Complejidad Computacional

3-SAT; 2-SAT; Max2SAT & NAESAT

Emmanuel Cruz Hernández

3-SAT

El problema 3-SAT es un caso especial del problema SAT, en el que cada una de las cláusulas contiene exactamente 3 literales.

Por demostrar que el problema 3SAT es NP.
Dado un ejemplar de 3SAT con n variables distintas. Se asigna un valor de verdad aleatorio a cada una de las variables, y como hay n variables, asignarles un valor aleatorio a cada variable toma tiempo O(n). Una vez asignados los valores aleatoriamente, se verifica la satisfacibilidad de cada cláusula recorriendo cada una. Todo esto toma tiempo polinomial.
Por lo tanto, 3-SAT $\in$ NP.

Por demostrar que el problema 3-SAT es NP-Completo.

Se realiza la reducción SAT α 3SAT.
Sea C un conjunto de cláusulas formado a partir del conjunto finito de variables U. Para formar las cláusulas del ejemplar del problema SAT a cláusulas para el problema 3SAT se consideran los siguientes casos para $c \in C$:

- Si la cláusula c tiene sólo una literal l, entonces repetimos la misma literal dos veces más, de tal forma que se genera una nueva cláusula con la forma: $(l \vee l \vee l)$.
- La cláusula c tiene dos literales l_1 y l_2, entonces se repite alguna de ellas en la cláusula. Sin pérdida de generalidad, supongamos que se repite la cláusula l_1, así que tendríamos una nueva cláusula de la forma: $(l_1 \vee l_2 \vee l_1)$.
- Si la cláusula c ya tiene exactamente 3 literales, no se le hace alguna modificación, ya que ya tiene la forma de una cláusula en 3SAT.
- Si la cláusula c tiene k literales, donde k>3. La cláusula en el problema SAT tiene la forma c = $l_1 \vee l_2 \vee l_3 \vee \ldots \vee l_k$. Sean z_1, z_2, ..., z_{k-3}, $k-3$ variables

nuevas. Lo que hacemos es reemplazar la cláusula *c* con las *k-3* cláusulas de la siguiente forma:
$(l_1 \vee l_2 \vee z_1), (l_3 \vee -z_1 \vee z_2), (l_4 \vee -z_2 \vee z_3), ..., (l_{k-2} \vee -z_{k-4} \vee z_{k-3}), (l_{k-1} \vee l_k \vee -z_{k-3})$
Con esta asignación se cumplen las siguientes propiedades:

1. Dada una asignación de verdad para las variables x1, ..., xn que hacen que *c* sea verdadera, así que existe una asignación para z_1, z_2, ..., z_{k-3} tal que la conjunción de las cláusulas es verdadera.
2. En caso de que la asignación haga que *c* sea falso, entonces no existe alguna asignación para z_1, z_2, ..., z_{k-3} que hagan que la conjunción sea verdadera.
    a. Para cada cláusula se utilizará un juego nuevo de variables *z* en este caso.

Al transformar todas las cláusulas toma tiempo polinomial, por lo que podemos concluir que SAT α 3SAT y por lo tanto, 3SAT $\in$ NP-Completo.

Por ejemplo: Si tenemos las cláusulas:
$c_1 = l_1 \vee l_2 \vee l_3 \vee l_4$
$c_2 = l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5$
$c_3 = l_1 \vee l_2$
$c_4 = l_1$

De tal forma que tenemos una expresión como $c, \wedge c_2 \wedge c_3 \wedge c_4$.
La transformación de las cláusulas serían las siguientes:

$c'_1 = (l_1 \vee l_2 \vee z1) (l_3 \vee l_4 \text{ -z1})$
$c'_2 = (l_1 \vee l_2 \vee z1) (l_3 \vee \text{-z1} \vee z2) \ (l_4 \vee l_5 \vee \text{-z2})$
$c'_3 = (l_1 \vee l_2 \vee l_1)$
$c'_4 = (l_1 \vee l_1 \vee l_1)$

De tal forma que la transformación estaría compuesta por las cláusulas $c', \wedge c'_2 \wedge c'_3 \wedge c'_4$.

Proposición: 3SAT $\in$ NP-Completo incluso si cada variable está restringida a aparecer a lo más dos veces.

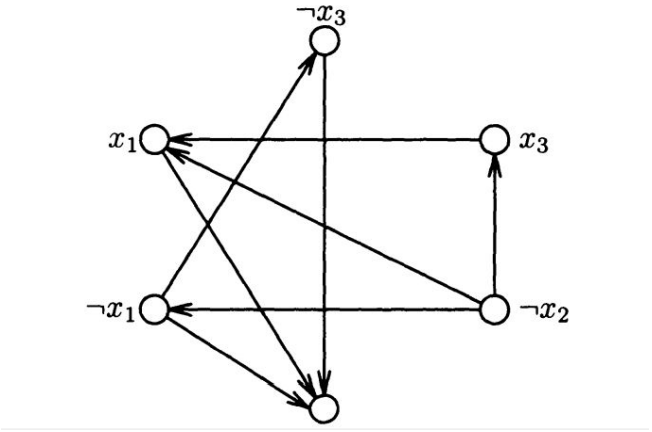
Por demostrar que el problema 3-SAT es NP-Completo.

Se reduce 3-SAT a 3-SAT con ciertas restricciones. La restricción es quitar las variables que aparecen repetidas varias veces.

Sea x una variable que aparece k veces. Se reemplaza la primera aparición de x con una nueva variable y. Ahora se reemplaza la segunda aparición de x con una nueva variable z, de tal forma que se crean k nuevas variables. Cada una de las variables reemplaza la variable x.
Al realizar este reemplazo de variables en las variables que se repiten, se garantiza que sólo aparecen una sola vez. Falta garantizar que las k nuevas variables que se crearon por cada variable que se repite varias veces toman el mismo valor de verdad.
Las nuevas cláusulas creadas a partir de las nuevas variables son ciertas si y sólo si las variables agregadas tienen todas el mismo valor de verdad. Con la modificación hecha anteriormente, cada variable aparece sólo una vez para la primera parte de la expresión, y además, aparece dos veces en las nuevas cláusulas, por lo que no se rompe la restricción de que una variable aparezca a lo más 3 veces. En el caso de las literales, estas aparecen una vez cada una, al igual que en las nuevas cláusulas.

Sea ϕ un ejemplar de 2SAT, un conjunto de cláusulas con dos literales cada una, y sea una gráfica G(ϕ) tal que los vértices de G son las variables de ϕ y sus negaciones, además decimos que hay una arista dirigida (α,β) si y sólo si existe una cláusula (-$\alpha \vee \beta$) (o ($\beta \vee$ -α)) en ϕ. Estas aristas capturan la implicación lógica de ϕ. Como resultado, G(ϕ) tiene una simetría: si (α,β) es una arista, entonces también lo es (-$\alpha \vee$ -β). Por ejemplo:

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee x_3)$$



Dicho lo anterior, por transitividad de la implicación, cualquier camino en G(ϕ) es una implicación lógica válida.

Teorema: Sea ϕ un ejemplar de 2SAT. ϕ es insatisfacible si y sólo si existe una variable *x* tal que hay en G(ϕ) caminos de *x* a *-x* y de *-x* a *x*.
DEM:
⇒ Supongamos que los caminos existen para alguna variable *x* y que ϕ es satisfacible bajo una asignación de verdad T. Supongamos que T(*x*) =*true*, y con esto existe un camino de *x* a *-x* al cumplirse que T(x) = *true* y T(-x) = *false*, debe haber una arista (α, β) a lo largo de este camino, tal que T(α) = *true* y T(β) = *false*. Sin embargo, desde (α, β) hay una arista de G(ϕ) tal que la cláusula (-$\alpha \vee \beta$) está en ϕ, pero está cláusula no es satisfacible por T, lo cual es una contradicción. Por lo tanto ϕ es insatisfacible cuando dichos caminos existe.
⇐ Supongamos que no existe ninguna variable con dichos caminos en G(ϕ). Se construye una asignación de verdad tal que ninguna arista de G(ϕ) vaya de *true* a *false*. Para esta construcción se consideran los siguientes pasos:

- Elegir un vértices α cuyo valor de verdad aún no ha sido definido tal que no exista un camino de α a -α.
- Todos los vértices de G(ϕ) que sean alcanzables desde α se les asigna un valor de verdad *true*.
- Se le asigna un valor de verdad *false* a las negaciones de dichos vértices. Las negaciones son los vértices desde los cuales se puede alcanzar -α. Este paso está bien definido, ya que si existiera algún camino desde α hasta β y -β entonces tendríamos caminos que van desde β y -β hasta -α, lo cual es una contradicción con la hipótesis definida. De hecho, si existiera un camino desde α a algún vértice que fue asignado con *false*, entonces α debería ser predecesor de ese vértice y también debería haberse marcado como *false*, lo cual no puede suceder.

Finalmente, se repiten estos pasos hasta que todos los vértices tengan asignado un valor de verdad. Mientras se asume que no hay caminos desde ninguna *x* a *-x* y de regreso, a todos los nodos se les asignará un valor de verdad. Cómo se garantiza que los sucesores de un vértice asignado a *true* serán *true* y los sucesores de un vértice asignado a *false* serán *false*, no hay ninguna arista que vaya de *true* a *false* y por lo tanto ϕ es satisfacible.

Algunas aplicaciones de 3SAT está en el diseño de redes para un ejemplar del problema con 50 variables booleanas y 218 cláusulas. Otra aplicación está en el diseño de circuitos de control de temperatura integrado en aparatos y sirve para que el sistema no modifique la temperatura una vez encontrada una temperatura perfecta dependiendo de la temperatura actual.

2-SAT

Corolario: 2SAT $\in$ NL y por tanto, 2SAT $\in$ P.
Antes de comenzar con la demostración, es importante definir NL. Es una clase de problemas decidibles por un algoritmo no determinístico en espacio logarítmico.
DEM:
Como NL es una clase cerrada bajo el complemento, se debe mostrar la posibilidad de reconocer expresiones insatisfacibles en NL.
Para demostrarlo usaremos un teorema. **Teorema:** Dada una gráfica G con un vértice x, el número de vértices alcanzables desde x en G se pueden computar con una Máquina de Turing no determinista en espacio $log\ n$.
Usando el teorema anterior, se puede probar la condición del teorema en espacio logarítmico de forma no determinista al elegir de forma aleatoria a x y sus caminos hacia $-x$ y viceversa. Así que 2SAT $\in$ NL, y además 2SAT $\in$ P.

MAX2SAT

Dado un ejemplar ϕ de 2CNF y un entero positivo k. El problema MAX2SAT consiste en determinar si existen al menos k cláusulas satisfacibles en ϕ.

Para demostrar que MAX2SAT es NP.
Se asigna un valor de verdad aleatorio a cada una de las variables, y como hay n variables, asignarles un valor aleatorio a cada variable toma tiempo O(n). Una vez asignados los valores aleatoriamente, se verifica la satisfacibilidad de cada cláusula recorriendo cada una y además, llevar un contador de la cantidad de cláusulas que se satisfacen. Si ese contador es al menos k entonces la asignación aleatoria de valores a las variables es un ejemplar de MAX2SAT, de lo contrario, decimos que no lo es. Todo esto toma tiempo polinomial.
Por lo tanto, MAX2SAT $\in$ NP.

Algunas aplicaciones de 2SAT es el posicionamiento geométrico de objetos libres de conflicto: ayuda a la colocación de etiquetas textuales en las características de un diagrama. Otra aplicación está en la tomografía discreta: ayuda a recuperar formas de secciones transversales, tal como resolver acertijos de nonogramas basados en cuadrículas y números y en este caso, el conjunto de cuadrados a determinar representa los píxeles oscuros en una imagen binaria de 2SAT.

Teorema: MAX2SAT es NP-Completo
DEM:
Consideremos el siguiente ejemplo, que es un conjunto de diez cláusulas:

(x)(y)(z)(w)(-x ∨ -y)(-y ∨ -z)(-z ∨-x)(x ∨ -w)(y ∨ -w)(z ∨ -w)

Esta es una manera de no satisfacer todas las cláusulas (por ejemplo, para satisfacer las primeras 4 cláusulas, se deben omitir o perder las siguientes tres cláusulas). Es importante destacar que las diez cláusulas anteriores cumplen con la propiedad de que cualquier asignación de verdad donde se satisfaga (x ∨ y ∨ z) se puede extender para satisfacer siete de las cláusulas y no más, mientras que el resto de asignaciones posibles, se pueden extender para satisfacer sólo seis de las diez cláusulas.

Se reduce 3SAT α MAX2SAT
Dado un ejemplar ϕ de 3SAT, se construye un ejemplar R(ϕ) de MAX2SAT de la siguiente manera:
Para cada cláusula $C_i = (\alpha \vee \beta \vee \gamma)$ de ϕ agregamos las diez cláusulas de la derecha con α, β y γ en vez de *x, y* y *z*, se reemplaza *w* por una nueva variable w_i particular para C_i. A estas diez cláusulas de R(ϕ) se les denominará *grupo* y corresponde a una cláusula de ϕ. Si ϕ tiene *m* cláusulas, entonces R(ϕ) tiene 10m cláusulas. Dado un objetivo k, decimos que el objetivo se puede alcanzar en R(ϕ) si y sólo si ϕ es satisfacible.

⇒ Supongamos que k = 7m cláusulas pueden satisfacer
Sabemos que en cada *grupo* se pueden satisfacer a lo más 7 cláusulas y que hay *m* grupos, así que 7 cláusulas de cada grupo deben ser satisfechas en cada uno, y esto hace que ϕ sea satisfacible.

$\Leftarrow$ Cualquier asignación de verdad que satisfaga todas las cláusulas en ϕ puede ajustarse para satisfacer las 7m cláusulas en R(ϕ) definiendo el valor de verdad w_i de cada grupo.

Con esto, se puede concluir que MAX2SAT es NP-Completo.

NAESAT

Este problema consiste en determinar si una fórmula con la forma 3CNF es satisfacible bajo alguna asignación de verdad con la restricción de que en ninguna cláusula puede haber tres literales con el mismo valor de verdad.

Para demostrar que NAESAT es NP.
Se asigna un valor de verdad aleatorio a cada una de las variables, y como hay *n* variables, asignarles un valor aleatorio a cada variable toma tiempo O(n). Una vez asignados los valores aleatoriamente, se verifica la satisfacibilidad de cada cláusula recorriendo cada una y además, verificar que las tres literales de cada cláusula no tienen el mismo valor de verdad. Si alguna de las cláusulas no cumple esta restricción, decimos que el ejemplar generado no es NAESAT, en caso contrario, sí lo es. Todo esto toma tiempo polinomial.
Por lo tanto, NAESAT $\in$ NP.

Se reduce SAT α NAESAT
Dado un circuito C, el objetivo es crear una expresión booleana representada por R(C) tal que R(C) es satisfacible si y sólo si C es satisfacible. Las variables de R(C) tiene todas las variables que tiene C y para cada compuerta *g* en C también se tendrá una variable en R(C), que también se llamará *g*. Finalmente, para compuerta en C se crearán algunas cláusulas en R(C) y se consideran los siguientes casos:

- Si *g* es una compuerta NOT cuyo predecesor en C es una compuerta *h*, entonces se agregan las cláusulas (-g $\vee$ -h) $\wedge$ (g $\vee$ h).
- Si *g* es una compuerta OR con predecesores *h* y *h'* , se agregan las cláusulas (-h $\vee$ g) $\wedge$ (-h $\vee$ g) $\wedge$ (h $\vee$ h' $\vee$ -g).
- Si *g* es una compuerta AND con predecesores *h* y *h'*, se agregan las cláusulas (-g $\vee$ h) $\wedge$ (-g $\vee$ h') $\wedge$ (-h $\vee$ -h' $\vee$ g).
- Si *g* es la compuerta de salida agregamos a R(C) la cláusula (g).

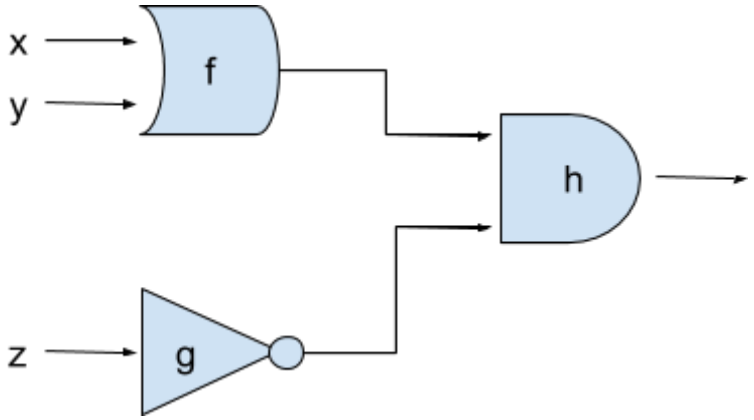
Siguiendo estos pasos, se generan cláusulas que pueden tener una, dos o tres literales. Si hay alguna cláusula con menos de tres literales se les agrega una literal que sea la misma. Por ejemplo, si tenemos la cláusula (l ∨ t), la cláusula se vería, sin pérdida de generalidad, como (l ∨ t ∨ l) y con las cláusulas de una sola literal, (l) la nueva cláusula se vería como (l ∨ l ∨ l).
Hasta este momento, se afirma que el conjunto de cláusulas resultantes es satisfacible si y sólo si C es satisfacible. Llamemos *z* a la literal que se repite en las cláusulas de menos de tres literales.

⇒ Supongamos que existe una asignación de verdad T que satisface todas las cláusulas con la restricción del problema NAEST. Dicha la suposición anterior, el complemento de la asignación de verdad -T también satisface todas las cláusulas con la restricción del problema NAEST. Esto por lo que se comentó anteriormente sobre la cerradura del complemento.
En alguna de las dos asignaciones *z=false*. En esta asignación de verdad ya se satisfacían todas las cláusulas antes de agregar a *z*, por lo que la reducción hecha es una asignación de verdad que satisface al circuito C.
⇐ Supongamos que hay una asignación de verdad que satisface al circuito C. Esto implica que existe una asignación T que satisface todas las cláusulas en un sentido 3SAT y tomamos T(z)=false. Se afirma que en ninguna cláusula todas las literales son *true*, ya que:

1. En el grupo correspondiente a la compuerta NOT [(-g ∨ -h ∨ z) (g ∨ h ∨ z)] todas las cláusulas tienen z, por lo que no todas sus literales son *true*.
2. En el grupo correspondiente a la compuerta AND [(-g ∨ h ∨ z) (-g ∨ h' ∨ z) (-h ∨ -h' ∨ g)] las primeras dos cláusulas tienen *z* por lo que no todas son *true*. Finalmente, para la tercera cláusula, si todas toman el valor de *true*, entonces la primera cláusula no se satisface.
3. Para el grupo correspondiente a la compuerta OR [(-h g z) (-h' g z) (h h' -g)], las primeras dos cláusulas tienen a *z*, así que no todas son *true*. Si todas las literales de la tercera cláusula son true, entonces la primera cláusula no se satisface.

Por lo tanto, NAESAT ∈ NP-Completo.

Si tenemos el circuito C siguiente:



El circuito SAT C se reduce a NAEST con las siguientes cláusulas, siguiendo los pasos de construcción.

R(C) = { (-x ∨ f ∨ w), (-y ∨ f ∨ w), (x ∨ y ∨ -f), (-z ∨ -g ∨ w), (z ∨ g ∨ w), (-h ∨ f ∨ w), (-h ∨ g ∨ w), (-f ∨ -g ∨ h), (h ∨ w ∨ w) }

De esta forma, el circuito se puede construir para que sea un ejemplar de 3SAT.

Referencias

- PAPADIMITRIOU, *C. Computacional Complexity*, Primera Edición, Addison-Wesley Publishing Company, EEUU, 1994.
- https://www.researchgate.net/figure/Application-to-3-SAT-A-Network-design-for-a-problem-instance-with-50-Boolean_fig3_299494543
- Carlos Guillén, Dr. Aurelio López, Dr. Guillermo De Ita, *Diseño de Algoritmos Combinatorios para # SAT y su Aplicación al Razonamiento Proposicional*, Coordinación de Ciencias Computacionales INAOE, 2005.