Diseño y Análisis de Algoritmos. Problema 3: Tito Quita y Pon

Jorge Soler González y Ernesto Alfonso Hernández

Facultad de Matemática y Computación, Universidad de La Habana, La Habana, Cuba

1. Definición del Problema

Tito quita y pon

Tito cayó en arresto domiciliario tras ser vinculado a una serie de negocios turbios. Tras días de encierro llegó a un punto en el que estaba absolutamente aburrido. De pronto, encontró un pequeño juego de mesa que, según la cubierta, se llamaba "Quita y pon". El juego contaba con una colección de cajitas. Cada cajita tenía 3 agujeros de colores. Habían k colores distintos. Además tenía con una bolsa con de pelotas, cada una con el tamaño exacto para rellenar un agujero. La cantidad de pelotas era suficiente para rellenar todos los agujeros.

Al inicio del juego, algunos agujeros están llenos con su pelota correspondiente. Tito puede quitar o poner pelotas en los agujeros siempre que cumpla con que si interactúa con un agujero de color c, debe interactuar con todos los agujeros de ese mismo color en todas las cajas existentes. Se entiende interactuar como quitar (agujero lleno) o poner una pelota (agujero vacío). El objetivo del juego es encontrar si existe un conjunto de interacciones que, luego de ejecutadas resulten en que todas las cajas tengan al menos un agujero lleno y al menos un agujero vacío.

2. Modelación del Problema

Para este problema vamos a asumir que los agujeros de una caja tienen colores diferentes. Sabemos que el juego empieza con un estado inicial en el cual hay agujeros llenos y agujeros vacíos. Ahora, sea k_i, k_j, k_z tres colores que pertenecen a los agujeros de una caja. El estado final que debe tener esa caja es uno en el que al menos un agujero de uno de esos colores esté lleno y otro vacío, por lo tanto cada caja se puede representar como una fórmula booleana de la siguiente manera:

$$(k_i \wedge k_j \wedge \neg k_z) \vee (k_i \wedge \neg k_j \wedge k_z) \vee (\neg k_i \wedge k_j \wedge k_z) \vee (\neg k_i \wedge \neg k_j \wedge k_z) \vee (\neg k_i \wedge k_j \wedge \neg k_z) \vee (k_i \wedge \neg k_j \wedge \neg k_z) \tag{1}$$

Esta fórmula representa todos los estados posibles en los que puede terminar una caja siendo k_i, k_j, k_z los colores que representan cada uno de los agujeros de la caja, de modo que si una de esas variables está positiva en la fórmula

significa que el agujero que corresponde a ese color en la caja está lleno, y si está negada significa que está vacío, luego el juego se representaría como una forma normal conjuntiva de esas cajas, donde cada cláusula es una caja y cada caja es una forma normal disyuntiva. Llegado a este punto, pensamos en reducir esa fórmula a una más pequeña pero equivalente, la siguiente fórmula es una fórmula reducida y equivalente a (1):

$$(k_i \wedge \neg k_j) \vee (k_j \wedge \neg k_z) \vee (\neg k_i \wedge k_z) \tag{2}$$

Ahora demostremos que (1) $\equiv$ (2), a partir de sus tablas de verdad:

k_i	k_j	k_z	(1)	(2)
1	1	1	0	0
1	0	1	1	1
1	1	0	1	1
1	0	0	1	1
0	1	1	1	1
0	0	1	1	1
0	1	0	1	1
0	0	0	0	0

En este punto sabemos que cada caja se puede representar por la fórmula (2), con todas estas transformaciones a lo que queremos llegar es a poder representar el juego como una formula normal conjuntiva de 3 variables por cláusula, con la fórmula 2 casi lo logramos, lo que nos queda es hacer un cambio de variable con las cláusulas de (2), para esto hacemos $y_i \Leftrightarrow$ (la cláusula), en la fórmula (2) todas las cláusulas tienen la misma forma, una variable positiva unida con otra negativa por un $\wedge$, por lo tanto todos los $\Leftrightarrow$ tienen la misma forma. Una vez que tengamos los cambios de variables hechos nos queda el juego representado en una forma normal conjuntiva con 3 variables en cada cláusula, pero a esa fórmula hay que añadirle las condiciones de los cambios de variables para que esté consistente. Para transformar los $\Leftrightarrow$ en una fórmula normal conjuntiva de 3 variables por cláusula nos guíamos por el algoritmo planteado en el Introduction to algorithms en la página 1084, que también busca una fórmula equivalente con tablas de verdad. por cada cambio de variable se añaden 4 nuevas cláusulas a la fórmula original.

En este punto el problema está representado por una fórmula normal conjuntiva de 3 variables en cada cláusula, y la solución al problema está en ver si dado un estado de las variables, esa fórmula está satisfecha. Este es un problema clásico dentro de la clase NP-Completo llamado 3-SAT. Vamos a enfocarnos en las formas de resolver el 3-SAT con la fórmula que representa nuestro problema

3. Primera Solución

Como primera solución se utiliza un algoritmo de fuerza bruta con el enfoque de backtracking que básicamente lo que hace es ir cambiando los valores a las

variables y comprobando si se satisface la fórmula en cada momento, de manera tal que se escojan todas las posibles combinaciones de valores.

El código de este algoritmo está en el archivo tools.py en el método $brute_force$. Este algoritmo es exponencial de orden $O(2^n)$ donde n es la cantidad de variables

4. Mejor solución encontrada

Utilizamos un algoritmo estocástico llamado Walk-SAT en específico el random-Walk-SAT, que funciona de la siguiente manera: Se verifica si todas las cláusulas están satisfechas dado un estado, de no ser así se escoge una de las cláusulas insatisfechas al azar, luego dentro de esta cláusula se escoge una variable al azar y se cambia su valor en toda la fórmula, luego se revisa si la fórmula está satisfecha, de lo contrario se vuelve a repetir el proceso una cantidad n de veces. Sobre este algoritmo se hace un trabajo estadísitico y se vel comportamiento del mismo para las instancias del problema. La implementacion del mismo se encuentra en tools.py en el metodo walkSat.

5. Conclusiones

Agregar que la polinomialidad del problema de 3-SAT está dada por la relación entre la cantidad de cláusulas y la cantidad de variables, existen algoritmos que para para una relación menor que 4 lo resuelven en tiempo polinomial, pero cuando hablamos de que lo resuelven en tiempo polinomial estamos hablando del caso promedio dado que son algoritmos estocásticos, en el caso de nuestro problema la cantidad de cláusulas es $12 * n$ donde n es la cantidad de cajas en el juego dado que por cada caja hay una cláusula y cada cláusula tiene 3 nuevas variables que generan 4 cláusulas cada una. La cantidad de variables sería $3 * n + k$ dado que por cada caja se genera 3 nuevas variables, luego la relación nuestra es menor que 4, por tanto sabemos que existen algoritmos que para el caso promedio del problema lo resuelve en tiempo polinomial, uno de ellos es el propio Walk-SAT pero añadiéndole el paso greedy con cierta probabilidad, que por lo general ese paso greedy es escoger cambiar el valor de la variable que menos se repita en cláusulas satisfechas.