Complejidad del testing adaptativo en escenarios de testing definidos extensionalmente

Trabajo de Fin de Grado del Doble Grado en Ingeniería Informática y Matemáticas

Alumno: David Rubio Ibáñez

Director: Ismael Rodríguez Laguna

20 de septiembre de 2019

Tabla de contenidos

Resumen

Introducimos el concepto de testing adaptativo definido extensionalmente con sus distintas variantes (hay determinismo o no, hay una única definición correcta de implementación o no) y demostramos cotas superiores de su complejidad para todos los casos (PSPACE) y para el caso determinista en el que solo hay una posible definición correcta de implementación (log-APX), así como cotas inferiores tanto para el caso no determinista (PSPACE-dureza) como para el caso determinista (log-APX-dureza). El hecho de que haya que hacer las definiciones extensionalmente quiere decir que este tipo de testing adaptativo es relativamente sencillo. Aún así obtenemos un nivel relativamente alto de complejidad, lo cual quiere decir que los tipos de testing adaptativos que generalicen a este tendrán también un nivel alto de complejidad.

Palabras clave: *testing adaptativo*, *complejidad computacional*, *PSPACE-completitud*, *log-APX-completitud*.

We introduce the concept of adaptive testing defined extensionally with different variants (determinism or not, there's only one correct implementation or not) and we prove upper bounds of its complexity for all the cases (PSPACE) and the variant with determinism and only one correct implementation (log-APX), as well as lower bounds to the non-deterministic variant (PSPACE-hardness) and for the deterministic variant (log-APX-hardness). Having extensional definitions means that this type of adaptive testing is relatively simple. Even so, we get a high complexity level, which means that the types of adaptive testing that generalize this one will also have a high complexity level.

Key words: *adaptive testing*, *computational complexity*, *PSPACE-completeness*, *log-APX-completeness*.

Introducción

En este trabajo estudiaremos la complejidad computacional del problema de testing adaptativo bajo ciertas condiciones.

En el ámbito de **testing** queremos determinar si la implementación de un sistema cumple con ciertas especificaciones a base de interactuar con ella. Además, consideraremos que no podemos observar el funcionamiento interno de la implementación, es decir, que es una caja negra, por lo que lo único que podemos aprender sobre la implementación son los outputs que produce cuando introducimos ciertos inputs.

Nosotros estudiaremos el **testing adaptativo**, en el que cada input introducido podrá depender de los outputs obtenidos previamente durante la aplicación de los inputs anteriores. En muchas circunstancias, esto puede acortar el número de interacciones necesarias para detectar errores.

Los resultados que vamos a obtener aquí tratarán sobre un tipo de testing adaptativo muy determinado. Concretamente, habrá una **cantidad finita de posibles comportamientos**, correctos o no, que la implementación podrá tener y cada uno de estos posibles comportamientos nos será dado **extensionalmente**.

Es decir, no se nos dará una regla o condición global que cumplan todas los sistemas posibles y otra regla o condición que cumplan todas las implementaciones correctas, si no que se nos listarán todas las posibles implementaciones y todas las implementaciones correctas una a una. Consideraremos que cada definición posible del comportamiento de una implementación para todos los inputs será dada por una función. Es más, la definición de cada una de estas funciones se nos dará extensionalmente: para cada input de un conjunto finito, nos dan el output que podrá devolver. Por tanto, no obtenemos el comportamiento a partir de una máquina abstracta (como una máquina finita de estados), si no como una simple lista finita de casos.

Esta forma de representación, finita y nada compacta, es obviamente limitada. No obstante, es un caso especialmente útil desde el punto de vista teórico para obtener complejidades mínimas, pues la complejidad computacional que hallemos para este caso será la mínima complejidad para cualquier otro escenario de testing que lo generalice. Por ejemplo, si permitimos que la definiciones sean definidas usando ciertas máquinas, y se pueden codificar nuestras definiciones finitas extensionales usando dichas máquinas, entonces la complejidad del nuevo escenario será, al menos, la del caso puramente extensional.

Aquí hay que tener en cuenta que las posibles definiciones de la implementación, tal y como nosotros las permitimos, no tienen en cuenta los inputs que se

les ha introducido anteriormente, como refleja el hecho de que hayamos escogido representarlas como funciones, donde obviamente cada aplicación de un input no afecta a posteriores aplicaciones. Esto puede ser una diferencia significativa con respecto a otros modelos de comportamiento como las máquinas de estados finitas o programas informáticos. Al testear modelos de cómputo que admiten estados internos tenemos que tener en cuenta que haciendo tests cambiamos el estado en el que estamos y, por tanto, también cambiamos el comportamiento que tendrá la máquina. Dado que volver al estado inicial puede no ser en absoluto trivial, nuestro modelo de testing parece sencillo en comparación, y sin embargo demostraremos para él un cierto nivel alto de complejidad mínima (PSPACE-dureza). Lo cual quiere decir que no es necesario ni tener distintos estados ni permitir infinitas formas posibles de interacción para alcanzar la PSPACE-dureza.

El caso extensional también tiene utilidad práctica tal y como es. Con frecuencia, lo testeadores de software definen su plan de pruebas a mano y de manera extensional (definen, test a test, los posibles resultados correctos e incorrectos que se pueden obtener). Si cada prueba es muy costosa en tiempo o dinero, entonce elegir qué tests aplicar de entre los posibles tests es una tarea esencial. Dicho problema encaja, tal cual, en nuestra definición extensional de testing adaptativo.

Aquí clasificaremos el testing adaptativo según las funciones (es decir, las posibles definiciones del comportamiento) puedan ser **deterministas o no**, y según pueda haber **una o más funciones correctas**. El caso determinista es un caso particular del no determinista, y el caso en el que solo hay una función correcta es un caso particular del caso en el que pueden haber varias.

Cuando definamos extensionalmente funciones no deterministas, en vez de indicar el único output para cierto input, lo que se indicará será el conjunto de posibles outputs. La versión determinista de este problema se puede ver como el caso particular de la versión no determinista en el que todos los conjuntos contienen un único elemento.

Las cotas superiores a la complejidad de un problema (como la pertenencia a cierta clase de complejidad) se pueden aplicar a los casos particulares de este, mientras que las cotas inferiores (como la la dureza o "hardness" con respecto a cierta clase) se pueden aplicar a las generalizaciones del problema en cuestión. Así, la cota superior a la complejidad que obtengamos del caso no determinista con múltiples funciones correctas nos servirá para todos los otros casos, y las cotas inferiores a la complejidad de los casos con una sola función correcta servirán para los casos con varias funciones correctas.

Definiciones

Clases de problemas

Los problemas computacionales se clasifican dependiendo del tiempo y espacio (memoria) máximos requeridos para resolver cualquier instancia. Cuando trabajemos con aproximaciones también se tendrá en cuenta la peor ratio de aproximación posible entre la solución obtenida y la correcta.

El tiempo, espacio y ratio de aproximación de un problema se miden asintoticamente con respecto al tamaño de los datos de entrada[1].

En este trabajo hablaremos de dos clases:

- **PSPACE**: es la clase de los problemas resolubles en espacio polinómico[1]. Contiene todos los problemas resolubles en tiempo polinómico de forma determinista (**P**) o no determinista (**NP**)[2] porque, intuitivamente, no "les da tiempo" a usar más memoria. También contiene muchos problemas en los que hay que comprobar si todas las ramas de un árbol de decisiones de profundidad polinómica cumplen cierta propiedad independiente del resto de ramas, pues no hace falta guardar más de una rama simultaneamente.
- **log-APX**: es la clase de problemas para los que se puede conseguir en tiempo polinómico una solución aproximada con ratio de aproximación logarítmica con respecto al tamaño de entrada[3].

Llamamos **reducción** de un problema A a otro B, y lo denotamos $A \preceq B$, a un procedimiento con el que transformar instancias del problema A a instancias del problema B que tengan el mismo resultado[4] o, en caso de estar usando aproximaciones, que mantengan cierta ratio de aproximación[5].

Intuitivamente, si $A \preceq B$ es porque B es tan o más difícil que A.

Hay distintos tipos de reducciones atendiendo a cuánto tiempo y espacio necesitan o a cómo de buena es la aproximación. Una reducción que use más recursos computacionales que la solución directa del problema que queremos reducir no nos sirve, así que dependiendo de la clase de problemas con la que estemos tratando buscaremos reducciones más o menos costosas.

Decimos que un problema es **duro** o **hard** con respecto a una clase si todos los problemas de esa clase se pueden reducir a él[6]. De nuevo, según la clase en cuestión permitiremos que la reducción consuma más o menos recursos computacionales. En el caso de nuestras demostraciones de dureza de PSPACE y log-APX podremos usar reducciones polinómicas en tiempo que, en el caso de log-APX, preserven el coste de todas las soluciones (no solo las óptimas, para así mantener la ratio de aproximación) [5][7].

Los problemas duros con respecto a una clase que pertenecen a dicha clase se llaman **completos** o **complete** con respecto a esa clase[6].

Aquí lidiaremos con **problemas de decisión** que consisten en determinar si un elemento pertenece o no a cierto conjunto[7] y con **problemas de optimización** en los que hay que encontrar o el elemento de cierto conjunto con el mínimo coste o el mínimo coste para el que existe un elemento en dicho conjunto[2]. De los problemas de optimización se puede hacer una versión de decisión que determinen si la solución buscada tiene un coste menor que un número dado[2].

Testing adaptativo

Dados dos conjuntos finitos no vacíos I y O que llamaremos conjuntos de inputs y outputs respectivamente diremos que un conjunto C de funciones de I a O no deterministas es un formalismo computacional de I y O[8]:

$$C \subseteq \{f : I \to \mathcal{P}(O) \setminus \{\emptyset\}\}$$

La finitud de I y O implica la de C. Cada $f \in C$ representa el posible comportamiento de la caja negra que vamos a testear.

Si toda $f \in C$ es determinista ($\forall i \in I \quad f(i)$ unipuntual) diremos que C es un formalismo computacional determinista, que será equivalente a definirlo como:

$$C \subseteq \{f : I \to O\}$$

Si C es un formalismo computacional, llamaremos especificación a un subconjunto suyo $E \subseteq C$ y diremos que las funciones incluidas en la especificación $f \in E$ son "correctas" y el resto de funciones de C no.

En el problema de decisión de testing adaptativo intentaremos averiguar si, dados un formalismo computacional C y una especificación $E \subseteq C$, se puede determinar con hasta k tests si cierta función, de la que solo conocemos los resultados de los tests, es "correcta" o no. Cada test consiste en proporcionar un input $i \in I$ y recibir un output $o \in f(i) \subseteq O$. Al ser testing adaptativo los inputs se pueden decidir en función de los resultados obtenidos en los test anteriores.

Formalmente, dados dos conjuntos finitos no vacíos I y O definimos el pro-

blema de decisión de testing adaptativo no determinista como:

$$\begin{gathered}\text{N-ATEST} := \{(C,E,k) : E \subseteq C \subseteq \{f : I \to \mathcal{P}(O) \setminus \{\emptyset\}\}, \\ \exists i_1 \in I \ \ \forall o_1 \in \bigcup_{f \in C} f(i_1) \\ \exists i_2 \in I \ \ \forall o_2 \in \bigcup_{f \in \{f \in C:\, o_1 \in f(i_1)\}} f(i_2) \\ \vdots \\ \exists i_l \in I \ \ \forall o_l \in \bigcup_{f \in \{f \in C:\, o_m \in f(i_m)\ \forall m < l\}} f(i_l) \\ \vdots \\ \exists i_k \in I \ \ \forall o_k \in \bigcup_{f \in \{f \in C:\, o_m \in f(i_m)\ \forall m < k\}} f(i_k) \\ \{f \in C : o_m \in f(i_m) \ \ \forall m \leq k\} \subseteq E \ \vee \\ \{f \in C : o_m \in f(i_m) \ \ \forall m \leq k\} \subseteq C \backslash E\}\end{gathered}$$

En esta definición los inputs dependen de los inputs y outputs anteriores. Los outputs han de ser consistentes con los inputs y outputs anteriores, es decir, tiene que existir una función en C que pueda dar esos resultados. Las funciones consistentes con todos los inputs y outputs ($\{f \in C : o_m \in f(i_m) \ \ \forall m \leq k\}$) han de ser todas correctas ($\subseteq E$) o todas incorrectas ($\subseteq C \backslash E$).

Interesa considerar aparte el caso en el que el formalismo computacional es determinista. Así, dados dos conjuntos finitos no vacíos I y O definimos el problema de decisión de testing adaptativo determinista como:

$$\begin{gathered}\text{ATEST} := \{(C,E,k) : E \subseteq C \subseteq \{f : I \to O\} \\ \exists i_1 \in I \ \forall o_1 \in \{f(i_1) : f \in C\} \\ \exists i_2 \in I \ \forall o_2 \in \{f(i_2) : f \in C,\ o_1 = f(i_1)\} \\ \vdots \\ \exists i_l \in I \ \forall o_l \in \{f(i_l) : f \in C,\ o_m = f(i_m)\ \forall m < l\} \\ \vdots \\ \exists i_k \in I \ \forall o_k \in \{f(i_k) : f \in C,\ o_m = f(i_m)\ \forall m < k\} \\ \{f \in C : o_m = f(i_m) \ \ \forall m \leq k\} \subseteq E \ \vee \\ \{f \in C : o_m = f(i_m) \ \ \forall m \leq k\} \subseteq C \backslash E\}\end{gathered}$$

Definiremos la versión de optimización del problema de testing adaptativo determinista como:

$$\text{mín}\{k : (C,E,k) \in \text{ATEST}\}$$

A continuación definiremos los problemas TQBF y SET-COVER que son los que reduciremos a N-ATEST y ATEST respectivamente para determinar su complejidad.

Fórmula booleana cuantificada

En el problema de la fórmula booleana cuantificada o True Quantified Boolean Formulae (TQBF) queremos determinar si una fórmula booleana con las variables cuantificadas con *paratodos* y *existes* es satisfactible[6].

Hay distintos enunciados de este problema que permiten que la fórmula esté de cualquier forma, que esté en forma normal conjuntiva o CNF (por sus siglas en inglés), que esté en forma normal disyuntiva o DNF, que esté en 3CNF o que esté en 3DNF[1][6]. Todas estas formas son equivalentes pues se pueden transformar unas en otras en tiempo polinómico manteniendo la satisfacibilidad, tal y como comentamos en el apéndice. Nosotros exigiremos que **la fórmula esté en CNF**.

También obligaremos a que **los *existes* y los *paratodos* se alternen** y no estén negados. Nuevamente, las fórmulas que no cumplan con esto pueden ser transformadas a otras que sí en tiempo polinómico.

Con todo esto la definición formal queda:

$$\begin{aligned}\text{TQBF} := \{\phi(x_1, y_1, x_2, y_2, \ldots, x_k, y_k) : \phi \text{ es CNF y} \\ \exists x_1 \forall y_1 \exists x_2 \forall y_2 \ldots \exists x_k \forall y_k \ \phi(x_1, y_1, x_2, y_2, \ldots, x_k, y_k) = 1\}\end{aligned}$$

Es importante que las variables tengan un orden establecido al ser cuantificadas, pues cambiar el orden de la cuantificación de variables para una misma fórmula puede hacer que el resultado de TQBF cambie. Como por ejemplo en este caso:

$$\exists x_1 \forall y_1 \exists x_2 \forall y_2 \ (x_2 \wedge y_1) \vee (\overline{x}_2 \wedge \overline{y}_1) \qquad \exists x_2 \forall y_2 \exists x_1 \forall y_1 \ (x_2 \wedge y_1) \vee (\overline{x}_2 \wedge \overline{y}_1)$$

la primera instancia de TQBF es cierta tomando $x_2 = \overline{y}_1$ pero la segunda no, pues cuando $y_1 = x_2$ la fórmula no se cumplirá.

TQBF es PSPACE-completo[6].

Set cover

En el problema de decisión de SET-COVER queremos saber si, dado un conjunto finito y un recubrimiento $\{S_l\}_{l=1}^m$ de este, existe un subrecubrimiento de como mucho k subconjuntos:

$$\text{SET-COVER} = \{(\{S_l\}_{l=1}^m, k) : \exists \{S_{l_j}\}_{j=1}^k \subseteq \{S_l\}_{l=1}^m, \bigcup_{j=1}^{k} S_{l_j} = \bigcup_{l=1}^{m} S_l\}$$

En el problema de optimización de SET-COVER queremos saber cual es el mínimo tamaño del subrecubrimiento con el que cubrir el conjunto:

$$\text{mín}\{k : (\{S_l\}_{l=1}^{m}, k) \in \text{SET-COVER}\}$$

El problema de optimización de SET-COVER es log-APX-completo[9][10] y el de decisión es NP-completo[4][11].

Demostraciones

N-ATEST es PSPACE

Comencemos notando que si N-ATEST está en PSPACE entonces ATEST también pues es un caso particular.

Para ver que N-ATEST está en PSPACE mostraremos un algoritmo con pseudocódigo que puede resolver el problema usando una cantidad de memoria polinómica con respecto al tamaño de los datos de entrada.

Supondremos que los conjuntos finitos I y O tienen un orden total, si no lo estableceremos nosotros. La única restricción que tiene que cumplir este orden es que para comparar dos elementos como mucho haga falta usar una cantidad de memoria polinómica con respecto al tamaño de los conjuntos I y O. Esto es sencillo pues para recordar un orden definido explícitamente (que es el peor caso) nos basta con $|I|^2$ y $|O|^2$ espacio.

En este algoritmo tendremos dos vectores i y o de longitud k y la variable l.

Las operaciones aquí realizadas pueden requerir una cantidad constante de variables auxiliares: obtener el mínimo o máximo de un conjunto, obtener el siguiente elemento de un conjunto, comprobar que $o[m] \in f(i[m])$ para todo $1 \leq m \leq k$, comprobar que f está en E o en $C \backslash E$.

El siguiente algoritmo comprueba todas las combinaciones de inputs y outputs posibles. Si una combinación permite distinguir si las funciones consistentes con ella son correctas o no, entonces se cambia el último output elegido para el que no se hayan probado ya todos los valores posibles. Si una combinación no permite distinguir si las funciones consistentes con ella son correctas o no, entonces se cambia el último input elegido para el que no hayamos probado ya todos los valores posibles.

En i y o guardaremos los inputs y outputs que estamos considerando. La variable entera l determina por que test vamos, es decir, en que nivel del árbol de decisiones estamos.

$l = 1$
[0]
$i[l] = \text{mín}(I)$
[1]
$o[l] = \text{mín}(\bigcup_{f \in \{f \in C:\, o[m] \in f(i[m])\ \forall m < l\}} f(i[l]))$
[2]

- si $l < k$:
    - aumentamos l en 1 y vamos a [0].

- si $l = k$:

  - si $\{f \in C : o[m] \in f(i[m])\ \forall m \leq k\}$ está contenido en $C \backslash E$ o en E (es decir, que si existe $f \in E$ tal que $o[m] \in f(i[m])$ para todo $1 \leq m \leq k$ entonces no existe $f \in C \backslash E$ tal que $o[m] \in f(i[m])$ para todo $1 \leq m \leq k$):

    mientras que $l > 0$ y $o[l] = \text{máx} \bigcup_{f \in \{f \in C:\, o[m] \in f(i[m])\ \forall m < l\}} f(i[l])$ se disminuye l de uno en uno.

    si $l = 0$: **ÉXITO**.

    si $o[l]$ no es el máximo valor en $\bigcup_{f \in \{f \in C:\, o[m] \in f(i[m])\ \forall m < l\}} f(i[l])$ entonces $o[l]$ pasa a ser el siguiente output y vamos a [2].

  - si $\{f \in C : o[m] \in f(i[m])\ \forall m \leq k\}$ no está contenido en $C \backslash E$ ni en E:

    mientras que $l > 0$ y $i[l]$ sea el máximo de I disminuimos l en 1.

    si $l = 0$: **FALLO**.

    si $i[l]$ no es el máximo de I entonces $i[l]$ pasa a ser el siguiente input y vamos a [1].

Probar dos veces el mismo input en la misma combinación de tests no va a ayudar. Podemos hacer el algoritmo más eficiente cambiando I por $I \backslash \{i[m] : 1 \leq m < l\}$.

Para resolver el problema de optimización bastaría con empezar con $k = 1$ y, en lugar de fallar, aumentar k en 1 y volver a empezar.

N-ATEST$_k$ es PSPACE-completo

Como ya hemos visto que N-ATEST está en PSPACE nos falta por ver que es PSPACE-duro para demostrar que es PSPACE-completo.

Para demostrar la PSPACE-dureza de N-ATEST tenemos que reducir todos los problemas de PSPACE a N-ATEST ($A \preceq$ N-ATEST $\forall A \in$ PSPACE). Como sabemos que TQBF es PSPACE-completo nos basta con reducir TQBF a N-ATEST. Así, dada una instancia de un problema de PSPACE, podremos transformarla a una instancia de TQBF, y esta podremos transformarla a una instancia de N–ATEST. Todas estas reducciones/transformaciones tendrán que ser en tiempo polinómico.

En resumidas cuentas, tendremos que demostrar que:

$$\phi(x_1, y_1, \ldots, y_k) \in \text{TQBF} \Leftrightarrow (C, E, k) \in \text{N-ATEST}$$

para ciertos C y E definidos a partir de $\phi(x_1, \ldots, y_k)$.

En nuestra demostración el conjunto E será unipuntual. Este es un caso particular de N-ATEST interesante. Que en la demostración usemos este caso nos asegurará que tanto la versión general de N-ATEST como este caso particular son PSPACE-completos.

Las funciones serán necesariamente no deterministas lo cual quiere decir que la versión determinista del problema, ATEST, no tiene porque ser PSPACE-completa.

La idea de la demostración es equiparar los *existes* de N-ATEST con los de TQBF y hacer lo mismo con los *paratodos*. Que cierta combinación de inputs y outputs permita distinguir la función correcta de las incorrectas equivaldrá a que cierta asignación de valores a las variables hagan cierta la fórmula booleana. Eso quiere decir que los inputs de nuestros tests tendrán que representar los valores de las variables x_j, y los outputs tendrán que representar los valores de las variables y_j. Efectivamente, lo que haremos será usar como inputs de los tests los literales x_j y $\overline{x}_j$, que representarán poner la variable x_j a cierto o falso en la fórmula booleana, y los outputs podrán valer 1 o 0 según la correspondiente y_j sea cierta o falsa.

En TQBF, cuando consideremos cierto valor de cierta y_j todavía no tenemos por qué saber si la fórmula se cumplirá o no. Igualmente, en N-ATEST cuando averigüemos el valor de cierto output no tenedremos por qué saber si estamos ante la función correcta o no. Para que esto sea así la función correcta tendrá que poder devolver los valores 0 y 1. De hecho, definiremos una función correcta como la que devuelve de forma no determinista 0 o 1 ante cualquier input. Así, cuando queramos que una función sea descartada con cierto input usaremos

outputs que la única función correcta no pueda devolver, es decir ni 0 ni 1, (usaremos -1 y -2).

Para replicar la fórmula booleana, crearemos una función no correcta por cada cláusula. Queremos que las funciones asociadas a cláusulas que se cumplen sean descartadas. Por ello, las descartaremos inmediatamente si se les pasa como input un literal (x_j o $\overline{x}_j$) que esté en su correspondiente cláusula. También queremos que se descarten si y_j o $\overline{y}_j$ están en la cláusula y el valor que devuelve la implementación testeada coincide con el valor que debe tomar la variable (y_j) para que dicho literal (y_j o $\overline{y}_j$) sea cierto. Para conseguir esto, la función solo tiene que devolver el valor contrario.

Este esquema de reducción tiene un gran problema: los inputs de los tests no tienen porque ser introducidos en orden, mientras que las variables de la fórmula sí que tienen que tomar los valores por orden. Para arreglar esto crearemos una serie de funciones que solo puedan ser descartadas por completo si las variables se introducen en orden. También duplicaremos los inputs (tendremos $x_j, \overline{x}_j, x'_j$ y $\overline{x}'_j$). La idea será que por cada variable haya dos funciones: al introducir una variable se descartará una de las dos funciones, aunque el testeador no sabrá cual de de las dos hasta que no reciba el output, y para descartar la otra función, tendrá que introducir la versión con o sin prima adecuada de la siguiente variable (la que descarte necesariamente la que no pudo descartar antes). Si una variable se introduce antes que su anterior, entonces no habrá forma de garantizar que ambas funciones se descarten. (Las variables inicial y final requerirán un tratamiento especial).

Formalicemos esta idea.

Llamaremos c_i a la i-ésima cláusula disyuntiva de ϕ y diremos que las variables de las que está formada pertenecen a c_i (e.g. si $c_3 = \overline{x}_1 \vee y_2 \vee x_3$ diremos que $\overline{x}_1, y_2, x_3 \in c_3$ pero $x_1, \overline{y}_2, \overline{x}_3, x_2, x_4 \notin c_3$).

Usaremos $\mathrm{I} = \{x_j, \overline{x}_j, x'_j, \overline{x}'_j\}_{j=1}^{k}$ y $\mathrm{O} = \{-2, -1, 0, 1\}$.

Llamaremos g a la función que devuelve 0 o 1 con cualquier input:

$$g(i) = \{0,\, 1\} \quad \forall i \in I$$

E será el conjunto $\{g\}$ y C será:

$\{g,\ f_{c_i}$ para $1 \leq i \leq N$ (nº de clausulas de ϕ), f_l, f'_l para $1 \leq l < k,\ f_0\}$

donde, con $1 \leq i \leq N$ y $1 \leq j \leq k$:

- si $x_j \in c_i$ entonces $f_{c_i}(x_j) = \{-1\}$
- si $\overline{x}_j \in c_i$ entonces $f_{c_i}(\overline{x}_j) = \{-1\}$

- si $y_j, \overline{y}_j \in c_i$ entonces $f_{c_i}(x_j) = f_{c_i}(\overline{x}_j) = \{-1\}$
- si $y_j \in c_i$ pero $x_j, \overline{y}_j \notin c_i$ entonces $f_{c_i}(x_j) = \{0\}$
- si $y_j \in c_i$ pero $\overline{x}_j, \overline{y}_j \notin c_i$ entonces $f_{c_i}(\overline{x}_j) = \{0\}$
- si $\overline{y}_j \in c_i$ pero $x_j, y_j \notin c_i$ entonces $f_{c_i}(x_j) = \{1\}$
- si $\overline{y}_j \in c_i$ pero $\overline{x}_j, y_j \notin c_i$ entonces $f_{c_i}(\overline{x}_j) = \{1\}$
- en el resto de casos $f_{c_i}(x_j) = f_{c_i}(\overline{x}_j) = \{0, 1\}$

Los inputs con prima $(x'_j, \overline{x}'_j)$ generan los mismos resultados que sus versiones sin prima:

$$f_{c_i}(x'_j) = f_{c_i}(x_j) \qquad\qquad f_{c_i}(\overline{x}'_j) = f_{c_i}(\overline{x}_j)$$

Estas funciones son las encargadas de que el problema de testing adaptativo sea aceptado si y solo si todas las cláusulas de ϕ se cumplen. Funciona de la siguiente manera:

Si un literal está en la cláusula c_i entonces la función f_{c_i} será descartada cuando se le pase ese literal porque devolverá un número negativo (la única función correcta, g, solo devuelve 0 o 1). f_{c_i} también será descartada si tanto y_j como $\overline{y}_j$ están en la cláusula.

Si los literales x_j y $\overline{x}_j$ no están en la cláusula pero su correspondiente y_j orx $\overline{y}_j$ sí entonces descartaremos f_{c_i} cuando nuestro test nos devuelva un 1 o, respectivamente, un 0, es decir cuando c_i se cumpla gracias a y_j o a $\overline{y}_j$. En cualquier otro caso f_{c_i} devolverá lo mismo que g: 0 o 1; para no ser descartada.

Por otro lado, para garantizar que las variables se introducen en orden crearemos $2k - 1$ funciones. Cada literal descartará dos de estas funciones excepto el último literal que descartará una. Para que cada variable descarte distintas funciones que el resto será necesario que los literales se introduzcan en orden.

Cuando se introduzca alguna de $x_l, \overline{x}_l, x'_l, \overline{x}'_l$ se descartará f_l o f'_l, y para descartar la función otra habrá que introducir la versión con o sin prima adecuada de $x_{l+1}, \overline{x}_{l+1}, x'_{l+1}, \overline{x}'_{l+1}$.

Para $1 \leq l < k$:

- $f_l(x_l) = \{0\} \quad f_l(x'_l) = \{0\}$
- $f'_l(x_l) = \{1\} \quad f'_l(x'_l) = \{1\}$
- $f_l(x_{l+1}) = \{0, 1\} \quad f_l(x'_{l+1}) = \{-2\}$
- $f'_l(x_{l+1}) = \{-2\} \quad f'_l(x'_{l+1}) = \{0, 1\}$

- $f_l(x_j) = f'_l(x'_j) = \{0, 1\}$ para $j \neq l, l+1$, es decir, el resto de casos.

Las funciones f_l y f'_l se comportan igual con una variable que con su negada:

$$f_l(x_j) = f_l(\overline{x}_j) \quad f_l(x'_j) = f_l(\overline{x}'_j)$$

$$f'_l(x_j) = f'_l(\overline{x}_j) \quad f'_l(x'_j) = f'_l(\overline{x}'_j)$$

Para que alguna de $x_1, \overline{x}_1, x'_1, \overline{x}'_1$ descarte dos funciones tenemos que crear una función adicional que solo sea descartada por alguno de estos inputs:

$$f_0(x_1) = f_0(\overline{x}_1) = f_0(x'_1) = f_0(\overline{x}'_1) = \{-2\}$$

$$f_0(x_j) = f_0(\overline{x}_j) = f_0(x'_j) = f_0(\overline{x}'_j) = \{0, 1\} \text{ para } 1 < j \leq k$$

Y hasta aquí la transformación necesaria. A continuación demostraremos que la instancia de TQBF se cumple si y solo si la instancia de N-ATEST que hemos creado a partir de ella también se cumple.

Nótese que esta transformación se puede hacer en tiempo polinómico con respecto al tamaño de ϕ pues el número de funciones es del orden del número de cláusulas más el número variables y cada una acepta una cantidad de inputs del orden del número de variables.

"TQBF $\Rightarrow$ N-ATEST"

Veamos que si la fórmula booleana cuantificada se cumple entonces la instancia correspondiente de testing adaptativo también:

$$\phi(x_1, \ldots, y_k) \in \text{TQBF} \Rightarrow (C, E, k) \in \text{N-ATEST}$$

Primero veamos que descartaremos todas las f_l si ponemos los $x_j, \overline{x}_j, x'_j, \overline{x}'_j$ en orden y elegimos bien entre las versiones prima y no prima.

Es decir, si

$$i_1 \in \{x_1, \overline{x}_1, x'_1, \overline{x}'_1\}$$

y para $1 \leq l < k$:

- si $o_l < 0$ entonces $i_{l+1} \in \{x_{l+1}, \overline{x}_{l+1}, x'_{l+1}, \overline{x}'_{l+1}\}$
- si $o_l = 0$ entonces $i_{l+1} \in \{x'_{l+1}, \overline{x}'_{l+1}\}$
- si $o_l = 1$ entonces $i_{l+1} \in \{x_{l+1}, \overline{x}_{l+1}\}$

entonces o bien algún output es negativo y todas las funciones consistentes con los inputs y outputs son incorrectas:

$$\{f \in C : o_m \in f(i_m),\, m \leq k\} \subseteq C \backslash E \quad \text{y} \quad \exists o_M < 0$$

o bien todos los outputs son positivos y las funciones f_l, f'_l y f_0 son descartadas:

$$\forall l \; f_l, f'_l, f_0 \notin \{f \in C : o_m \in f(i_m),\, m \leq k\} \quad \text{y} \quad \nexists o_M < 0$$

Si $\exists o_M < 0$ entonces todas las funciones consistentes con los inputs y outputs son incorrectas:

$$\{f \in C : o_m \in f(i_m),\, i \leq k\} \subseteq \{f \in C : 0 > o_M \in f(i_M)\} \subseteq C \backslash E$$

pues la única función correcta g ($E = \{g\}$) solo tiene como outputs 0 y 1.

Si $\nexists o_M < 0$ entonces para todo $1 \leq l < k$:

$$o_l = 0 \Rightarrow \begin{cases} 0 = o_l \notin f'_l(i_l) = \{1\} \\ i_{l+1} \in \{x'_{l+1}, \overline{x}'_{l+1}\} \Rightarrow 0 \leq o_{l+1} \notin f_l(i_{l+1}) = \{-2\} \end{cases}$$

$$o_l = 1 \Rightarrow \begin{cases} 1 = o_l \notin f_l(i_l) = \{0\} \\ i_{l+1} \in \{x_{l+1}, \overline{x}_{l+1}\} \Rightarrow 0 \leq o_{l+1} \notin f'_l(i_{l+1}) = \{-2\} \end{cases}$$

Y como $i_1 \in \{x_1, \overline{x}_1, x'_1, \overline{x}'_1\}$ entonces $0 \leq o_1 \notin f_0(i_1) = \{-2\}$.
Así que $f_l, f'_l, f_0 \notin \{f \in C : o_m \in f(i_m),\, m \leq k\}$.

Veamos ahora que si la fórmula booleana cuantificada se cumple entonces podemos descartar todas las f_{c_i}.
Es decir que si

$$\exists x_1 \forall y_1 \ldots \exists x_k \forall y_k \; \phi(x_1, y_1, x_2, y_2, \ldots, x_k, y_k) = 1$$

y si $x_1 = 1$ para la fórmula entonces tomamos $i_1 \in \{x_1, x'_1\}$
si $x_1 = 0$ entonces tomamos $i_1 \in \{\overline{x}_1, \overline{x}'_1\}$
y para $1 \leq l < k$:
si $o_l < 0$ entonces $i_{l+1} \in \{x_{l+1}, \overline{x}_{l+1}, x'_{l+1}, \overline{x}'_{l+1}\}$
si no, tomando $y_l = o_l$, elegimos $\begin{cases} i_{l+1} \in \{x_{l+1}, x'_{l+1}\} & \text{si } x_{l+1} = 1 \\ i_{l+1} \in \{\overline{x}_{l+1}, \overline{x}'_{l+1}\} & \text{si } x_{l+1} = 0 \end{cases}$
entonces o bien algún output es negativo y todas las funciones consistentes con los inputs y outputs son incorrectas

$$\{f \in C : o_m \in f(i_m),\, m \leq k\} \subseteq C \backslash E \text{ y } \exists o_M < 0$$

o bien todos los outputs son positivos y las funciones f_{c_i} son descartadas:

$$\forall i \; f_{c_i} \notin \{f \in C : o_m \in f(i_m),\, m \leq k\} \text{ y } \nexists o_M < 0$$

Nótese que esta elección de i_l es compatible con la elección que hemos hecho para descartar las funciones f_l, f'_l y f_0 (en la que elegíamos entre las versiones con y sin prima) y que el resultado también es compatible.

Si $\exists o_M < 0$ entonces todas las funciones consistentes con los inputs y outputs son incorrectas:

$$\{f \in C : o_m \in f(i_m),\, i \leq k\} \subseteq \{f \in C : 0 > o_M \in f(i_M)\} \subseteq C \backslash E$$

pues la única función correcta g ($E = \{g\}$) solo tiene como outputs 0 y 1.
Si $\nexists o_M < 0$ entonce para todo $1 \leq i \leq N$ (número de cláusulas de ϕ) y para cualquier $1 \leq j \leq k$:

- si $x_j = 1$ (entonces $i_j \in \{x_j,\, x'_j\}$) y $x_j \in c_i \Rightarrow 0 \leq o_j \notin f_{c_i}(i_j) = \{-1\}$.
- si $x_j = 0$ (entonces $i_j \in \{\overline{x}_j, \overline{x}'_j\}$) y $\overline{x}_j \in c_i \Rightarrow 0 \leq o_j \notin f_{c_i}(i_j) = \{-1\}$.
- si $y_j, \overline{y}_j \in c_i$ como $i_j \in \{x_j, \overline{x}_j, x'_j, \overline{x}'_j\}$ entonces $0 \leq o_j \notin f_{c_i}(i_j) = \{-1\}$.
- si $y_j \in c_i$ pero $\overline{y}_j \notin c_i$ y $y_j = 1$ entonces $f_{c_i}(i_j)$ vale $\{0\}$ o $\{-1\}$ para $i_j \in \{x_j, \overline{x}_j, x'_j, \overline{x}'_j\}$ y $1 = y_j = o_j \notin f_{c_i}(i_j) \subset \{-1, 0\}$.
- si $\overline{y}_j \in c_i$ pero $y_j \notin c_i$ y $y_j = 0$ entonces $f_{c_i}(i_j)$ vale $\{1\}$ o $\{-1\}$ para $i_j \in \{x_j, \overline{x}_j, x'_j, \overline{x}'_j\}$ y $0 = y_j = o_j \notin f_{c_i}(i_l) \subset \{-1, 1\}$.

En definitiva, si c_i se cumple (tiene algún literal con valor 1) entonces $f_{c_i} \notin \{f \in C : o_m \in f(i_m),\, m \leq k\}$. Luego si $\phi(x_1, \ldots, y_k)$ se cumple entonces ninguna f_{c_i} está en $\{f \in C : o_m \in f(i_m),\, m \leq k\}$.

Combinando ambos argumentos: si introducimos las variables en orden y ϕ se cumple entonces podemos descartar o g o el resto de funciones (f_{c_i}, f_l, f_0) y por tanto $(C,\, E) \in$ N-ATEST$_k$.

"N-ATEST $\Rightarrow$ TQBF"

Veamos ahora que la fórmula booleana cuantificada se cumple si la instancia correspondiente de testing adaptativo también:

$$\phi(x_1, \ldots, y_k) \in \text{TQBF} \Leftarrow (C, E, k) \in \text{N-ATEST}$$

Primero demostraremos que para que $(C, E, k) \in$ N-ATEST es necesario usar al menos un input de entre $x_j, \overline{x}_j, x'_j, \overline{x}'_j$ para cada $1 \leq j \leq k$ y, por tanto, como solo podemos usar k inputs entonces tenemos que usar un y solo un literal por cada j. Es decir:

$$(C, E, k) \in \text{N-ATEST} \Rightarrow \nexists j :\, x_j, \overline{x}_j, x'_j, \overline{x}'_j \notin \{i_m\}_{m=1}^{k}$$

Supongamos que $(C, E, k) \in$ N-ATEST y tomemos una elección de inputs $\{i_m\}_{m=1}^k$ con sus correspondientes outputs $\{o_m\}_{m=1}^k$.

Si existe $o_m < 0$ entonces $\{f \in C : o_m \in f(i_m), i \leq k\} \subseteq C \backslash E$, pues $E = \{g\}$ y g solo devuelve 0 o 1.

Consideremos ahora solo el otro caso: $o_m \in \{0, 1\}$ para todo $1 \leq m \leq k$.

En este caso tenemos que $g \in \{f \in C : o_m \in f(i_m), i \leq k\}$ y como $(C, E, k) \in$ N-ATEST entonces $E = \{g\} = \{f \in C : o_m \in f(i_m), i \leq k\}$.

Veamos que esto no es cierto si $\exists j : x_j, \overline{x}_j, x'_j, \overline{x}'_j \notin \{i_m\}_{m=1}^k$:

- Si $j = 1$ entonces f_0 no se podría distinguir de g con ningún input. Es decir, $f_0 \in \{f \in C : o_m \in f(i_m), i \leq k\}$ lo cual es contradictorio.
- Si $j > 1$ y $\nexists i_n \in \{x_{j-1}, \overline{x}_{j-1}, x'_{j-1}, \overline{x}'_{j-1}\}$ entonces para todo m tendríamos que $f_{j-1}(i_m) = f'_{j-1}(i_m) = \{0, 1\}$ y por tanto $f_{j-1}, f'_{j-1} \in \{f \in C : o_m \in f(i_m), i \leq k\}$ contradiciendo nuestra premisa.
- Si $j > 1$ y $\exists! i_M \in \{x_{j-1}, \overline{x}_{j-1}, x'_{j-1}, \overline{x}'_{j-1}\}$ entonces si $o_M = 0$ $f_{j-1} \in \{f \in C : o_m \in f(i_m), i \leq k\}$, pues solo las variables $x_j, \overline{x}_j, x'_j, \overline{x}'_j$ podrían distinguir f_{j-1} de g, y análogamente su $o_M = 1$ entonces tendríamos $f'_{j-1} \in \{f \in C : o_m \in f(i_m), i \leq k\}$.
- Si $j > 1$ y $\exists i_{M_1}, i_{M_2}, \ldots, i_{M_p} \in \{x_{j-1}, \overline{x}_{j-1}, x'_{j-1}, \overline{x}'_{j-1}\}$ entonces en el caso en que $o_{M_1} = o_{M_2} = \ldots = o_{M_p}$ no podríamos descartar f_{j-1} por el razonamiento anterior así que esa elección de i_M no es válida.

Ahora demostraremos que hay que introducir las variables por orden. Es decir:

$$(C, E, k) \in \text{N-ATEST} \Rightarrow \forall j : i_j \in \{x_j, \overline{x}_j, x'_j, \overline{x}'_j\}$$

Supongamos que $(C, E, k) \in$ N-ATEST y tomemos una elección de inputs $\{i_m\}_{m=1}^k$ con sus correspondientes outputs $\{o_m\}_{m=1}^k$ tales que las funciones en C consistentes con esos outputs sean todas "correctas" o todas "incorrectas".

En caso de que exista $o_m < 0$ tendremos que $\{f \in C : o_m \in f(i_m), i \leq k\} \subseteq C \backslash E$, pues $E = \{g\}$ y g solo devuelve 0 o 1. Así que tendremos que buscar la contradicción en los restantes casos: $o_m \in \{0, 1\}$ para todo $1 \leq m \leq k$ y

por tanto $g \in \{f \in C : o_m \in f(i_m), i \leq k\}$. Para que $(C, E, k) \in$ N-ATEST hace falta que en estos casos $E = \{g\} = \{f \in C : o_m \in f(i_m), i \leq k\}$.

Razonemos por reducción al absurdo y tomemos el menor j tal que $i_j \notin \{x_j, \overline{x}_j, x'_j, \overline{x}'_j\}$. Tendremos entonces que $i_j \in \{x_n, \overline{x}_n, x'_n, \overline{x}'_n\}$ para algún $n > j$.

Llamemos $i_l \in \{x_{n-1}, \overline{x}_{n-1}, x'_{n-1}, \overline{x}'_{n-1}\}$ y tendremos que $l > j$ (de lo contrario $l = n-1$ lo cual entra en contradicción con que i_j sea el primer input desordenado). Entonces si $i_j \in \{x_n, \overline{x}_n\}$ (respectivamente $i_j \in \{x'_n, \overline{x}'_n\}$), cuando $o_j \in \{0, 1\}$ tendremos que f_{n-1}(respectivamente f'_{n-1}) $\in \{f \in C : o_m \in f(i_m), i \leq \boldsymbol{j}\}$ y en los casos en los que $o_l = 0$ (respectivamente $o_l = 1$) tendremos que f_{n-1}(respectivamente f'_{n-1})$\in \{f \in C : o_m \in f(i_m), i \leq \boldsymbol{k}\}$, pues solo usando las variables con subíndices n y $n-1$ se puede distinguir estas funciones de g.

Demostraremos ahora que si $(C, E, k) \in$ N-ATEST entonces $\phi(x_1, \ldots, y_k) \in$ TQBF sabiendo ya que $i_j \in \{x_j, \overline{x}_j, x'_j, \overline{x}'_j\}$.

Para resolver ϕ tomaremos:

- $x_j = 1$ si $i_j \in \{x_j, x'_j\}$
- $x_j = 0$ si $i_j \in \{\overline{x}_j, \overline{x}'_j\}$

Los valores de i_j, con $j > 1$, vendrán dados por los valores anteriores de o_{j-1} que tomaremos como $o_{j-1} = y_{j-1}$.

Como $o_j = y_j$ estamos en los casos en que $o_m \in \{0, 1\}$ para todo $1 \leq m \leq k$, luego $g \in \{f \in C : o_m \in f(i_m), i \leq k\}$ y, para que $(C, E, k) \in$ N-ATEST, tendrá que ser $E = \{g\} = \{f \in C : o_m \in f(i_m), i \leq k\}$.

Veamos que si f_{c_i} se ha descartado ($f_{c_i} \notin \{f \in C : o_m \in f(i_m), i \leq k\}$) entonces c_i se cumple.

Para descartar f_{c_i} tiene que existir i_M tal que $o_M \notin f_{c_i}(i_M)$.

Esto se dará, para empezar, en los casos en que $f(i_M) = \{-1\}$ que son:

- $x_M \in c_i$, $i_M \in \{x_M, x'_M\}$ y, por tanto, $x_M = 1$.
- $\overline{x}_M \in c_i$, $i_v \in \{\overline{x}_v, \overline{x}'_M\}$ y, por tanto, $x_M = 0$.
- $y_M, \overline{y}_M \in c_i$.

En todos estos casos $c_i = 1$, es decir, se cumple.

En el resto de casos se dará:

- si $y_M \in c_i$ pero $x_M, \overline{y}_M \notin c_i$ entonces $f_{c_i}(i_M) = \{0\}$ para $i_M \in \{x_M, x'_M\}$ y por tanto f_{c_i} será distinguible si $o_M = 1$.
- si $y_M \in c_i$ pero $\overline{x}_M, \overline{y}_M \notin c_i$ entonces $f_{c_i}(i_M) = \{0\}$ para $i_M \in \{\overline{x}_M, \overline{x}'_M\}$ y por tanto f_{c_i} será distinguible si $o_M = 1$.
- si $\overline{y}_M \in c_i$ pero $x_M, y_M \notin c_i$ entonces $f_{c_i}(i_M) = \{1\}$ para $i_M \in \{x_M, x'_M\}$ y por tanto f_{c_i} será distinguible si $o_M = 0$.
- si $\overline{y}_M \in c_i$ pero $\overline{x}_M, y_M \notin c_i$ entonces $f_{c_i}(i_M) = \{1\}$ para $i_M \in \{\overline{x}_M, \overline{x}'_M\}$ y por tanto f_{c_i} será distinguible si $o_M = 0$.

Es decir que si f_{c_i} es distinguible entonces $y_M \in c_i$ y $o_M = y_M = 1$ o $\overline{y}_M \in c_i$ y $o_M = y_M = 0$. En ambos casos $c_i = 1$.

Ya no hay más casos en los que f_{c_i} sea distinguible porque en el resto de casos $f_{c_i}(i_M) = \{0, 1\}$.

Como todos los c_i se cumplen entonces ϕ se cumple.

ATEST con una única función correcta es log-APX

Reduciremos el problema de optimización ATEST con una única función correcta a el problema de optimización SET-COVER que está en log-APX. Hemos definido estos problemas de optimización como:

$$\text{mín}\{k : (\{S_l\}_{l=1}^{m}, k) \in \text{SET-COVER}\}$$

y

$$\text{mín}\{k : (C, E, k) \in \text{ATEST}\}$$

Luego si conseguimos reducir el problema de decisión de ATEST al de SET-COVER de forma que todas las soluciones que obtengamos (no solo las correspondientes a las óptimas del problema de optimización) en SET-COVER se traduzcan a soluciones de ATEST con exactamente el mismo coste k, entonces tendremos que el resultado del problema de optimización (incluso con aproximaciones) es exactamente el mismo. Esta reducción, evidentemente, preserva la ratio de aproximación (pues tanto la solución optima como las aproximadas son exactamente iguales). A este tipo de reducción se le conoce como S-reducción, que es la más fuerte de una serie de tipos de reducciones que propagan la pertenencia y dureza en log-APX[5].

Para reducir un problema de decisión a otro tenemos que ver que:

$$(C, \{h\}, k) \in \text{ATEST} \Leftrightarrow (\{S_i\}_{i\in I}, k) \in \text{SET-COVER}$$

donde $\bigcup_{i\in I} S_i = C\backslash\{h\}$ y $f \in S_i$, con $i \in I$, si $f(i) \neq h(i)$.

Si $(\{S_i\}_{i\in I}, k) \in$ SET-COVER es porque existe $J \subseteq I$ con $|J| \leq k$ tal que $\bigcup_{i\in I} S_i = C\backslash\{h\} = \bigcup_{j\in J} S_j$ entonces para todo $f \in C\backslash\{h\}$ existe $j \in J$ tal que $f \in S_j$ o, lo que es lo mismo, $f(j) \neq h(j)$. Tomando entonces los elementos de J como inputs en ATEST_k tenemos que si algún output es distinto de lo que respondería h entonces hemos descartado h, si no hemos descartado todas las demás porque para cualquier $f \in C\backslash\{h\}$ algún $j \in J$ hace que $f(j) \neq h(j)$.

Recíprocamente, si $(C, \{h\}, k) \in$ ATEST y elegimos los conjuntos S_j de las j que usamos como inputs en el caso en que todos los outputs sean consistentes con h entonces como todas las funciones en $C\backslash\{h\}$ han sido descartadas es porque $f(j) \neq h(j) \Leftrightarrow f \in S_j$, para algún j de los inputs usados.

Con esto no solo podemos decir que la versión de optimización de ATEST es log-APX, también podemos decir que la versión de decisión de ATEST es NP como SET-COVER.

ATEST es log-APX-duro

Demostraremos ahora que el problema de optimización SET-COVER se puede reducir al problema de optimización ATEST que es log-APX-duro. Igual que antes, para demostrar la reducción entre las versiones de optimización nos basta con reducir las versiones de decisión manteniendo la misma k, es decir:

$$(\{S_l\}_{l=1}^m, k) \in \text{SET-COVER} \Leftrightarrow (C, \{h\}, k) \in \text{ATEST}$$

donde $I = \{1, 2, \ldots, m\}$, $O = \{0, 1\}$, $h \equiv \mathbf{0}$, $C = \{h,\ f_e \text{ con } e \in \bigcup_{1 \leq l \leq m} S_l\}$ y $f_e(l) = 1$ si $e \in S_l$, $f_e(l) = 0$ si $e \notin S_l$. Es decir, los conjuntos son inputs y sus elementos son funciones que devuelven 1 o 0 según el elemento esté en el conjunto input o no.

Si $(\{S_l\}_{l=1}^m, k) \in$ SET-COVER es porque $\exists \{S_{l_j}\}_{j=1}^k \subset \{S_l\}_{l=1}^m$ que cumple $\bigcup S_l = \bigcup S_{l_j}$ y entonces para todo $e \in \bigcup S_l$ existe S_{l_M} tal que $e \in S_{l_M}$ y por tanto $f_e(l_M) = 1$. Es decir que si tomamos los inputs como $i_j = l_j$ todas las funciones f_e devolverán 1 para algún input, luego podrán ser distinguidas de la función h, que es idénticamente nula. O más formalmente: si existe $o_M = 1$ entonces $h \notin \{f \in C : o_M = 1 = f(i_M)\} \subseteq \{f \in C : o_m = f(i_m),\ m \leq k\} \subseteq C \backslash E$ y si $o_m = 0$ para todo $1 \leq m \leq k$ entonces para cualquier $e \in \bigcup S_l$, $f_e \notin \{f \in C : o_m = 0 = f(i_m),\ m \leq k\}$ pues como hemos visto existe $i_M = l_M$ tal que $f_e(i_M) = 1$ y por tanto $\{f \in C : o_m = f(i_m),\ m \leq k\} \subseteq E = \{h\}$.

Recíprocamente si $(C, E, k) \in$ ATEST tomaremos $l_j = i_j$ con $o_m = 0$ para $1 \leq m \leq k$, lo cual es posible porque $h \equiv \mathbf{0}$. Con estos inputs y outputs sabemos que $\{f \in C : o_m = f(i_m),\ m \leq k\} = E = \{h\}$ porque $(C, E, k) \in$ ATEST. Luego para todo $e \in \bigcup S_l$ existe $l_M = i_M$ tal que $f_e(i_M) = 1$ y por tanto $e \in S_{l_M}$. Así que $\bigcup S_l = \bigcup S_{l_j}$.

Esta también es una S-reducción: las soluciones de uno y otro problema tienen exactamente el mismo coste[5].

Con esto no solo podemos decir que la versión de optimización de ATEST es log-APX-duro, también podemos decir que la versión de decisión de ATEST es NP-duro como SET-COVER.

Conclusiones

Como ya adelantábamos en la introducción, hemos determinado la complejidad computacional distintas versiones del testing adaptativo (salvo la versión determinista con múltiples funciones correctas, para la que no hemos encontrado resultado de completitud, aunque sí dureza).

La siguiente tabla resume los resultados obtenidos:

	Una función correcta	Múltiples funciones correctas
No determinista	PSPACE-completo	
Determinista	log-APX-completo (optimización) NP-completo (decisión)	log-APX-duro (optimización) PSPACE, NP-duro (decisión)

Determinar la complejidad exacta del caso determinista con múltiples funciones correctas queda como trabajo futuro.

As we already discuss in the introduction, we've determined the computational complexity of different variants of the adaptive testing (except the deterministic version with multiple correct functions, for which we haven't found a completeness result, but just a harndness one). The next table summarizes the obtained results:

	One correct function	Multiple correct functions
No deterministic	PSPACE-complete	
Deterministic	log-APX-complete (optimization) NP-complete (decision)	log-APX-hard (optimization) PSPACE, NP-hard (decision)

Determining the exact complexity of the deterministic case with multiple correct functions is left as future work.

Apéndice

Comentaremos brevemente como transformar una fórmula cuantificada cualquiera en una en la que los cuantificadores estén alternados y la expresión booleana esté en forma normal conjuntiva (el caso de la forma normal disyuntiva es análogo).

Primero quitaremos los cuantificadores negados usando las siguientes equivalencias:

$$\nexists x\, \psi(x) \Leftrightarrow \forall x\, \neg\psi(x) \qquad\qquad \nforall x\, \psi(x) \Leftrightarrow \exists x\, \neg\psi(x)$$

donde ψ puede contener más cuantificadores y la negación se aplicaría al primero de ellos.

Para que los existes y para todos se alternen, basta con crear variables auxiliares que no se usen. Esto hace que las fórmulas no sean equivalentes, pues una tiene más variables que la otra, pero sí se mantiene la satisfacibilidad.

Para convertir la fórmula booleana a CNF primero tenemos que quedarnos solo con variables y operaciones básicas: conjunciones, disyunciones y negaciones. Las implicaciones, disyunciones exclusivas y equivalencias hay que expresarlas en términos de operaciones básicas.

A continuación hay que usar las leyes de De Morgan y la doble negación para que las negaciones solo se apliquen a variables.

El último paso es el que más problemas da, pues consiste en cambiar disyunciones por conjunciones y viceversa usando la ley distributiva. Esto puede causar que la nueva fórmula tenga un tamaño exponencial comparado con la fórmula original. En lugar de esto, lo que habrá que hacer es introducir nuevas variables.

Lo que haremos para transformar una disyunción en una conjunción será, por cada cláusula conjuntiva original, crear una variable cuantificada con un existe. Todas las nuevas variables se pondrán en una disyunción (así que al menos una tendrá que ser cierta), y por cada literal original y cada claúsula conjuntiva original en la que estuviese se crea una disyunción que contiene el literal con la nueva variable (asociada a la cláusula en la que estaba el literal) negada. Así, como al menos una variable nueva tendrá que ser cierta, obligará a que se cumplan todos los literales que estaban en la cláusula a la que está asociada dicha nueva variable[7][12].

Para convertir una fórmula de CNF a 3CNF hay que dividir sucesivamente las disyunciones en dos hasta que ninguna tenga más de 3 variables. Para cada división creamos una nueva variable, dos de las variables originales se ponen en disyunción con la nueva variable, el resto de variables se ponen con la nueva variable pero negada. Una de estas disyunciones será cierta gracias a la nueva variable, la otra tendrá que hacerse cierta por una de las variables originales.

Bibliografía

[1] Sipser, Michael. *Introduction to the theory of computation*. Vol. 2. Boston: Thomson Course Technology, **2006**.

[2] Papadimitriou, C. H. *Computational complexity*. Addison Welsey, Reading, Massachusetts (**1994**).

[3] Ausiello, Giorgio, and Paschos, Vangelis. *Approximability preserving reduction*. (**2005**).

[4] Cormen, Thomas H., et al. *Introduction to algorithms*. MIT press, **2009**.

[5] Crescenzi, Pierluigi. *A short guide to approximation preserving reductions*. Proceedings of Computational Complexity. Twelfth Annual IEEE Conference. IEEE, **1997**.

[6] Stockmeyer, Larry. *Classifying the computational complexity of problems*. The journal of symbolic logic 52.1 (**1987**): 1-43.

[7] Arora, Sanjeev, and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, **2009**.

[8] Rodriguez, Ismael, Luis Llana, and Pablo Rabanal. *A general testability theory: classes, properties, complexity, and testing reductions*. IEEE Transactions on software engineering 40.9 (**2014**): 862-894.

[9] Johnson, David S. *Approximation algorithms for combinatorial problems*. Journal of computer and system sciences 9.3 (**1974**): 256-278.

[10] Escoffier, Bruno, and Vangelis Th Paschos. *Completeness in approximation classes beyond APX*. Theoretical computer science 359.1-3 (**2006**): 369-377.

[11] Karp, Richard M. *Reducibility among combinatorial problems. Complexity of computer computations*. Springer, Boston, MA, **1972**. 85-103.

[12] Luis Rademacher. *Computability and Complexity Course Notes*. CSE 6321 at Ohio State University.