_y, q_N, stop\}) \times S$ en $Q \times S \times D$, donde $D = \{Iz, Dr, Nn\}$ es el conjunto de direcciones (Iz: izquierda, Dr: derecha, Nn: ninguno).*

Si en la definición anterior δ es una función, entonces se tiene una máquina de **Turing determinística** y δ es llamada *función de transición*, lo que refleja la principal diferencia entre una máquina determinística y una no determinística. Esto significa que para una situación particular de una NT, no existe solo una instrucción a seguir sino que la máquina NT puede escoger no determinísticamente entre varias posibilidades la instrucción a ejecutar; de ahí que la computación de una entrada x en una máquina de Turing no determinística $NT(x)$, se puede ver como un árbol, llamado *árbol de computación*, donde los nodos corresponden a los estados de la máquina y las aristas representan las elecciones de la relación δ.

Las dos clases de complejidad más conocidas P y NP tiene asociada una máquina de Turing; la clase P por ejemplo, se define como la clase de todos los lenguajes que son aceptados por una máquina de Turing determinística, esto es:

$P = \{L/L = L(M)$ para alguna máquina de Turing M que corre en tiempo polinomial$\}$

donde

$$L(M) = \{w \in \Sigma^* : M \, acepta \, w\}$$

y M es una máquina de Turing determinística [Coo00].

La clase NP está conformada por todos los lenguajes que son aceptados por una máquina de Turing no determinística. El siguiente resultado es de gran utilidad para caracterizar esta clase de complejidad.

TEOREMA 1.1. *[PC94] Un lenguaje L pertenece a NP si, y sólo si, existen un lenguaje $L_{chequeo} \in P$ y un polinomio p tales que*

$$L = \{x : \exists y(\langle x, y\rangle \in L_{chequeo} \wedge |y| \leq p(|x|))\}$$

Ejemplo 3.

A fin de dar mayor claridad al teorema anterior, se presenta un problema que pertenece a la clase NP, de gran importancia[3] en la teoría de la complejidad y conocido como el problema de *Satisfabilidad*, el cual se ilustrará en seguida.

Dado un conjunto $U = \{u_1, u_2, ..., u_n\}$ de n variables booleanas, una fórmula booleana f se dice que está en *forma normal conjuntiva* (FCN), si se puede expresar como conjunción de m cláusulas C_i, esto es:

$$f = C_1 \wedge C_2 \wedge ... \wedge C_m$$

donde cada cláusula C_i, es a su vez una disyunción de literales, esto es:

$$C_i = (l_{i1} \vee l_{i2} \vee ... \vee l_{ik})$$

[3]Su importancia se debe a la relación directa que tiene con uno de los teoremas más importantes en la teoría de la complejidad, conocido como el Teorema de Cook [GJ79].

cada literal l_{ij} denota una variable booleana u_i o la negación de la variable booleana ($\sim u_i$) de U.

Una asignación de verdad para U, es una función $t : U \rightarrow \{verdadero,\ \ falso\}$ que asigna a cada variable booleana un valor de verdad; de acuerdo a esto se tiene que:

- Un literal l es verdadero, si $l = u_k$ y $t(u_k) = verdadero$ o $l =\sim u_k$ y $t(u_k) = falso$.
- Una cláusula C se satisface para alguna asignación de verdad, si por lo menos un literal incluido en ella es verdadero.
- La fórmula f es satisfactible si existe alguna asignación de verdad tal que las m cláusulas sean satisfechas.

Cuando, en el problema anterior, cada cláusula tiene a lo mas 3 variables se denota por **3-SAT** y se ha demostrado además que es un problema NP-completo [GJ79].

Solucionar el problema, es determinar si existe una asignación de verdad T, tal que la fórmula booleana f sea satisfactible. Para este ejemplo, el certificado[4] que está acotado polinomialmente es precisamente la asignación T.

Teniendo en cuenta las definiciones 1.3, 1.4 y el teorema anterior se puede establecer el siguiente criterio a fin de clasificar el *complemento de* NP, denotado co-NP.

TEOREMA 1.2. *A co-NP pertenecen todos aquellos lenguajes para los cuales no es posible encontrar un certificado acotado polinomialmente.*

Demostración. Sea $L \in NP$, entonces existen $L_{chequeo} \in P$ y un polinomio p tales que

$$L = \{x : \exists y(\langle x, y\rangle \in L_{chequeo} \wedge |y| \leq p(|x|))\}$$

Como $L \in NP$ entonces $L^c \in coNP$, de ahí que, para todo $L_{chequeo} \in P$ y todo polinomio p se tiene que

$$L^c = \{x : \forall y(\langle x, y\rangle \notin L_{chequeo} \vee |y| > p(|x|))\}$$

[4]Del teorema 1.1, a y se le conoce como el certificado.

En consecuencia, para L^c no es posible encontrar un certificado que esté acotado polinomialmente, por lo tanto co-NP está conformado por todos aquellos lenguajes que no tienen un certificado acotado polinomialmente. ♦

1.2.1. Máquinas de Turing Probabilísticas

Es conveniente aclarar el concepto de máquina de Turing probabilística, debido a que la definición de pruebas interactivas se fundamenta en este tipo de máquinas.

Definición 1.6. *Una* máquina de Turing probabilística (MTP) *es una máquina de Turing no determinística en la que:*

1. *El grado de NT es 2, es decir, su árbol de computación es binario.*
2. *Cada paso de la MTN es aleatorio, de ahí que un estado global tiene solo dos posibles sucesos.*
3. *Todas las computaciones sobre una entrada x paran después del mismo número de pasos.*
4. *Existe un nuevo estado de parada agregado a los ya conocidos de aceptación y rechazo, el estado "no se".*

Definición 1.7. *Dada una máquina de Turing probabilística (MTP) y una entrada x se definen:*

$\alpha(MTP, x)$ *como la razón entre el número de computaciones de aceptación, en el árbol de computación asociado a* $MTP(x)$ *y el número total de computaciones de la máquina.*

$\beta(MTP, x)$ *como la razón entre el número de computaciones de rechazo, en el árbol de computación asociado a* $MTP(x)$*, sobre el número total de computaciones de la máquina.*

Las máquinas de Turing probabilísticas se clasifican, de acuerdo a la probabilidad de aceptación y rechazo de una entrada dada, en [Ast04]:

- PP-máquinas
- BPP-máquinas
- R-máquinas
- ZPP-máquinas

En los sistemas de pruebas interactivas, el criterio de aceptación de una entrada es similar al criterio de la BPP-máquina, por lo tanto, se presenta la definición de este tipo de máquinas.

Definición 1.8. *Una máquina de Turing probabilística MTP se dice que es una BPP-máquina si existe una constante $\epsilon \in (0, \frac{1}{2})$ tal que.*

1. *Para cualquier x, se cumple cualquiera de los casos: $\alpha(MTP, x) > \frac{1}{2} + \epsilon$ ó $\beta(MTP, x) > \frac{1}{2} + \epsilon$.*
2. *Para cualquier x, $MTP(x)$ acepta si $\alpha(MTP, x) > \frac{1}{2} + \epsilon$.*
3. *Para cualquier x, $MTP(x)$ rechaza si $\beta(MTP, x) > \frac{1}{2} + \epsilon$.*

1.2.2. Máquina de Turing con Oráculo

Dentro de la teoría de las pruebas interactivas, es de gran utilidad una máquina de Turing que es capaz de resolver cualquier tipo de problemas y cuyo funcionamiento se puede contabilizar como un paso computacional; a esta clase de máquina se le denomina *Máquina de Turing con Oráculo* y se define de la siguiente manera.

Definición 1.9 (Oráculo). *Una máquina de Turing no determinística con oráculo, es una máquina de k-cintas con una cinta objetivo llamada cinta oráculo y tres estados especiales denominados estado de pregunta, estado de aceptación y el estado de rechazo. Por notación la máquina de Turing no determinística NT al combinarla con cualquier oráculo L se denota por NT^L.*

Si una máquina de Turing con oráculo entra al estado de pregunta, después tiene que escribir una cadena en la cinta oráculo y el siguiente estado debe ser el estado de aceptación o el estado de rechazo, dependiendo de si la palabra pertenece o no al lenguaje oráculo.

Para tratar de interpretar el concepto de oráculo, se presentan algunos resultados importantes en los cuales se evidencia la importancia de las máquinas de Turing con oráculo.

Con el fin de establecer una mejor comprensión de la conjetura ¿$P = NP$? Se han obtenido los siguientes resultados relacionados con esta, utilizando principalmente el concepto de oráculo. En el primer teorema, el oráculo se ve como un lenguaje tan poderoso que relativamente, muestra que las clases P y NP coinciden.

TEOREMA 1.3. *[PC94] Sea A un lenguaje definido así*

$$A = \{\langle T, x, 0^k\rangle : T(x) \, acepta \, y \, usa \, a \, lo \, mas \, k - cintas\}.$$

Entonces $P^A = NP^A$.

Demostración. Tenemos que $P^A \subseteq NP^A$, debido a que si $L \in P^A$, entonces existe una máquina de Turing determinística tiempo polinomial T con oráculo que acepta L, cualquier máquina de Turing no determinística tiempo polinomial NT con oráculo también acepta L, en consecuencia $L \in NP^A$.

Ahora se probará que $NP^A \subseteq P^A$. Sea $L \in NP^A$, entonces existe una máquina de Turing no determinística NT con oráculo que decide L, en a lo mas $p(|x|)$ pasos, para algún polinomio p.

Se debe derivar una máquina de Turing determinística T', usando a lo sumo $q(|x|)$ pasos para un polinomio que sea equivalente a la máquina NT con oráculo; una vez hecho esto, es posible encontrar una máquina de Turing determinística tiempo polinomial con oráculo que acepte L.

Para derivar una máquina $T^{'}$ de la máquina NT con oráculo se realiza lo siguiente:
En una máquina de Turing NT se tiene que cada camino de computación requiere a lo mas un número polinomial de cintas y, además, cada vez puede seguir utilizando el mismo número de cintas y con ello, el espacio empleado es siempre el mismo.

Supóngase que la máquina NT pregunta al oráculo la cadena $\langle T^{''}, y, 0^k\rangle$. $T^{'}$ puede responder esta pregunta teniendo $k - cintas$ necesarias para computar $T^{''}(y)$. Es claro que $k \leq p(|x|)$, porque de lo contrario, no sería posible que la pregunta estuviera en la cinta oráculo.

En conclusión, $T^{'}$ responde cualquier pregunta de la forma $\langle T^{''}, y, 0^k\rangle$ en espacio polinomial, así T' decide L usando a lo más $q(|x|)$ pasos. De lo anterior se tiene que, dada una entrada x, una máquina T' escribe en la cinta de pregunta la cadena $\langle T^{''}, x, 0^{q(|x|)}\rangle$, según la derivación, $T^{'}$ acepta de acuerdo a la respuesta del oráculo, es decir, $L \in P^A$. En consecuencia, como L es un lenguaje arbitrario de NP^A, entonces $NP^A \subseteq P^A$. ⧫

El siguiente resultado muestra que por medio de una técnica llamada de diagonalización y la utilización de un oráculo B, las clases relativas P^B y NP^B son clases diferentes.

Técnica de la Diagonalización
Sean $X = \{x_0, x_1, ..., x_n, ...\}$ y $A = \{A_i \in \mathbb{P}(X), i \geq 0\}$ un conjunto contable. El conjunto diagonal D se define como

$$D = \{x_i : x_i \notin A_i, i \geq 0\},$$

entonces $D \neq A_i$ para todo i, es decir, el conjunto D no es contable, a menos que el conjunto X sea finito.

Ejemplo 4.
Sean $X = \mathbb{R}$ y $A_i = \{ni : n \in \mathbb{Z}\}$ para todo $i = 1, 2, 3....$
El conjunto diagonal D será

$$D = \{x_i : x_i \notin A_i\}.$$

De acuerdo a esto se tiene que

$$A_1 = \{..., -3, -2, -1, 0, 1, 2, 3, ...\}$$
$$A_2 = \{..., -6, -4, -2, 0, 2, 4, 6, ...\}$$
$$A_3 = \{..., -9, -6, -3, 0, 3, 6, 9, ...\}$$
$$\vdots$$

Por lo tanto, el conjunto diagonal podría ser

$$D = \{...; -3{,}3;\ -2{,}5;\ -1{,}6;\ 0{,}7;\ 1{,}4;\ 2{,}3; ...\}$$

Al aplicar la técnica anterior a las máquinas de Turing se obtiene:

Dadas $T_1, T_2, ...$ máquinas de Turing con oráculo X, se tiene que

1. $P = \{L : L = L(T_k^X)$ para algún $k\}$
2. Para cualquier $k \geq 0$ y una entrada x, $T_k^X(x)$ se detiene después de a lo mas $n^k + k$ pasos.

También se tiene que para cualquier lenguaje X, se define un nuevo lenguaje denotado L_X así:

$$L_X = \{0^n : \exists x[x \in X \wedge |x| = n]\}$$

El lenguaje L_X está formado por todas las cadenas que pertenecen al lenguaje X y que tienen longitud n.

Se tiene que $L_X \in NP^X$, para cualquier oráculo X; en efecto, comprobar si 0^n pertenece a L_X, es suficiente formar todas las cadenas de longitud n, de manera no determinística y verificar si el oráculo X incluye por lo menos una de ellas.

El siguiente teorema muestra la existencia de un oráculo X, tal que el lenguaje definido como L_X no pertenece a la clase P^X.

TEOREMA 1.4. *[PC94] Existe un oráculo B, tal que $P^B \neq NP^B$.*

Demostración. Mediante la técnica de diagonalización, a cada máquina de Turing determinística con oráculo T se le asocia un entero n, tal que si $T^B(O^n)$ acepta y $B_n = \emptyset$, o $T^B(0^n)$ rechaza y $B_n \neq \emptyset$; entonces ninguna máquina de Turing determinística con oráculo decide L_B.

El lenguaje B se construye por etapas. $B(i)$ denota el lenguaje finito conformado por cadenas adheridas a B después de i-ésimas etapas y n_i denota la longitud máxima de las cadenas de $B(i)$.

Inicialmente, $B(0) = \emptyset$ y $n_0 = 0$, posteriormente, al lenguaje B se le adicionan cadenas teniendo en cuenta lo siguiente:

$$n_i = \text{mín}\{m : m > n_{i-1}^{i-1} + i - 1 \wedge 2^m > m^i + i\}.$$

De acuerdo a lo anterior, si $T_i^{B(i-1)}(0^{n_i})$ acepta, entonces $B(i) = B(i-1)$.

En caso contrario, a y se le asigna la cadena más pequeña de longitud n_i que no ha sido preguntada por $T_i^{B(i-1)}(0^{n_i})$ y entonces a $B(i) = B(i-1) \cup \{y\}$.

El oráculo B lo definimos como $B = \bigcup_{i>0} B(i)$.

Para cada i, la computación de T_i con la entrada 0^{n_i} y el oráculo $B(i-1)$ es la misma que con el oráculo B, puesto que la cadena adherida a B en la i-ésima etapa, es la que T_i no pregunta en la computación y se adhieren hasta completar una longitud de $n_i^i + i$.

Supóngase que exista una máquina de Turing determinística con oráculo B que acepta L_B, entonces existe un entero i, tal que $L_B = L(T_i^B)$, donde $T_i^B(0^{n_i})$ acepta y este oráculo no contiene ninguna cadena de longitud n_i, es decir, $0^{n_i} \notin L_B$. Por otro lado, $T_i^B(0^{n_i})$ rechaza y B contiene al menos una cadena de longitud n_i, esto es $0^{n_i} \in L_B$. En ninguno de los

dos casos, T_i puede decidir si 0^{n_i} pertenece a L_B; por lo tanto, no existe una máquina de Turing T_i^B y con ello se tiene que $L_B \notin P_B$. ♦

1.3. CLASE DE COMPLEJIDAD PSPACE

El resultado más importante relacionado con los sistemas de pruebas interactivas, es aquel donde se muestra la capacidad de la clase IP y se relaciona con una de las clases de complejidad más importantes conocida como $PSPACE$. Esta última se define de la siguiente manera.

Definición 1.10 (Clase $PSPACE$). *Un lenguaje L pertenece a la clase $PSPACE$ si existe un algoritmo determinístico A, que lo decide y un polinomio $p(n)$, tal que la cantidad de almacenamiento requerido por A sobre cualquier entrada x de L es a lo más $p(n)$, donde n es el tamaño de x.*

Dentro de esta clase de complejidad denotada $PSPACE$, también existen algunos problemas que son más difíciles de resolver y forman la clase de complejidad denominada $PSPACE - completo$.

Ejemplo 5.
Se mostrará un problema que pertenece a la clase de complejidad $PSPACE$, al cual se le denomina QBF:

ENTRADA: Dada una Fórmula Booleana f definida sobre n variables $x_1, x_2, ..., x_n$ y una secuencia de n cuantificadores $Q_1, Q_2, ..., Q_n$ que pueden ser universales ($\forall$) o existenciales ($\exists$), que están prefijos a la fórmula f de manera que

$$F = (Q_1 x_1)(Q_2 x_2)...(Q_n x_n) f(x_1, x_2, ..., x_n)$$

PREGUNTA: ¿Es F verdadera?

TEOREMA 1.5. *[PC94] El problema QBF está en $PSPACE$-completo.*

Demostración. Primero se probará que QBF está en $PSPACE$, para ello se utilizará el siguiente algoritmo que utiliza almacenamiento polinomial para decidir F.

Inicio {Entrada: F}
  $n \leftarrow |F|$;
  Mientras $i \leq 2^{p(n)}$ **hacer**
    inicio
      $T_i(F)$ {Asignación de las variables booleanas}
      Si $T_i(F)$ **entonces**
        $i \leftarrow 2^{n+1}$
      sino
        $i \leftarrow i + 1$;
    fin;
  Si $i = 2^{n+1}$ **entonces**
    F es verdadera
  sino
    F es falsa;
Fin.

Puesto que la asignación de las variables booleanas requiere a lo más un número polinomial de celdas respecto a la tamaño de F y como todas las asignaciones tienen la misma longitud, entonces el algoritmo utiliza un número polinomial de celdas para decidir F, es decir, el problema $QBF \in PSPACE$.

Ahora se probará que QBF es $PSPACE - completo$, para ello sea T una máquina que decide un lenguaje en espacio polinomial y sea x una entrada. El objetivo es derivar una reducción tiempo polinomial, la cual transforma la pareja $\langle T, x\rangle$ en una fórmula booleana cuantificada F_x, tal que F_x es verdadera si, y sólo si T acepta x.

Denótese por n la longitud de x y con $s(n)$ el polinomio que limita el espacio requerido por T. De ahí que, $T(x)$ usa a lo más $s^* = s(n)$ celdas en la cinta y ejecuta a lo más $t^* = 2^{cs(n)}$ pasos, para una constante c.

Supóngase que los estados globales son tomados como cadenas binarias de longitud fija L. Así, estas cadenas pueden ser vistas como asignaciones de $L - tuplas$ valores a las variables booleanas. De esta manera la notación $\exists X$, significa que $\exists x_1, ..., \exists x_L$ y así mismo $\forall X$ quiere decir $\forall x_1, ..., \forall x_L$.

Mediante la función *alcanzar*, se realiza la transformación de la computación de la máquina T sobre una entrada x, en una fórmula booleana cuantificada.

Funcion $alcanzar(S, S^{'}, j)$:Booleano;
Inicio
  Si $j = 0$ **entonces**
    si $(S = S^{'})$ **o** (es posible pasar de S a $S^{'}$) **entonces**
      $alcanzar \leftarrow verdadera$
    sino
      $alcanzar \leftarrow falso$
  sino
  inicio
    $alcanzar \leftarrow falso$
    para cada estado global $S^{''}$ **hacer**
      si $(alcanzar(S, S^{''}, j-1))$ **y** $(alcanzar(S^{''}, S^{'}, j-1))$ **entonces**
        $alcanzar \leftarrow verdadera$;
  fin;
Fin.

La transformación mencionada se realiza de acuerdo a lo siguiente, si la máquina T puede alcanzar el estado $S^{'}$ a partir del estado S en a lo más 2^j pasos, entonces la correspondiente fórmula F_j es verdadera. De lo anterior se tiene que T acepta x si, y sólo si, la fórmula

correspondiente a $alcanzar(S_0, S_\Omega, cs^*)$ es verdadera, donde S_0 es el estado inicial, S_Ω es el estado final y cs^* es el total de pasos que realiza T para una entrada x.

Para obtener la fórmula booleana F_j es necesario realizar las siguientes definiciones:

1. $gs(X)$ es verdadera si, y sólo si, X representa un estado global.
2. $producir(X, Y)$ es verdadera si, y sólo si Y es un estado global que puede ser alcanzado a partir de un estado X en a lo más un paso.
3. $ini(X)$ es verdadera si, y sólo si X representa el estado global inicial.
4. $acepta(X)$ es verdadera si, y sólo si X representa el estado global de aceptación.

La fórmula F_j es construida de una manera inductiva de la siguiente manera. Para $j = 0$, la correspondiente fórmula $F_0(X, Y)$ se expresa como

$$F_0(X, Y) = gs(Y) \wedge producir(X, Y)$$

El paso inductivo se define como $F_{j+1}(X, Y)$ tal que

$$F_{j+1}(X, Y) = (\exists Z)(F_j(X, Z)) \wedge (F_j(Z, Y))$$

Sin embargo, la aproximación utilizada en el algoritmo *alcanzar* no completa el objetivo, puesto que la fórmula F_{j+1} crece exponencialmente con j. Esto se resuelve reformulando F_{j+1} como

$$F_{j+1}(X, Y) = (\exists Z)(\forall X^{'})(\forall Y^{'})[((X^{'} = X \wedge Y^{'} = Z) \vee (X^{'} = Z \wedge Y^{'} = Y)) \rightarrow F_{j+1}(X^{'}, Y^{'})]$$

Las dos definiciones de F_{j+1} son equivalentes y además, en la última, la longitud de F_{j+1} crece polinomialmente respecto a j.

Finalmente, la fórmula F_x se define como

$$F_x = (\exists X)(\exists Y)[ini(X) \wedge acepta(Y) \wedge F_{cs^*}(X, Y)]$$

Por lo tanto se ha construido una fórmula booleana cuantificada a partir de una máquina T y una entrada x, de tal manera que si T acepta x la fórmula booleana cuantificada es verdadera, así mismo T rechaza x equivale a que la fórmula booleana sea falsa. ♦

1.4. TEORÍA DE LA APROXIMABILIDAD

Debido a que solucionar algunos problemas de optimización de manera eficiente[5] es una tarea difícil de realizar, se utilizan algoritmos de aproximación, los cuales aunque no retornan la solución exacta, dan una respuesta aproximada a la óptima.

Algunos problemas de optimización tienen asociado un problema de decisión que es NP-completo, y por tanto no se conoce algoritmo tiempo polinomial que los resuelva, a este tipo de problemas se les denomina NP-duros. Dentro de este grupo de problemas, la teoría de aproximabilidad estudia principalmente aquellos que hacen parte de la clase NPO. Dentro de esta gran clase, se agrupan los problemas de aproximación en subclases denotas por APX, $PTAS$, y $FPTAS$ que se diferencian por el factor de aproximación y cuyas definiciones se encuentran formalmente en [HB03]. Para obtener mayor claridad en los conceptos que se presentaran posteriormente es conveniente precisar algunas definiciones.

Definición 1.11 (CLASE NPO). *Un problema de optimización $A = \langle I, sol, m, obj\rangle$ pertenece a la clase NPO si:*

1. *I es el conjunto de instancias de A y es reconocible en tiempo polinomial, esto es, existe un algoritmo tiempo polinomial que decide para cualquier entrada, si está corresponde a una instancia del problema.*

2. *Dada una instancia $x \in I$, $sol(x)$ denota el conjunto de soluciones posibles de x. Estas soluciones deben ser cortas, esto es, existe un polinomio p tal que, para todo $y \in sol(x)$, $|y| \leq p(|x|)$. Además, para todo x y para todo y tal que $|y| \leq p(|x|)$, es decidible en tiempo polinomial si $y \in sol(x)$.*

3. *m es una función tal que para cualquier instancia x en I, $m(x, y) \in \mathbb{Z}^+$ representa el costo o valor de la solución $y \in sol(x)$. La función m es computable en tiempo polinomial y es llamada función objetivo.*

[5]Eficiente hace referencia a tiempo polinomial en la longitud de la entrada.

4. *obj* $\in \{$*máx,mín*$\}$ *determina si el problema es de maximización o minimización.*

Resolver un problema de optimización $A = \langle I, sol, m, obj \rangle$ consiste en encontrar para cada $x \in I$, una solución $y \in sol(x)$ tal que

$$\begin{aligned} m(x, y) &= obj\{m(x, z) : z \in sol(x)\} \\ &= opt(x) \end{aligned}$$

Con el fin de comprender mejor la definición anterior y como aplicación de la teoría de aproximabilidad, están estos dos ejemplos denominados $MAX3SAT$ y $MAXCLIQUE$, que pertenecen a la clase NPO y cuyos problemas de decisión asociados, SAT y $CLIQUE$ respectivamente, pertenecen a la clase NP-completo.

Ejemplo 6.

Problema de **MAX 3SAT**

INSTANCIA: Una colección $C = \{c_1, c_2, ..., c_m\}$ de cláusulas sobre un conjunto finito U de variables booleanas tales que $|c_i| = 3$ para $1 \leq i \leq m$.

SOLUCIÓN: Una asignación de verdad para U.

MEDIDA: Número de cláusulas que se satisfacen con la asignación de verdad.

OBJETIVO: Maximizar.

Ejemplo 7.

Problema de **MAX CLIQUE**

INSTANCIA: Un grafo $G = (N, E)$, N un conjunto de nodos y E un conjunto de aristas y un $k \in \mathbb{Z}^+$

SOLUCIÓN: Un grafo completo de tamaño k.

MEDIDA: Número de nodos que conforman el grafo completo.

OBJETIVO: Maximizar.

Definición 1.12 (CLASE APX). *Sea A un problema en NPO. A pertenece a la clase APX si es ϵ-aproximable para alguna constante $\epsilon > 1$.*

Esta definición quiere decir que si $A \in APX$, entonces existe un algoritmo T que es ϵ-aproximable tal que

- Si A es un problema de maximización entonces $\frac{1}{\epsilon} opt(x) \leq m(x, T(x)^6)$
- Si A es un problema de minimización entonces $m(x, T(x)) \leq \epsilon opt(x)$

Se debe tener claro, que no todo problema en NPO es ϵ-aproximable para alguna constante ϵ, a menos que $P = NP$[7].

[6] $T(x)$ denota lo que retorna el algoritmo T con la instancia x.

[7] Esta afirmación se demuestra posteriormente en el teorema 2.10

2. SISTEMA DE PRUEBAS INTERACTIVAS

En las últimas décadas se dice que las clases de complejidad alcanzaron un gran desarrollo, debido a que se estudiaron diferentes modelos de computación entre los cuales estaba la computación probabilística; que se inicio a su vez con un test para probar la primalidad. Esto llevo a definir las clases de complejidad probabilísticas y mas adelante los sistemas de pruebas interactivas que permitieron encontrar resultados para la aproximación de ciertos problemas NP-completos.

Los sistemas de pruebas interactivas se consideran de forma muy simple como dos máquinas de Turing, denominadas *El Probador* y *El verificador*, las cuales se denotan **P** y **V** respectivamente. Las dos máquinas pueden intercambiar mensajes, aunque es conveniente limitar tanto el número como la longitud de ellos.

El intercambio de mensajes se realiza en dos cintas de comunicación, etiquetadas *P-a-V* y *V-a-P*. La primera cinta es sólo escritura para **P** y sólo lectura para **V**, así mismo, la cinta *V-a-P* es de sólo escritura para **V** y sólo lectura para **P**. Tanto el probador como el verificador tiene su propia cinta de trabajo y ambos pueden leer la misma entrada.

Definición 2.13. *Al intercambio de mensajes entre el probador y el verificador se le denomina el* ***protocolo*** *del sistema y el número de mensajes los* ***rounds*** *del sistema.*

La siguiente gráfica representa un esquema de un sistema de pruebas interactivas; es claro que las dos máquinas tienen la misma entrada y que cada máquina es independiente de la otra, es decir, cada una realiza su propia computación.

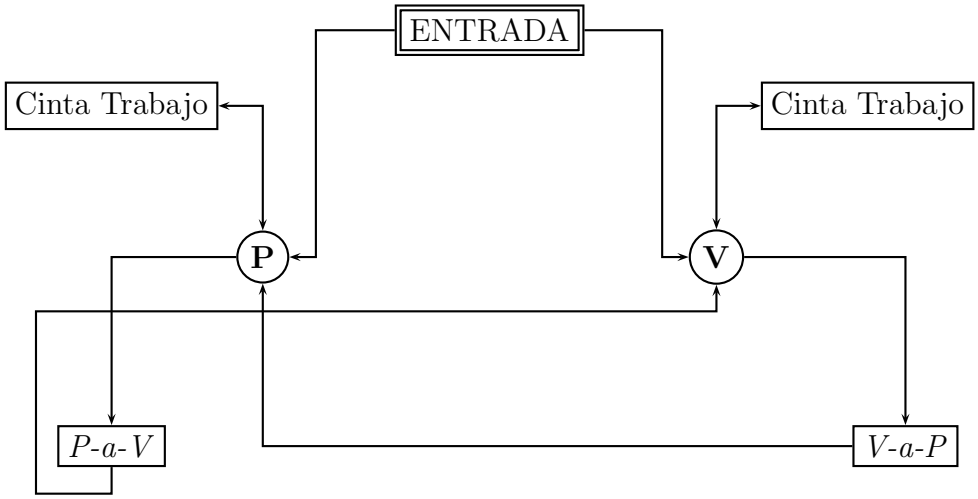


Figura 2.1: Sistema de Pruebas Interactivas

Los sistemas de pruebas interactivas se clasifican en **sistemas de pruebas interactivas determinísticas** y en **sistemas de pruebas interactivas probabilísticos**; esta clasificación se debe exclusivamente al tipo de verificador que se utilice, esto es, el sistema es determinístico si el verificador es un máquina de Turing determinística, de igual forma, el sistema se considera probabilístico si el verificador es una máquina de Turing probabílistica.

2.1. SISTEMA DE PRUEBAS INTERACTIVAS DETERMINÍSTICO

Como ya se estableció anteriormente, un sistema de pruebas interactivas determinístico esta determinado por una pareja de máquinas de Turing que intercambian una serie de mensajes con el fin de resolver un problema determinado, este sistema tiene las siguientes características:

1. El verificador V es una máquina de Turing determinística tiempo polinomial.

2. El probador P es una máquina de Turing sin límite computacional, tanto en tiempo como en espacio en memoria.

3. V inicia la computación, las dos máquinas se activan y toman sus turnos. Cuando una máquina está activa, puede realizar la computación internamente, leer y escribir en las cintas correctas y además enviar un mensaje a la otra máquina en la cinta de comunicación apropiada.

4. Tanto la longitud como el número de mensajes intercambiados entre P y V debe ser acotado por un polinomio conveniente en la longitud de la entrada.

5. V termina la computación interactiva en estado de aceptación o en estado de rechazo.

El criterio de aceptación del sistema de pruebas (P, V) descrito está completamente determinado por el verificador V, esto es, el sistema acepta una entrada x si después de toda la computación interactiva, el verificador V termina en estado de aceptación. Así mismo, el sistema (P, V) rechaza una entrada x, cuando el verificador V finaliza la computación interactiva en estado de rechazo.

La siguiente definición permite comprender cuándo un lenguaje dado L es aceptado por un sistema de pruebas interactivas determinístico.

Definición 2.14. *Un lenguaje L admite un sistema de pruebas interactivo determinístico, si existe un verificador V tal que:*

1. *Existe un probador P^* tal que el sistema (P^*, V) acepta todas las $x \in L$.*

2. *Para todos los probadores P, el sistema (P, V) rechaza todas las $x \notin L$.*

De la anterior definición se tiene que si x pertenece al lenguaje L, entonces existe una manera fácil de probar este hecho de tal forma que el verificador V lo acepte. Por otro lado, si x no pertenece al lenguaje L, entonces no es posible convencer al verificador de lo contrario, es decir, no existe probador P que pueda engañar al verificador V.

2.1.1. La Clase DIP

Definición 2.15. *El conjunto de todos los lenguajes L, que admiten un sistema de pruebas interactivas determinístico, forman una nueva clase de complejidad denotada* ***DIP****.*

El siguiente resultado presenta la relación de la clase DIP con una de las clases más importantes de la teoría de la complejidad computacional, la clase NP.

TEOREMA 2.6. *[PC94] Un lenguaje L admite un sistema de pruebas interactivas determinístico si, y sólo si, $L \in NP$, es decir $DIP = NP$.*

Demostración. Se mostrará primero que $NP \subseteq DIP$. Sea L un lenguaje que pertenece a la clase NP. De acuerdo con el teorema 1.1, la clase NP se puede ver como el conjunto de lenguajes que tienen un componente de verificación tiempo polinomial. Por lo tanto, si $x \in L$, entonces el probador P,(que no tiene límite computacional) calcula y y la envía como componente de verificación al verificador V; como $L \in NP$, entonces existe un polinomio p tal que $|y| \leq p(|x|)$, luego, V que es una máquina de Turing determinística corre en tiempo polinomial respecto a la longitud de x y comprueba que y es el certificado. Así mismo, si $x \notin L$, entonces el probador no puede convencer al verificador de que x está en L, puesto que no le es posible hallar un componente de verificación tiempo polinomial, por tanto, si L es un lenguaje que pertenece a la clase NP, entonces L admite un sistema de pruebas interactivas determinístico, es decir L está en la clase DIP.

Ahora se probará que $DIP \subseteq NP$. Sea L un lenguaje que admite un sistema de pruebas interactivas determinístico y sea V un verificador para L. Dada una entrada x, por la definición de sistema de pruebas interactivo, se tiene que el número de mensajes intercambiados entre P y V es polinomial con respecto a la longitud de x. Es por ello que se puede derivar una máquina de Turing no determinística tiempo polinomial P^*, con respecto a los mensajes del probador P. La derivación se realiza, tomando el árbol de computación del probador P y escogiendo los caminos para el árbol de computación de P^*, de la siguiente manera: si $x \in L$ entonces por lo menos P^* debe tener el camino de computación que corresponde al probador que para en el estado de aceptación. Por otro

lado, si $x \notin L$ entonces se tiene que todas las computaciones paran en estado de rechazo, dado que el probador no puede convencer al verificador. Por tanto, se tiene que existe una máquina de Turing no determinística que acepta L en tiempo polinomial, esto es, L pertenece a la clase NP. ♦

El teorema anterior permite establecer que en realidad la clase DIP, únicamente muestra que los problemas que se resuelven mediante algoritmos no deterministas tiempo polinomial, también se pueden ver como problemas que se resuelven con un sistema de pruebas interactivas determinístico. Por otro lado, se tiene que mediante este tipo de sistemas no se ha logrado un avance significativo dentro del campo de la complejidad computacional; es por ello que existe un sistema de pruebas interactivas con el cual se obtienen resultados importantes y una nueva caracterización a algunas clases de complejidad.

2.2. SISTEMAS DE PRUEBAS INTERACTIVAS

Para obtener mayor capacidad en los sistemas de pruebas interactivas se cambia la máquina de Turing determiniística tiempo polinomial que hace las veces de verificador, por una máquina de Turing Probabilística tiempo polinomial. De esta forma el sistema de pruebas interactivas probabilístico o simplemente el sistema de pruebas interactivas, mantiene las mismas características del sistema descrito en la sección anterior, aunque es conveniente aclarar que el criterio de aceptación cambia de determinístico a probabilístico, por lo tanto se tiene que, dado un verificador probabilístico V:

- *Existe un probador P^*, tal que el sistema (P^*, V) acepta $x \in L$ con una alta probabilidad.*
- *Ningún probador P, puede convencer al verificador V a aceptar con una probabilidad considerable, una entrada x dado que $x \notin L$.*

Este criterio de aceptación es similar al de la BPP-máquina[8]. Luego, los caminos de computación de un sistema de pruebas interactivas (P, V) para una entrada x están dados por un árbol binario. De ahí que $\alpha(P, V, x)$ denota el número de caminos de aceptación sobre el número total de caminos en el árbol de computación, respectivamente $\beta(P, V, x)$.[9]

Formalmente un sistema de pruebas interactivas se define así:

Definición 2.16. *Un* ***sistema de pruebas interactivo*** *para un lenguaje L es una pareja (P, V), probador y verificador, con las siguientes propiedades:*

1. *Si $x \in L$ entonces V acepta con una probabilidad al menos de $2/3$.*
2. *Si $x \notin L$ entonces para cualquier probador P^*, V acepta con una probabilidad a lo más de $1/3$.*
3. *Para cualquier x, el tiempo total de computación de V en $(P, V)(x)$ es a lo mas polinomial en la longitud de x ($|x|$).*

En otras palabras se tiene que un sistema (P, V) acepta si $\alpha(P, V, x) > 2/3$ y rechaza si $\beta(P, V, x) > 2/3$. La siguiente proposición muestra que es posible cambiar la constante $2/3$ por cualquier constante mayor que $1/2$.

PROPOSICIÓN 2.1. *[Tre00] Dado L un lenguaje, si L admite un sistema de pruebas interactivas, el error de probabilidad es de a lo mas $2^{-p(\bullet)}$, para cualquier polinomio $p(\bullet)$.*

Demostración. Sea L un lenguaje que admite un sistema de pruebas interactivas. Se corre k veces el sistema y se toman las terminaciones que son mayoría. Denotamos con X la variable aleatoria que representa el número de computaciones de aceptación. De acuerdo a esto, X es la suma de k eventos independientes, por lo tanto es posible utilizar la distribución de Bernoulli donde la probabilidad es al menos de $2/3$.

[8] La definición de este tipo de máquina de Turing probabilística es la 1.8.

[9] α y β tienen el mismo significado que en la definición 1.7

El error se presenta cuando $X \leq \frac{k}{2}$, es decir, si de todas las computaciones, las que son de aceptación son menos de la mitad. Así entonces,

$$Pr[error] \leq Pr[X \leq k/2] \leq \sum_{i=0}^{k/2} \binom{k}{i} \left(\frac{2}{3}\right)^i \left(\frac{1}{3}\right)^{k-i}$$

Por otro lado se tiene que:

$$2 = \frac{\frac{2}{3}}{\frac{1}{3}}$$

por lo tanto,

$$(2)^{\frac{k}{2}-i} = \left(\frac{2/3}{1/3}\right)^{\frac{k}{2}-i} > 1.$$

De lo anterior, se tiene que

$$\sum_{i=0}^{k/2} \binom{k}{i} \left(\frac{2}{3}\right)^i \left(\frac{1}{3}\right)^{k-i} \leq \sum_{i=0}^{k/2} \binom{k}{i} \left(\frac{2}{3}\right)^i \left(\frac{1}{3}\right)^{k-i} \left(\frac{2/3}{1/3}\right)^{\frac{k}{2}-i}$$

$$= \sum_{i=0}^{k/2} \binom{k}{i} \left(\frac{2}{3}\right)^i \left(\frac{1}{3}\right)^{k-i} \left(\frac{2}{3}\right)^{\frac{k}{2}-i} \left(\frac{1}{3}\right)^{i-\frac{k}{2}} = \sum_{i=0}^{k/2} \binom{k}{i} \left(\frac{2}{3}\right)^{\frac{k}{2}} \left(\frac{1}{3}\right)^{\frac{k}{2}}$$

$$= \left(\frac{2}{9}\right)^{\frac{k}{2}} \sum_{i=0}^{k/2} \binom{k}{i}$$

Por el teorema del Binomio $(a+b)^k = \sum_{i=0}^{k} \binom{k}{i} a^{k-i} b^i$, si $a = b = 1$, entonces $\sum_{i=0}^{k} \binom{k}{i} = 2^k$; por lo tanto,

$$\left(\frac{2}{9}\right)^{\frac{k}{2}} \sum_{i=0}^{k/2} \binom{k}{i} \leq \left(\frac{2}{9}\right)^{\frac{k}{2}} 2^k = \left(\frac{\sqrt{8}}{3}\right)^k$$

Como $0 < \frac{\sqrt{8}}{3} < 1$, entonces

$$\left(\frac{\sqrt{8}}{3}\right)^{\frac{k}{12}} < 2^{\frac{k}{12}}$$

Así, k puede ser un polinomio cualquiera. ♦

2.2.1. La Clase IP

Definición 2.17. *El conjunto de todos los lenguajes que admiten un sistema de pruebas interactivo, forma una nueva clase de complejidad denotada por* ***IP.***

En los sistemas de pruebas interactivos en general, el generador de la aleatoriedad del verificador es *Privado*, es decir, el probador sólo recibe la información ya computada por el verificador, esto permite que el probador no pueda engañar al verificador fácilmente; pero existen sistemas de pruebas en los cuales el probador tiene información acerca de la aleatoriedad del verificador, a estos sistemas se les denomina *sistemas de pruebas interactivas de Arthur-Merlin*. El verificador es llamado Arthur y al probador se le conoce como Merlin. Arthur realiza la computación aleatoria y envía los resultados a Merlin; en este sistema se realiza un último paso en la computación para que el verificador decida si acepta o rechaza la entrada. Además Arthur provee a Merlin de los bits aleatorios suficientes en toda la prueba. Los sistemas de pruebas de Arthur-Merlin son llamados algunas veces pruebas interactivas *con aleatoriedad pública,* por el hecho de que Merlin puede ver todos los resultados aleatorios de Arthur. El conjunto de lenguajes que admiten este tipo de pruebas, forman una clase de complejidad conocida como clase **AM.**

Si un lenguaje L admite un sistema de pruebas interactivas, entonces tiene un sistema de pruebas de Arthur-Merlin y además, el número de pasos en la computación es el mismo. Es decir, las dos versiones de sistemas interactivos son equivalentes. Por lo tanto, $IP = AM$ [Dan96].

Ahora se presentan algunos problemas que admiten un sistema de pruebas interactivas, es decir problemas que pertenecen a la clase IP. Los dos ejemplos descritos a continuación se caracterizan por definirse en términos del complemento de otros problemas tales como el problema de grafos isomorfos y el problema de residuos cuadrt́icos. Dados dos grafos, decidir si ellos son isomorfos es un problema que está en la clase NP, debido a que existe un certificado (la función biyectiva) acotado polinomialmente en la longitud de la entrada; este problema no se conoce si hace parte de los NP-completos. Por otro lado, dado dos

números primos relativos, decidir si son residuos cuadráticos es un problema que no se conoce si está en la clase NP.

Ejemplo 8.
Descripción de un sistema de pruebas interactivas para *grafos no isomorfos.*

Primero se definirá el problema original, este es el problema de grafos isomorfos, el cual es un problema que pertenece a la clase NP y no se conoce si pertenece a la clase P, o NP-completo.

Dados dos grafos $G_0 = (N_0, E_0)$ y $G_1 = (N_1, E_1)$ se dice que son isomorfos si existe una función biyectiva, $f : N_0 \to N_1$ tal que $\langle x, y\rangle \in E_0$ si, y sólo si $\langle f(x), f(y)\rangle \in E_1$, para todo par $x, y \in N_0$; se denota $G_0 \cong G_1$. Para resolver este problema, no se conoce algoritmo eficiente, pero sí existe un verificador que corre en tiempo polinomial con respecto a la longitud de la entrada, en este caso el verificador es la función biyectiva, es por ello que este problema esta en la clase NP.

De acuerdo a lo anterior, el complemento del problema de grafos isomorfos se define así:

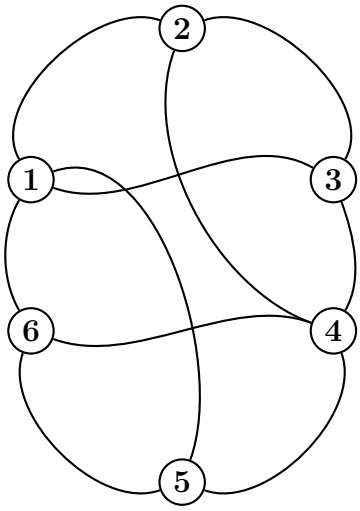
ENTRADA: Dos grafos, $G_0 = (N_0, E_0)$ y $G_1 = (N_1, E_1)$.

PREGUNTA: No existe una función biyectiva, $f : N_0 \to N_1$ tal que $\langle x, y\rangle \in E_0$ si, y sólo si, $\langle f(x), f(y)\rangle \in E_1$ para todo par $x, y \in N_0$, es decir que $G_0 \ncong G_1$.

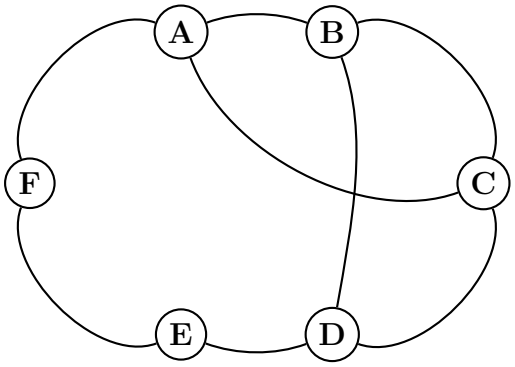
Este último problema admite un sistema de pruebas interactivas, que se describe mediante los siguientes algoritmos, tanto para el verificador como para el probador, respectivamente.

Por definición de sistemas de pruebas interactivas, se tiene que el verificador es una máquina de Turing Probabilística tiempo polinomial, por lo tanto, el algoritmo que representa a esta máquina es probabilístico.[10]

[10]La definición formal de esta clase de algoritmos se encuentra en [Ast04]



Grafo 0



Grafo 1

Figura 2.2: Grafos No-isomorfos

Inicio {Entrada:G_0, G_1}
  $s \leftarrow$verdadero;
  repita {dos veces}
    inicio
      $i \leftarrow$**random**$(0, 1)$;
      cree aleatoriamente un grafo H isomorfo a G_i;
      transmita H;
      reciba j;
      si $i \neq j$ **entonces**
        $s \leftarrow$ falso;
    fin;
  si s **entonces**
    acepte
  sino
    rechace;
Fin.

El siguiente algoritmo representa el probador del sistema. Él realiza la prueba de isomor-

fismo entre dos grafos de una manera eficiente[11], aunque este problema no se conoce que esté en la clase P, esto se debe a la capacidad que por definición tiene el probador, porque recuérdese, que es una máquina de Turing no determinística sin límite computacional, tanto en tiempo como en espacio en memoria.

Algoritmo del **probador** del sistema de pruebas interactivas para grafos no isomorfos.

Inicio {Entrada: G_0, G_1}
    reciba H;
    si H es isomorfo a G_0 **entonces**
        $j \leftarrow 0$
    sino
        $j \leftarrow 1$;
    transmita j;
Fin.

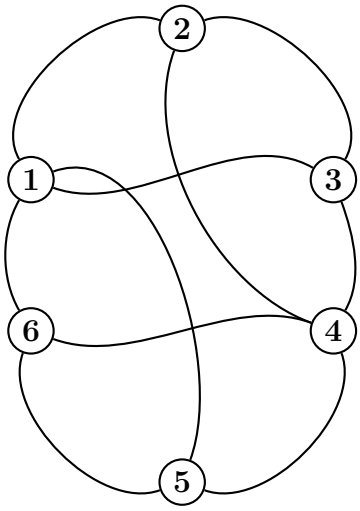
Para tener una mejor comprensión del funcionamiento del sistema, se realizará un seguimiento de los algoritmos.

Sean los grafos G_0 y G_1, formados por los conjuntos de vértices $N_0 = \{1, 2, 3, 4, 5\}$ y $N_1 = \{A, B, C, D, E\}$ respectivamente, $figura$ 2.2.

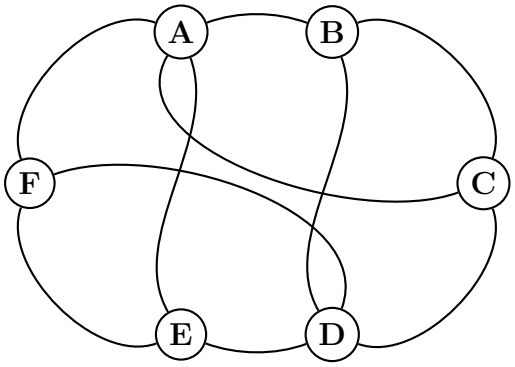
Es claro que los grafos G_0 y G_1 no son isomorfos, porque tienen distinto número de aristas, por lo tanto no existe una biyección entre ellos.

Al hacer el seguimiento, el verificador inicia la computación, por tanto la variable booleana s es verdadera, luego, elige aleatoriamente el valor de i, que puede ser 0 o 1 y genera un grafo isomorfo al grafo G_i. Este procedimiento lo realiza dos veces. Por ejemplo, supóngase que se tomó $i = 0$, entonces se crea un grafo H isomorfo a G_0 y se envía al probador. El probador comprueba si el grafo H es isomorfo a G_0, como esto es correcto, entonces a j

[11]Eficiente, es porque sólo se cuenta como un paso, su tiempo de complejidad



Grafo 0



Grafo 1

Figura 2.3: Grafos Isomorfos

le asigna 0 y la envía al verificador. El verificador compara si $i \neq j$ y realiza el mismo procedimiento. Si el verificador escoge aleatoriamente $i = 1$, genera H isomorfo a G_1 y lo envía al probador. El probador vuelve a comparar H con G_0, como no son isomorfos entonces a j le asigna 1 y lo envía al verificador, este último compara nuevamente si $i \neq j$, como son iguales, se tiene que el valor de s nunca cambió, por lo tanto, el verificador acepta y con ello el sistema de pruebas acepta la entrada, lo que quiere decir que los grafos G_0 y G_1 no son isomorfos.

Si de entrada los grafos G_0 y G_1 son isomorfos (ver *figura* 2.3) y el verificador escoge aleatoriamente $i = 0$ las dos veces, entonces el verificador acepta la entrada (puesto que s nunca cambia su valor inicial), lo que quiere decir que para el sistema los dos grafos no son isomorfos. Esto es un error puesto que de entrada se tiene que los dos grafos son isomorfos.

El sistema descrito anteriormente forma un sistema de pruebas interactivas, puesto que para el verificador V existe un probador P^* tal que si $G_0 \ncong G_1$ entonces el sistema (P^*, V) acepta la entrada con una probabilidad de 1. Por otro lado si $G_0 \cong G_1$ la probabilidad de error es igual a $1/4$, debido a que el probador no sabe a qué grafo es isomorfo H. Como V tiene que elegir entre dos grafos, entonces la probabilidad es $1/2$ y como repite dos veces el procedimiento, entonces el error es a lo más $1/4$. Este error probabilístico también podría

considerarse como la probabilidad de elegir la pareja (0,0) entre las cuatro posibilidades ((0,0),(0,1),(1,0)(1,1)).

Ejemplo 9.

Otro problema que se resuelve mediante un sistema de pruebas interactivas es el de *no residuos cuadráticos*, que es el complemento del problema estudiado en teoría de números denominado *residuos cuadráticos*, el cual se define así:

ENTRADA: Dos enteros n, m con $n < m$ y $(m, n) = 1$

PREGUNTA:¿Existe un entero $x \in \mathbb{Z}_m$ tal que: $x^2 \cong n$ *(mód m)*?

El complemento del problema anterior admite un sistema de pruebas interactivas, se denomina el problema de *no residuos cuadráticos* y se define de la siguiente manera:

ENTRADA: Dos enteros n, m con $n < m$ y $(m, n) = 1$

PREGUNTA:¿Es cierto que para todo $x \in \mathbb{Z}_m$, $x^2 \ncong n$ *(mód m)*?

Para resolver el problema de los no residuos cudráticos se han establecido los siguientes algoritmos, que se presentan a continuación, tanto para el verificador como para el probador.

Algoritmo del **Verficador** para el sistema de pruebas interactivas del problema de no residuos cuadráticos.

Inicio {Entrada: n,m}

$k \leftarrow |m|$; {$|m|$ tamaño en bits del número m}

Escoger aleatoriamente $z_1, z_2, ..., z_k$ tales que $z_i < m$ y $(z_i, m) = 1$;

Escoger aleatoriamente $b_1, b_2, ..., b_k$ con $b_i \in \{0, 1\}$;

Para $i \leftarrow 1$ hasta k hacer

Si $b_i = 1$ entonces

$w_i \leftarrow {z_i}^2 mod m$

sino

$w_i \leftarrow n{z_i}^2 mod m$;

Transmita $w_1, w_2, ..., w_k$;

Reciba $c_1, c_2, ..., c_k$;

Mientras $(b_i \neq c_i)$ y $(j <> k)$ hacer

inc(j);

Si $(j = k)$ y $(b_k = c_k)$;

acepte

sino

rechace;

Fin.

Algoritmo del **probador** del sistema de no residuos cuadráticos.

Inicio {Entrada: n,m}

Reciba $w_1, w_2, ..., w_k$;

Para $i \leftarrow 1$ hasta k hacer

Si $a^2 \cong w(modm)$ entonces

$c_i \leftarrow 1$

sino

$c_i \leftarrow 0$;

Transmita $c_1, c_2, ..., c_k$;

Fin.

Si se realiza un seguimiento a los algoritmos anteriores, tomando por ejemplo a $n = 5$ y $m = 18$, una posible solución podría estar dada por

Z denota el vector formado por los z_i y supóngase que se generaron en este orden

$$Z = (12,\ 1,\ 8,\ 2,\ 7)$$

Se denotará por B el vector formado por los b_i, y supóngase que, escogidos aleatoriamente, se obtuvo este orden

$$B = (0,\ 1,\ 1,\ 0,\ 0,)$$

Al realizar el seguimiento de los algoritmos se obtiene:

$$w_1 \leftarrow 0 \qquad c_1 = 0$$

$$w_2 \leftarrow 1 \qquad c_2 = 1$$

$$w_3 \leftarrow 8 \qquad c_3 = 1$$

$$w_4 \leftarrow 2 \qquad c_1 = 0$$

$$w_5 \leftarrow 11 \qquad c_5 = 0$$

Los c_i denotan los valores que devuelve el probador después de realizar la computación respectiva. Es claro que $b_i = c_i$, para todo $i = 1, ..., 5$, por lo tanto, el sistema de pruebas interactivo acepta y en consecuencia se concluye que 5 no es residuo cuadrático modulo 18.

Ahora se mostrará que el problema de los no residuos cuadráticos pertenece a la clase IP.

Supóngase que n y m, de entrada, no son residuos cuadráticos, es decir que no existe $x \in \mathbb{Z}_m$ tal que: $x^2 \cong n(modm)$, esto es, $x^2 \cong n(modm) \forall x \in \mathbb{Z}_m$.

Se tiene que, existe un probador P^* que siempre retorna las respuestas correctas y por lo tanto, el sistema acepta la entrada con una probabilidad de 1, en efecto, si $b = 1$, entonces $w = z^2(modm)$, por tanto, el probador envía $c = 1$, puesto que existe $a \in \mathbb{Z}_m$, $a = z$ tal que $a^2 \cong w(modm)$.

Si $b = 0$, entonces $w = nz^2(modm)$. Para este caso se tiene que w no es residuo cuadrático módulo m, porque si w es un residuo cuadrático módulo m, entonces existe $y \in \mathbb{Z}_m$ tal que

$$y^2 \cong w(modm).$$

Al reemplazar w se tiene que

$$y^2 \cong nz^2(modm).$$

Por otro lado, $\mathbb{Z}_m$ forma un grupo respecto a la multiplicación y como $(z, m) = 1$ entonces existe $z^{-1} \in \mathbb{Z}_m$, luego

$$y^2(z^{-1})^2 \cong n(modm).$$

Como $y, z^{-1} \in \mathbb{Z}_m$, entonces $(yz^{-1})^2 \in \mathbb{Z}_m$ y con ello n es residuo cuadrático módulo m, lo cual es una contradicción puesto que n es no residuo cudrático módulo m.

En consecuencia, si n y m de entrada no son residuos cuadráticos, entonces el sistema (P^*, V) acepta con una probabilidad de 1.

Supóngase ahora que n y m son residuos cuadráticos, entonces existe $x \in \mathbb{Z}_m$ tal que

$$x^2 \cong n(modm).$$

Por otro lado, se tiene que $w = z^2(modm)$ o $w = nz^2(modm)$. En los dos casos w es residuo cuadrático módulo m puesto que para el primer caso, existe $a \in \mathbb{Z}_m$, $a = z$. Y para el segundo caso, como $x^2 \cong n(modm)$ y $w = nz^2(modm)$ entonces existen $k_1, k_2 \in \mathbb{Z}$ tales que

$$x^2 - n = k_1 m \, y \, w - nz^2 = k_2 m,$$

luego,

$$w = (x^2 - k_1 m)z^2 + k_2 m$$

$$w = x^2 z^2 - k_1 m z^2 + k_2 m.$$

Sea $k \in \mathbb{Z}$, $k = z^2 k_1 - k_2$ entonces

$$w = (xz)^2 - km.$$

Por tanto, existe $a \in \mathbb{Z}_m$ $a = xz$ tal que

$$a^2 \cong w(modm).$$

En conclusión, si n y m son residuos cuadráticos cualquier probador P no diferencia los w_i puesto que siempre son residuos cuadráticos, por lo tanto el error probabilístico ocurre cuando los b_i son todos uno y se estima en $\frac{1}{2^k}$ donde k es el tamaño de m. Debido a los que b_i forman un vector de k componentes, las cuales solo toman valores de 0 o 1. Como $0 < |n| \leq |m|$, entonces $|m| = k \geq 2$. Es decir, el error probabilístico es de a lo más $\frac{1}{4}$.

2.3. ARITMETIZACIÓN DE FÓRMULAS BOOLEANAS

La idea básica de la aritmetización es que al aplicar está técnica a una fórmula booleana se obtenga un polinomio y al evaluar el polinomio se decida si la fórmula es satisfactible o no. Además, por medio de la aritmetización de fórmulas booleanas es posible demostrar que todos los lenguajes que pertenecen a co-NP admiten un sistema de pruebas interactivas, es decir, co-$NP \subseteq IP$. Esta técnica puede tomarse como una función, dado que a cada fórmula booleana se le asigna un único polinomio. El resultado más importante de esta sección es el que muestra la capacidad de las pruebas interactivas, puesto que se muestra que $IP = PSPACE$[12].

2.3.1. Técnica de Aritmetización

Mediante la técnica de aritmetización de una fórmula booleana f en FNC, donde se tienen tres literales por cláusula, se obtiene un polinomio que se le denomina *fórmula aritmética* A_f, el cual se halla de la siguiente forma:

1. *Aritmetización de un literal:* $A_{x_i} = 1 - z_i$ donde x_i es una variable booleana y z_i es una variable entera; de la misma forma, $A_{\sim x_i} = z_i$.

2. *Aritmetización de una cláusula:* $A_{l_1 \vee l_2 \vee l_3} = 1 - A_{l_1} A_{l_2} A_{l_3}$ donde l_1, l_2, l_3 son literales.

3. *Aritmetización de f :* $A_f = A_{c_1 \wedge ... \wedge c_m} = A_{c_1} ... A_{c_m}$ donde $c_1, ..., c_m$ son las cláusulas de f.

A continuación se aplica la técnica de aritmetización a una fórmula f en forma normal conjuntiva que tiene cuatro cláusulas.

Ejemplo 10.

Sean

$$U = \{x_1, x_2, x_3, x_4\} \; y \; f = (c_1 \wedge c_2 \wedge c_3 \wedge c_4)$$

[12]Esta clase de complejidad está descrita de manera formal en la definición 1.10.

donde

$$c_1 = (x_1 \vee x_2 \vee \sim x_3)\ ,\ c_2 = (x_2 \vee \sim x_3 \vee \sim x_4),$$

$$c_3 = (x_1 \vee x_3 \vee x_4)\ ,\ c_4 = (\sim x_1 \vee \sim x_2 \vee x_4)$$

por tanto,

$$f = (x_1 \vee x_2 \vee \sim x_3) \wedge (x_2 \vee \sim x_3 \vee \sim x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\sim x_1 \vee \sim x_2 \vee x_4)$$

La aritmetización por cada una de las cláusulas es la siguiente:

$A_{c_1} = 1 - (1 - z_1)(1 - z_2)z_3$
$A_{c_2} = 1 - (1 - z_2)z_3(1 - z_4)$
$A_{c_3} = 1 - (1 - z_1)(1 - z_3)(1 - z_4)$
$A_{c_4} = 1 - z_1 z_2(1 - z_4)$

De acuerdo a lo anterior la fórmula aritmética A_f es:

$$A_f = [1-(1-z_1)(1-z_2)z_3][1-(1-z_2)z_3(1-z_4)][1-(1-z_1)(1-z_3)(1-z_4)][1-z_1z_2(1-z_4)]$$

Al realizar todos los productos, es claro que A_f es un polinomio en las variables $z_1, z_2, z_3\, y\, z_4$. El grado de este polinomio es a lo mas $3m$ puesto que si cada cláusula tiene tres variables booleanas, entonces el grado después de aritmetizar cada cláusula es 3 y dado que m representa el número total de cláusulas de la fórmula booleana, se tiene que el grado del polinomio es en total $3m$.

Como se ve en el ejemplo, al aritmetizar una fórmula booleana f se obtiene el producto de la aritmetización de cada cláusula, por tanto la fórmula f es satisfactible si, y sólo si, la fórmula aritmética A_f es distinta de cero para todas las asignaciones de las variables enteras. Esto se debe a que si f es no satisfactible, entonces existe al menos una cláusula c_i que no se satisface para cada asignación de verdad T.

PROPOSICIÓN 2.2. *Sea f una fórmula booleana en forma normal conjuntiva, con tres variables por cláusula. f no es satisfactible si, y sólo si $A_f = 0$ para todas las asignaciones 0-1 de las variables enteras.*

Demostración. Sea

$$f = C_1 \wedge ... \wedge C_i \wedge ... \wedge C_m$$

Se tiene que la aritmetización de f, es decir, el polinomio A_f, es igual al producto de todas las cláusulas aritmetizadas, esto es:

$$A_f = A_{c_1}...A_{c_i}...A_{c_m}$$

Sea $A_f \neq 0$ para alguna asignación de las variables enteras. Luego $A_f \neq 0$ si, y sólo si $A_{c_i} \neq 0$ para todo $i = 1, ..., m$; además, se tiene que la aritmetización de cada cláusula es $A_{c_i} = 1 - A_{x_{i1}}A_{x_{i2}}A_{x_{i3}}$. Por lo tanto, $A_{c_i} \neq 0$ si, y sólo si $A_{x_{i1}}A_{x_{i2}}A_{x_{i3}} \neq 1$, donde $A_{x_{i1}}, A_{x_{i2}}$, y $A_{x_{i3}}$ son las aritmetizaciones de las variables booleanas x_{i1}, x_{i2} y x_{i3} respectivamente, de la cláusula c_i.

Por otro lado, se tiene que a las variables enteras z_i sólo se les asigna valores de 0 o 1, de ahí que $A_{x_{i1}}A_{x_{i2}}A_{x_{i3}} \neq 1$ si, y sólo si $A_{x_{i1}}A_{x_{i2}}A_{x_{i3}} = 0$, y esta última ecuación se tiene si, y sólo si, existe $A_{x_{ij}} = 0$ para algún $j = 1, 2, 3$.

Por la aritmetización se tiene que $A_{x_{ij}} = 1 - z_{ij}$ o $A_{x_{ij}} = z_{ij}$, dependiendo si x_{ij} es la variable booleana o es la negación de la variable booleana.

- $A_{x_{ij}} = 1 - z_{ij}$ si, y sólo si $z_{ij} = 1$ y esto es si, y sólo si x_{ij} es verdadero y por tanto, c_i es verdadera para todo $i = 1, ..., m$, lo que es equivalente a que f es satisfactible.
- $A_{x_{ij}} = z_{ij}$ si, y sólo si $z_{ij} = 0$ y además se tiene que $z_{ij} = 0$ si, y solo si x_{ij} es falsa, por tanto $\sim x_{ij}$ es verdadera y con ello la cláusula c_i es verdadera para todo $i = 1, ..., m$. Esto es equivalente a que f sea satisfactible.

En conclusión, se tiene que f es satisfactible si, y sólo si existe una asignación de 0-1 a las variables enteras tal que $A_f \neq 0$. De acuerdo a lo anterior, se tiene que f no es satisfactible si, y sólo si $A_f = 0$ para todas las asignaciones de 0-1 de las variables enteras. ⧫

Para realizar el total de asignaciones de las variables enteras en el polinomio A_f se utiliza la siguiente expresión:

$$\sum_{z_1=0}^{1}\sum_{z_2=0}^{1}...\sum_{z_n=0}^{1} A_f(z_1,...,z_n)$$

Como $A_f(z_1,...,z_n) = A_f = A_{c_1}...A_{c_m}$ y $A_{c_i} = 0$ o $A_{c_i} = 1$, entonces $A_f(z_1,...,z_n) \leq 1$

luego,

$$\sum_{z_1=0}^{1}\sum_{z_2=0}^{1}...\sum_{z_n=0}^{1} A_f(z_1,...,z_n) \leq 2^n$$

También, se define para cada i, $0 \leq i \leq n$, el conjunto de polinomios $A_f^i(z_1,...,z_i)$ de la siguiente forma:

$$A_f^i(z_1,...,z_i) = \sum_{z_i+1=0}^{1}...\sum_{z_n=0}^{1} A_f(z_1,...,z_n) \tag{2.1}$$

Por lo anterior se tiene que $A_f^n = A_f$. Además, $A_f^0 = 0$ si, y sólo si f no es satisfactible, y para cada i:

$$A_f^{i-1} = A_f^i(z_i = 0) + A_f^i(z_i = 1). \tag{2.2}$$

La ecuación (2.2) se obtiene reemplazando $z_i = 0$ y $z_i = 1$ en la ecuación (2.1) de la siguiente manera:

$$A_f^i(z_1,...,z_i) = \sum_{z_i+1=0}^{1}...\sum_{z_n=0}^{1} A_f(z_1,...,z_n)$$

$$A_f^i(z_1,...,z_i) = A_f(z_1,...,z_i,0,...,0) + ... + A_f(z_1,...,z_i,1,...,1)$$

reemplazando al lado derecho de la ecuación $z_i = 0$ y $z_i = 1$, se tiene que:

$$A_f(z_1,...,z_{i-1},0,...,0) + ... + A_f(z_1,...,z_{i-1},1,...,1)$$

esta última expresión es

$$A_f^{i-1}(z_1,...,z_{i-1})$$

En consecuencia, se ha probado que la ecuación (2.2) es correcta.

A continuación se presentará una aplicación de la aritmetización de fórmulas booleanas. En este resultado se establece la relación entre la clase co-NP y la clase de problemas

que admiten un sistema de pruebas interactivas, esto es, los problemas que pertenecen a la clase IP. Antes de presentar el teorema, es conveniente demostrar que la clase IP es cerrada bajo reducibilidad polinomial.

PROPOSICIÓN 2.3. *Sean L_1 y L_2 dos lenguajes. Si $L_2 \in IP$ y $L_1 \leq_p L_2$, entonces $L_1 \in IP$.*

Demostración. Sean L_1 y L_2 dos lenguajes tales que $L_1 \leq_p L_2$. Si $L_2 \in IP$ se debe mostrar que $L_1 \in IP$.

Sean Σ_1^* y Σ_2^* los alfabetos de L_1 y L_2 respectivamente. Por hipótesis, $L_1 \leq_p L_2$, es decir, existe una reducción polinomial $f : \Sigma_1^* \to \Sigma_2^*$ de L_1 en L_2.

Sea MTf la máquina de Turing tiempo polinomial que computa f. Como $L_2 \in IP$, existe un probador P_2^* tal que (P_2^*, V) decide L_2 para V un verificador dado. Se debe probar que $L_1 \in IP$, esto es, que existe un probador P_1^* tal que el sistema (P_1^*, V) decide L_1.

El sistema (P_1^*, V) puede ser construido así: para cada entrada $x \in \Sigma_1^*$, (P_1^*, V) utiliza MTf para transformar x en $f(x) \in \Sigma_2^*$ y luego usa (P_2^*, V) para determinar si $f(x) \in L_2$.

- Si $f(x) \in L_2$, entonces $\alpha(P_2^*, V, f(x)) > \frac{2}{3}$ y (P_1^*, V) acepta. Así, $\alpha(P_1^*, V, x) > \frac{2}{3}$.
- Si $f(x) \notin L_2$, entonces $\beta(P_2^*, V, f(x)) > \frac{2}{3}$ y (P_1^*, V) rechaza. Así, $\beta(P_1^*, V, x) > \frac{2}{3}$.

Puesto que $x \in L_1$ si, y sólo si, $f(x) \in L_2$, se tiene que (P_1^*, V) decide L_1. Además, (P_1^*, V) es un sistema de pruebas interactivas, luego $L_1 \in IP$. ⧫

TEOREMA 2.7. *[PC94] Todos los lenguajes que pertenecen a la clase co-NP admiten un sistema de pruebas interactivas, es decir,*

$$co - NP \subseteq IP$$

Demostración. De la proposición anterior se tiene que la clase IP es cerrada bajo reducibilidad polinomial, por lo tanto se demostrará que un lenguaje en co-NP-completo admite un sistema de pruebas interactivas y con ello se completará la prueba.
Para iniciar la demostración, se tomará el complemento de $3-SAT$, el cual se define así:

ENTRADA: Una fórmula booleana f en FNC de m cláusulas sobre n variables (a lo mas 3 variables por cláusula).

PREGUNTA: ¿Para toda asignación de verdad T, la fórmula f es no satisfactible?

Sean f una fórmula booleana y A_f la correspondiente fórmula aritmética de f. Para mostrar que el complemento de $3-SAT$ pertenece a la clase IP, se construirá un sistema de pruebas para f como se indica enseguida:

El protocolo se inicia cuando el verificador pregunta al probador un número primo p de tamaño considerable. V comprueba polinomialmente, que p es primo. En el i-ésimo paso del protocolo, el verificador V elige aleatoriamente un número r_i tal que $r_i \in \{0, 1, ..., p-1\}$ y calcula b_i; este último valor se transmite al probador y éste retorna los coeficientes de un polinomio apropiado.

Al iniciar el i-ésimo paso, para $i \geq 1$, los números $r_1, ..., r-i-1$ han sido seleccionados y los valores $b_0, b_1, ..., b_{i-1}$ calculados. Posteriormente, el probador P determina los coeficientes de un polinomio g' univariado. El verificador conoce que el polinomio adecuado es

$$g_i(x) = A_f^i(r_1, ..., r_{i-1}, x),$$

por ello, para comprobar, realiza el test de consistencia $b_{i-1} = g_i'(0) + g_i'(1)$, obtenido a partir de la ecuación (2.2). Si esta condición no se dá entonces el verificador rechaza la entrada, sino, genera un número aleatorio r_i y calcula el valor $b_i = g_i'(r_i)$. En el n-ésimo paso del protocolo, el verificador realiza el último test

$$b_n = A_f(r_1, r_2, ..., r_n).$$

El verificador acepta la entrada si todos los pasos son consistentes con el test.

El algoritmo para el **verificador** del sistema de pruebas interactivas para el complemento de 3-SAT es el siguiente.

Inicio {Entrada: f}
construya A_f de f;
envíe {pregunte un número primo};
reciba p;
Si p no es primo o $p < 2^{|f|}$ entonces
rechace;
$b_0 \leftarrow 0$;
Para $i \leftarrow 1$ hasta n hacer
inicio
envié b_{i-1};
reciba los coeficientes de g';
si $b_{i-1} \neq g'_i(0) + g'_i(1)$ entonces
rechace;
$r_i \leftarrow$ random $(0, p-1)$;
$b_i \leftarrow g'_i(r_i)$;
fin;
Si $b_n = A_f(r_1, ..., r_n)$ entonces
acepte
sino
rechace;
Fin.

Para este sistema de pruebas interactivas la capacidad del probador es evidente desde la elección de un primo adecuado; por tanto, un posible algoritmo del **probador** para el complemento de 3-SAT sería el siguiente.

Inicio {Entrada: f }
reciba {pregunta};

construya un número primo $p > 2^{|f|}$;

envié p;

reciba b_{i-1};

construya g' un polinomio {tamaño adecuado};

envié los coeficientes de g';

Fin.

Por el postulado de Bertrand [JGR99], se tiene que para cada $n > 0$ siempre existe un número primo p tal que

$$n < p < 2n.$$

Tomando $n = 2^i$ para cada $i > 0$, se tiene que existe un primo p, tal que

$$2^i < p < 2^{i+1},$$

de está manera se garantiza la existencia del número primo que elige el probador. Por otro lado se tiene que para verificar si un número dado es primo, existe un algoritmo que se ejecuta en tiempo polinomial. Ahora se probará que este sistema es en realidad un sistema de pruebas interactivas.

Si de entrada se tiene que f no es satisfactible, entonces el probador siempre responde correctamente puesto que el polinomio g' lo toma de la siguiente manera:

$$g'(x) = A_f^i(r_1, ..., r_{i-1}, x)$$

Así, el sistema verificador-probador acepta la entrada con una probabilidad de 1.

Si f es satisfactible de entrada y el probador quiere engañar al verificador, se probará que la probabilidad de error del sistema es de a lo mas $(1 - 3m/p)^i$ para cada i.

Sean g y g' dos polinomios cuyo grado es a lo más $3m$. Por la aritmetización, g denota el polinomio correcto para el sistema y g' denota el polinomio que utiliza el probador para engañar al verificador.

Para $i = 1$, se tiene que g_1 y g'_1 son iguales a cero, por tanto el verificador se equivoca y continua.

Supóngase que el sistema se ha ejecutado $i-1$ veces y que g_{i-1} y g'_{i-1} son distintos con una probabilidad de al menos $(1-3m/p)^{i-1}$. Como son g y g' dos polinomios de una variable, de grado a lo mas $3m$, entonces ellos pueden coincidir en a lo más $3m$ puntos. De ahí que, como los polinomios se evalúan en los r tales que $r \in \{0, ..., p-1\}$, entonces $g_{i-1}(r) \neq g'_{i-1}(r)$ por lo menos en $p - 3m$ puntos. Luego, la probabilidad de que los dos polinomios sean iguales es $3m/p$, por lo tanto la probabilidad de que los dos polinomios sean distintos (el complemento) es $1 - 3m/p$. En conclusión, la probabilidad de que g_i y g'_i sean diferentes después de i pasos es por lo menos

$$(1-3m/p)^{i-1}(1-3m/p) = (1-3m/p)^i$$

Se tiene que el sistema se ejecuta en n pasos, entonces la probabilidad de que los dos polinomios sean diferentes es por lo menos $(1-3m/p)^n$. Además, como $p > 2^{|f|}$ donde $|f| = 3m$, se tiene que

$$\begin{aligned} p &> 2^{3m} \\ \frac{1}{p} &< \frac{1}{2^{3m}} \\ 1 - \frac{1}{p} &> 1 - \frac{1}{2^{3m}} \\ \left(1 - \frac{1}{p}\right)^n &> \left(1 - \frac{1}{2^{3m}}\right)^n \end{aligned}$$

Luego, la probabilidad de que los dos polinomios sean diferentes y el sistema acepte es muy pequeña, puesto que

$$\begin{aligned} 3(2^{3m} - 1) &> 2^{3m} \\ \frac{2^{3m} - 1}{2^{3m}} &> \frac{1}{3}, \end{aligned}$$

por lo tanto,

$$1 - \frac{1}{2^{3m}} > \frac{1}{3} \tag{2.3}$$

$$\left(1 - \frac{1}{p}\right)^n > \left(\frac{1}{3}\right)^n \tag{2.4}$$

En conclusión, se ha probado que un problema que pertenece a la clas co-NP-completo admite un sistema de pruebas interactivas, es decir, pertenece a la clase IP, por tanto $co - NP - completo \subseteq IP$ y en consecuencia, $co - NP \subseteq IP$. ⧫

En los siguientes resultados se determina la capacidad de los sistemas de pruebas interactivas, debido a que se establece la relación entre la clase IP y una de las clases de complejidad más importantes conocidas hasta el momento, denominada $PSPACE$ definida en 1.10; para realizar la demostración se hace necesario presentar algunas definiciones previas y un resultado que de ellas se deriva, así como también un ejemplo de aplicación de estos nuevos conceptos.

TEOREMA 2.8. *[Uzu98] Sea L un lenguaje que admite un sistema de pruebas interactivas, entonces L es decidido por un algoritmo determinístico que utiliza a lo mas almacenamiento polinomial en la computación para cada entrada x de L. Es decir,*

$$IP \subseteq PSPACE.$$

Demostración. Sea L un lenguaje que admite un sistema de pruebas interactivas, se debe mostrar que L es decidido por un algoritmo determinístico que utiliza a lo mas almacenamiento de longitud polinomial para cualquier entrada de L.

De acuerdo a lo anterior, sea (P, V) el sistema de pruebas que decide L y sea x una entrada. Se define una máquina M determinística que recibe como entrada x y simula el intercambio de mensajes entre el probador P y el verificador V así como la respuesta dada por el verificador. Se tiene que M decide L utilizando espacio polinomial respecto al tamaño de x, puesto que por definición de pruebas interactivas, la longitud y el número

de mensajes está acotado polinomialmente respecto a $|x|$. También, el verificador es una máquina de Turing tiempo polinomial.

Como $L \in IP$ y existe un verificador V para L, entonces para cualquier x se puede establecer el árbol de todas las posibilidades del texto. Los nodos de este árbol corresponden a los mensajes enviados entre el verificador y probador. Si $x \in L$, entonces el árbol del texto contiene más de $2/3$ del total de las computaciones son de aceptación, de ahí que la máquina M toma el subárbol correspondiente como criterio de aceptación, de la misma forma si $x \notin L$, la máquina M toma el subárbol correspondiente a las computaciones de rechazo para rechazar la entrada. ♦

Para demostrar la otra inclusión, es decir $PSPACE \subset IP$, se hace necesario presentar las siguientes notaciones.

Definición 2.18. *Sea p un polinomio multivariado. Para cualquier variable x se define:*

1. $AND_x(p) = p(x = 0)p(x = 1)$.
2. $OR_x(p) = p(x = 0) + p(x = 1) - p(x = 0)p(x = 1)$.
3. $RED_x(p) = p(x = 0) + [p(x = 1) - p(x = 0)]x$

$RED_x(p)$ coincide con el polinomio p en las asignaciones de ceros y unos de las variables, debido a que

$$\begin{aligned} RED_x(p) &= p(x = 0) + [p(x = 1) - p(x = 0)]x \\ &= p(x = 0) + xp(x = 1) - xp(x = 0) \\ &= (1 - x)p(x = 0) + xp(x = 1) \end{aligned}$$

Por tanto, si $x = 0$ entonces $RED_x(p) = p(x = 0)$, así mismo si $x = 1$ entonces $RED_x(p) = p(x = 1)$.

Recuérdese que de la técnica de aritmetización se tiene que, a partir de una fórmula booleana se obtiene un polinomio multivariado, y es a este polinomio que se le aplica la definición anterior, como en el siguiente ejemplo.

Ejemplo 11.

Sea f una fórmula booleana en forma normal conjuntiva.

$$f = (x_1 \vee x_2 \vee x_3) \wedge (\sim x_1 \vee x_3 \vee \sim x_4) \wedge (\sim x_2 \vee \sim x_3 \vee \sim x_4)$$

Al aritmetizar f se obtiene el siguiente polinomio multivariado

$$A_f(z_1, z_2, z_3, z_4) = (1 - (1 - z_1)(1 - z_2)(1 - z_3))(1 - z_1(1 - z_3)z_4)(1 - z_2 z_3 z_4))$$

Al aplicar la definición 2.18, tomando $x = z_3$ se tienen los polinomios

$AND_{z_3}(A_f) = (1 - (1 - z_1)(1 - z_2))(1 - z_1 z_4)(1 - z_2 z_4)$

$OR_{z_3}(A_f) = (1-(1-z_1)(1-z_2))(1-z_1 z_4)+(1-z_2 z_4)-(1-(1-z_1)(1-z_2))(1-z_1 z_4)(1-z_2 z_4)$

$RED_{z_3}(A_f) = (1-(1-z_1)(1-z_2))(1-z_1 z_4)+[(1-z_2 z_4)-(1-(1-z_1)(1-z_2))(1-z_1 z_4)]z_3$

En el siguiente resultado se prueba que la clase IP es de alguna forma, o en algún sentido cerrada bajo las operaciones AND, OR y RED. Para ello es necesario conocer el problema denominado *p-chequeo* que se define así.

ENTRADA: Dado un polinomio multivariado $p(x_1, ..., x_n)$ sobre n variables, una n-tupla $a_1, ..., a_n$ y un número b.

PREGUNTA: ¿Es cierto que $p(a_1, ..., a_n) = b$?

LEMA 2.1. *[PC94] Si el problema p-chequeo pertenece a la clase IP, entonces $AND_x(p)$-chequeo, $OR_x(p)$-chequeo y $RED_x(p)$-chequeo también pertenecen a la clase IP.*

Demostración. Se mostrará únicamente que $AND_x(p)$-chequeo está en la clase IP, puesto que las otras dos pruebas se realizan de manera análoga.

Como el polinomio multivariado p está dado, entonces la entrada para el sistema de pruebas interactivas que se va a construir es $a_1, ..., a_n$ y b. Este sistema es de forma similar al que se utilizó en el teorema 2.7, por lo tanto el probador lo primero que hace es elegir un número primo, con la misma intención del sistema mencionado anteriormente. Posteriormente el probador envía al verificador los coeficientes de un polinomio univariado $S^{'}(x)$, que debe ser correctamente calculado como $S(x) = p(x, a_2, ..., a_n)$. En este caso el test de consistencia que realiza el verificador es

$$S^{'}(0)S^{'}(1) = c_{i-1}$$

Si esta igualdad no se cumple, entonces el verificador rechaza la entrada. Los algoritmos para este sistema son:

El algoritmo del **verificador**
Inicio{Entrada: $a_1, ..., a_n$; b}
envié {Pregunte un primo};
reciba q;
si (q no es primo) **o** $q < n$ **entonces**
rechace
sino
$c_0 \leftarrow b$;
$i \leftarrow 1$;
Mientras $(i < n)$ **hacer**
inicio
envié c_{i-1};
reciba los coeficientes de $S_i^{'}$;
si $c_{i-1} \neq S_i^{'}(0)S_i^{'}(1)$ **entonces**
$i \leftarrow n + 1$
sino
$r_i \leftarrow random(0, q - 1)$;
$c_i \leftarrow S_i^{'}(r_i)$;
$inc(i)$;

fin;

si $(i = n)$ **entonces**

acepte

sino

rechace;

Fin.

Para completar el sistema, se presenta el algoritmo para el **probador.**

Inicio{Entrada: $a_1, ..., a_n$; b}

envié un primo $q > n$;

reciba c_{i-1};

calcule $S_i'(x) = f_i(a_1, ..., a_{i-1}, x, a_{i+1}, ..., a_n)$;

envié los coeficientes de S_i';

Fin.

Ahora se probará que el sistema descrito es un sistema de pruebas interactivas.

- Supóngase primero que la entrada es correcta, es decir que

$$p(a_1, ..., a_n) = b,$$

de ahí que el polinomio que elige el probador es $f = p$, además como se tiene que el problema p-chequeo pertenece a la clase IP, entonces

$$c_{i-1} = S_i'(0)S_i'(1) \tag{2.5}$$

con una probabilidad de 1; puesto que determinar (2.5) es resolver el problema p-chequeo con $n - 1$ variables y la entrada correcta.

- Supóngase ahora que la entrada no es correcta y además que el problema p-chequeo tiene un error de probabilidad igual ϵ. De acuerdo a esto el polinomio $S'(x)$ que elige el probador es distinto al polinomio $S(x) = p(x, a_2, ..., a_n)$, pero coinciden en

el punto r_i para todo $i = 1, ..., n$, esto es, $S^{'}(r_i) = S(r_i)$.

Como la entrada es incorrecta, es decir, $p(a_1, ..., a_n) \neq b$ entonces el sistema acepta con una probabilidad ϵ debido a que este es el error para decidir el problema p-chequeo; como además los polinomios S y $S^{'}$ son diferentes pero satisfacen el test de consistencia se realiza el mismo análisis del teorema anterior, esto es, dos polinomios diferentes que coinciden en n puntos (ecuación (2.3)), de ahí que el sistema acepta una entrada incorrecta con una probabilidad de a lo mas $\epsilon + \left(1 - \frac{1}{q}\right)^n$, donde q es el número primo que elige el probador.

Por lo tanto se tiene que el sistema descrito es realmente un sistema de pruebas interactivo y en consecuencia el problema $AND_x(p)$-chequeo pertenece a la clase IP.

De manera similar se realiza la demostración para los otros dos casos. Para el problema $OR_x(p)$-chequeo el test de consistencia es

$$S^{'}(0) + S^{'}(1) - S^{'}(0)S^{'}(1) = c_{i-1}.$$

Mientras que para el problema $RED_x(p)$-chequeo el test de consistencia es

$$S^{'}(0) + [S^{'}(1) - S^{'}(0)]a_i = c_{i-1}.$$

En conclusión se tiene que, si el problema p-chequeo pertenece a la clase IP, entonces los problemas $AND_x(p)$-chequeo, $OR_x(p)$-chequeo y $RED_x(p)$-chequeo también pertenecen a la clase IP. ♦

Antes de presentar el siguiente resultado, se hace necesario definir la técnica de aritmetización para fórmulas booleanas cuantificadas, esto es:

- Cada cuantificador existencial ($\exists$) se reemplaza por una sumatoria ($\sum$).
- Cada cuantificador universal ($\forall$) se reemplaza por un productorio ($\prod$).
- Para el resto de la fórmula se le aplica la técnica descrita anteriormente.

Ejemplo 12.

Sea F una fórmula booleana cuantificada,

$$F = (\exists x_1)(\forall x_2)(\exists x_3)(\forall x_4)[(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \sim x_3 \vee \sim x_4)]$$

Al aplicar la técnica de aritmetización a F, las variables enteras se evalúan en cero y uno, por ello se obtiene como resultado A_F

$$A_F = \sum_{z_1=0}^{1} \prod_{z_2=0}^{1} \sum_{z_3=0}^{1} \prod_{z_4=0}^{1} [1-(1-z_1)(1-z_2)(1-z_3)][1-(1-z_2)z_3z_4]$$

Al realizar las evaluaciones se obtiene

$$A_F = \sum_{z_1=0}^{1} \prod_{z_2=0}^{1} \sum_{z_3=0}^{1} [1-(1-z_1)(1-z_2)(1-z_3)]^2[1-(1-z_2)z_3]$$

$$A_F = \sum_{z_1=0}^{1} \prod_{z_2=0}^{1} [1-(1-z_1)(1-z_2)]^2 + [1-(1-z_2)]$$

$$A_F = \sum_{z_1=0}^{1} [1-(1-z_1)]^2 = 2$$

Recuérdese que F es verdadera si, y sólo si, el valor obtenido al aplicar la técnica de la aritmetización es diferente de cero.

TEOREMA 2.9. *[LF00] Sea L un lenguaje que está en la clase $PSPACE$, entonces L admite un sistema de pruebas interactivas, esto es*

$$PSPACE \subseteq IP$$

Demostración. La prueba consiste en derivar un sistema de pruebas interactivas para el problema QBF,[13]el cual como ya se demostró en el capítulo anterior pertenece a la clase $PSPACE$-completo.

El primer paso es aritmetizar la fórmula booleana cuantificada, es posible que en este proceso los polinomios obtenidos sean de grado exponencial, por ello se utiliza la técnica de reducción de grado en todas las variables, esto es aplicar $RED_{x_i}(A_f)$ para $i = 1, 2, ..., n$, donde A_f representa la aritmetización de la fórmula booleana f en FNC.

[13]Este problema se definió en el ejemplo 5 de la sección 1.3

Sea F una fórmula booleana cuantificada,

$$F = (Q_1 x_1)(Q_2 x_2)...(Q_n x_n) f(x_1, x_2, ..., x_n)$$

donde f es una fórmula booleana en forma normal conjuntiva sobre n variables $x_1, x_2, ..., x_n$ y Q_i con $1 \leq i \leq n$, denota los cuantificadores.

El siguiente algoritmo decide si la fórmula F es verdadera o falsa, es decir, mediante este algoritmo es posible decidir el problema QBF, la secuencia de pasos descritos utilizan pruebas interactivas descritos anteriormente.

Inicio { Entrada: F }
    Obtenga A_f;
    $g \leftarrow A_f$;
    para $i \leftarrow 1$ **hasta** n **hacer**
        $g \leftarrow RED_{x_i}(g)$;
        si $Q_{n-i+1} = \forall$ **entonces**
            $g \leftarrow AND_{x_{n-i+1}}(g)$;
            el sistema rechaza si g no pasa el test de consistencia
        sino
            $g \leftarrow OR_{x_{n-i+1}}(g)$;
            el sistema rechaza si g no pasa el test de consistencia;
    fin;
    acepte;
Fin.

Puesto que las operaciones AND y OR tienen su propio sistema de pruebas interactivas, entonces el algoritmo descrito anteriormente rechaza cuando el polinomio g no satisface alguno de los test de consistencia.

Después de los n pasos se obtiene una constante cuyo valor depende del valor de verdad de la fórmula booleana cuantificada, esto es, cero en el caso de que F sea falsa y distinto

de cero cuando F sea verdadera. Por otro lado, los polinomios obtenidos no son de grado mayor que el grado de A_f. Por lo tanto, el problema de decidir si F es verdadera se reduce a comprobar si $A_f(a_1, ..., a_n) = b$ para valores dados $a_1, ..., a_n$ y b. Esto último lo puede realizar el verificador en tiempo polinomial. ♦

Corolario 2.1. $IP = PSPACE$

Demostración. De los teoremas 2.8 y 2.9. ♦

2.4. COMPROBACIÓN PROBABILÍSTICA DE PRUEBAS

Otro modelo de sistemas de pruebas interactivas, es aquel donde el probador se considera como un *oráculo*[14]. En este caso el verificador tiene acceso aleatorio a cualquier parte de la prueba y se tendrán en cuenta principalmente dos casos, el primero de ellos es realizar intercambios en el orden de la prueba sin que esta se afecte y el segundo caso es la cantidad de preguntas que se deben realizar para comprobar que es correcta.

Definición 2.19. *Sea L un lenguaje. L admite una prueba de comprobación probabilística, si existe PT una BPP-máquina con oráculo tal que:*

1. *PT sólo se le permite generar bits aleatorios, es decir, la función* ***random***[15] *de PT sólo puede aceptar valores binarios.*

2. *Para cada $x \in L$, existe un oráculo X_x tal que PT^{X_x} acepta x.*

3. *Para cada $x \notin L$ y para cada oráculo X, PT^X rechaza x.*

[14]La definición de esta clase de máquina está en el capitulo anterior. (1.9)
[15]La definición de esta función se encuentra en [Ast04].

De acuerdo a lo anterior el oráculo X_x puede ser visto como un lenguaje que depende de la entrada x. Sin embargo, esto puede suponerse finito, puesto que dado un x, el número de posibles preguntas realizadas por PT es acotado por una función exponencial con respecto a la longitud de x.

Definición 2.20. *La clase $PCP(f, g)$ es el conjunto de todos los lenguajes que admiten una prueba de comprobación probabilística, tal que la correspondiente máquina PT genera a lo mas, $O[f(n)]$ de bits random (aleatorios) y lo mas $O[g(n)]$ preguntas.*

En este modelo la verificación se realiza de una manera aleatoria y no interactiva, cuando la prueba está dada, el verificador elige partes de esta y realiza algunas preguntas, esto dificulta que la comprobación sea eficiente pues no se verifica la prueba completa.

Uno de los principales resultados conocidos hasta el momento en teoría de la aproximación, es aquel que caracteriza de forma diferente la clase NP. Debido a que para su verificación sólo es necesario un número logarítmico de bits aleatorios y un número constate de preguntas.[16]

2.4.1. La Clase PCP y Algoritmos de Aproximación

El concepto de completitud en problemas de optimización no ha producido resultados significativos principalmente en problemas de decisión. Sin embargo, para probar la *no-aproximalidad* de un problema de optimización se hace uso de una técnica llamada la *técnica gap*, la cual se describe a continuación.

Sea L un lenguaje NP-completo que puede ser reducido a un problema A de NPO-maximización tal que para cualquier entrada x se tiene que

- *Si $x \in L$ entonces el $OPT_A[f(x)] \geq g(x)$.*

[16]Esto es lo que constituye uno de los teoremas más importantes de la teoría de complejidad, conocido como el teorema PCP [Tre00]

- *Si* $x \notin L$ *entonces* $OPT_A[f(c)] < g(x)(1-c)$.

Donde f, g *son dos funciones computables tiempo polinomial y* c *es una constante llamada* ***gap (hueco)***.

De lo anterior se tiene que, si el problema A admite un algoritmo ϵ-aproximable para cualquier $\epsilon \leq c$, entonces este problema es resuelto para cualquier entrada eficientemente y por tanto el problema de decisión adyacente, que es NP-completo, también se resuelve de manera eficiente y con ello se tiene que $P = NP$.

La relación entre las pruebas de comprobación probabilísticas y los algoritmos de aproximización se describe por ejemplo, para un lenguaje NP-completo, una prueba de comprobación probabilística necesita un gap de tamaño considerable en la probabilidad de aceptación, tanto para entradas correctas como incorrectas. Estas pruebas pueden ser usadas para construir instancias de un problema de optimización y así demostrar la no aproximabilidad de este, a menos que $P = NP$. Esto se evidencia en el teorema que se presenta a continuación.

TEOREMA 2.10. *[PC94] Si* $NP \subseteq PCP(log, log)$*, entonces el problema de MÁXIMO CLIQUE no pertenece a la clase* APX*, a menos que* $P = NP$.

Demostración. La prueba del teorema consiste en derivar un grafo a partir de una fórmula booleana y determinar si la fórmula es satisfactible dependiendo si el grafo tiene un *Clique* de tamaño determinado. Por un resultado de [PC94], se tiene que si el problema de *Máx Clique* es aproximable, entonces existe un algoritmo 1/2-aproximado para el óptimo.

Se tiene que $SAT \in NP$ entonces $SAT \in PCP(f, g)$ donde $f(n) = log(n)$ y $g(n) = log(n)$ luego, existe PT una BPP-máquina que genera a lo mas $O(f(n))$ bits random y a lo mas $O(g(n))$ preguntas.

Sea h una fórmula booleana de tamaño n, se construirá un grafo G_h con a lo mas $2^{f(n)+g(n)}$ nodos de manera que si h es satisfactible, entonces el máx clique en G_h tiene un tamaño de $\frac{2}{3}2^{f(n)}$. Por otro lado, si h no es satisfactible entonces el máx clique tiene un tamaño menor que $\frac{1}{3}2^{f(n)}$.

El grafo G_h se construirá usando la máquina PT, para h la fórmula booleana dada.

Se denotará por q_i una pregunta realizada por la máquina PT y por a_i la correspondiente respuesta. Una secuencia de bits random se notaran como una cadena binaria r.

Una *transcripción* de PT sobre una entrada x es una tupla $\langle r, q_1, a_1, q_2, a_2, ..., q_l, a_l\rangle$ donde $|r| \leq k_1 f(|x|)$ y $l \leq k_2 g(|x|)$ para algunas constantes k_1 y k_2, tales que para cualquier j con $1 \leq j \leq l$, q_j es la j-ésima pregunta hecha por $PT(x)$ asumiendo que la respuesta a q_i es a_i para $1 \leq i \leq j$ y los bits random generados hasta este punto son consistentes con r. Una transcripción t es una *transcripción de aceptación* si PT con entrada x, r bits aleatorios y una secuencia de preguntas y respuestas $\langle q_1, a_1, ..., q_l, a_l\rangle$ acepta x. Por último se dice que dos transcripciones $t = \langle r, q_1, a_1, ..., q_l, a_l\rangle$ y $t' = \langle r', q_1', a_1', ..., q_l', a_l'\rangle$ son *consistentes* si, para cada i y j, si $q_i = q_j'$ entonces $a_i = a_j'$.

Los nodos del grafo G_h serán todas las transcripciones de aceptación. Para construir este conjunto de nodos, se enumera primero todas las posibles transcripciones, la cadena de bits random son a lo mas $2^{f(n)}$ puesto que como $|r| \leq k_1 f(n)$ y las todas las posibilidades de la cadena de bits son $2^{|r|}$. Por otro lado el total de combinaciones de preguntas, es igual a $2^{g(n)}$, debido a que se toma el total de subconjuntos de $g(n)$ elementos, puesto que $l \leq g(n)$. En conclusión el número total de trascripciones es $2^{f(n)+g(n)}$, debido a que son todas las posibles combinaciones de bits random y preguntas. Por otro lado se tiene que dos nodos son adyacentes si, y sólo si, las transcripciones son consistentes.

De acuerdo a lo anterior se tienen los siguientes dos casos:

1. Si h es satisfactible entonces (por definición 2.19) existe un oráculo X_h tal que la

razón entre el número de caminos de computación de aceptación de $PT^{X_h}(h)$ y el número total de caminos de computación es al menos 2/3.

Para este oráculo X_h, el número de trascripciones que son consistentes es al menos $2^{f(n)}$, (fijando la secuencia de preguntas) y por lo menos 2/3 de ellas son de aceptación, es decir, el número de trascripciones de aceptación son $\frac{2}{3}2^{f(n)}$ y todas ellas son consistentes, por lo tanto forman un subgrafo completo. En consecuencia el grafo G_h tiene un Clique de tamaño $\frac{2}{3}2^{f(n)}$.

2. Si h es no satisfactible entonces (por la definición 2.19) para todo oráculo X la razón entre el número de caminos de computación de aceptación de $PT^X(h)$ y el número total de caminos de computación de $PT^X(h)$ es a lo más 1/3.

   En este caso el número de trascripciones de aceptación consistentes es menor que $\frac{1}{3}2^{f(n)}$. Por lo tanto, el grafo G_h contiene un Clique de tamaño menor que $\frac{1}{3}2^{f(n)}$.

Sea T un algoritmo 1/2-aproximado para Máx Clique. Se corre el algoritmo T sobre el algoritmo G_h.

- Si T devuelve un Clique de tamaño mayor o igual que $\frac{1}{3}2^{f(n)}$, entonces $OPT(G_h) = \frac{2}{3}2^{f(n)}$ y con ello la fórmula h es satisfactible (por el contrarrecíproco de 2).
- Si T devuelve un Clique de tamaño menor que $\frac{1}{3}2^{f(n)}$, entonces $OPT(G_h) = \frac{1}{3}2^{f(n)}$ y con ello h no es satisfactible (por el contrarrecíproco de 1).

En conclusión, T sería un algoritmo que decide SAT en tiempo polinomial y por lo tanto $P = NP$. ⧫

La clase de complejidad NP se caracteriza de manera diferente, tal que contiene todos aquellos lenguajes L los cuales contienen una prueba,(probar si $x \in L$) que ser verificada probabilísticamente en tiempo polinomial usando un número *logarítmico* de bits random y número constante de preguntas.

TEOREMA 2.11. *[Tre00]* $NP = PCP(log(n), 1)$

3. CONCLUSIONES

- Se realizó un estudio detallado de los sistemas de pruebas interactivas y su relación con la teoría de la complejidad.

- Se presentó una aplicación de las máquinas de Turing probabilísticas por medio de los algoritmos probabilísticos en la solución de problemas.

- Se estableció la relación entre la clase de complejidad originada a partir de los sistemas de pruebas interactivas y las principales clases de complejidad.

- Se mostró la solución de problemas en teoría de grafos y teoría de números utilizando pruebas interactivas, además se realizó el seguimiento de los algoritmos mediante ejemplos prácticos.

- Se presentó la definición y el funcionamiento de otro tipo de máquina de Turing denominada *Oráculo*. Utilizada para realizar algunas demostraciones y en la definición de otro tipo de sistemas de pruebas llamado, *Comprobación Probabilística de Pruebas*.

- Se hizo una introducción al estudio del teorema PCP, mediante la teoría de la aproximabilidad y se mostró un resultado muy importante para su comprensión.

- Utilizando la comprobación probabilística de pruebas se describió que es posible caracterizar la clase NP de manera diferente.

BIBLIOGRAFÍA

[Ast04] ASTUDILLO ASTUDILLO, Mayerlin. Algoritmos Probabilísticos. Universidad del Cauca, facultad de ciencias naturales, exactas y de la educación. Popayán. 2004.

[Coo00] COOK, Stephen. the P versus NP Problem. University of Toronto, 2000.<http:// www.claymath.org/Millenium-Prize- Problems/P-vs- NP/-Objects/Official-Problem-Description.pdf>.

[CLR90] CORMEN, Thomas; LEISERSON, Charles y RIVEST, Ronald. Introduction to Algorithms. Nueva york. McGraw Hill.1990.

[Dan96] DANTSIN, Evgeny. A Logic-Style Version of Interactive Proof. Steklov Institute of Mathematics St. Petersburg, Russia. 1996 <http:// www.researchindex.com>.

[FH02] FORTNOW Lance, HOMER Steve. A short history of computational complexity. University of Boston, 2002. <http:// www.researchindex.com>.

[FS88] FORTNOW Lance, SIPSER Michael. Are There Interactive Protocols for Co-NP Languages? 1988 <http:// www.researchindex.com>.

[GJ79] GAREY, Michael R y JOHNSON, David S. Computers and intractibility: A guide to the theory of NP- completeness. New york: W.H. Freeman, 1979.

[Has98] HASTAD Johan. Clique is Hard to Approximate Within $n^{1-\epsilon}$. Royal Institute of Technology. Sweden 1998. <http:// www.researchindex.com>.

[HB03] HERRERA FLOREZ, Maritza; BOLAÑOS RIVERA, Yudy Marcela. Aproximabilidad en problemas NP-duros. Universidad del Cauca, facultad de ciencias naturales exactas y de la educación. Popayán. 2003.

[JGR99] JIMÉNEZ Rafael, GORDILLO Enrique, RUBIANO Gustavo. Teoría de numeros para principiantes. Bogotá. Universidad Nacional de Colombia facultad de ciencias. 1999.

[Jon69] JONES Burton. Teoría de los Números. Ed. F. Trillas S.A. México, 1969.

[LF00] LUND Carsten, FORTNOW Lance. Algebraic Methods for Interactive Proof Systems. University of Boston, 2000. <http:// www.researchindex.com>.

[Pap94] PAPADIMITRIOU, Christos H. Computational Complexity. Addison Wesley, publishing Company.1994.

[PC94] PIERRE BOVE, Daniel; CRESSCENZi, Pierluig.Introdution to the theory of complexity. Gran Bretaña, Prentice-hall international. 1994

[SS98] SANJEEV Arora, SAFRA Shmuel. Probabilistic Checking of Proofs: A New Characterization of NP. 1998. <http:// www.researchindex.com>.

[Tre00] TREVISAN Luca. Interactive and Probabilistic Proof-Checking. Columbia University. 2000. <http:// www.researchindex.com>.

[Uzu98] UZURIAGA, Lopéz Vivian Libeth. Comprobación Probabilística de Resultados de Programas para Problemas de Optimización en Teoría de Grafos. Universidad del Valle. Santiago de Cali. 1998.