

*“Dado que el tejido del Universo es de la mayor perfección y la obra del más sabio Creador, nada en absoluto tiene lugar en el Universo sin que una regla de máximo o mínimo aparezca”*

Leonard Euler

## Marco teórico

Un problema de optimización es un tipo de problema matemático y computacional que consiste en encontrar la mejor solución entre un conjunto de soluciones factibles, maximizando o minimizando una función objetivo. Si las variables que intervienen en este son enteras, entonces se dice que es un problema de optimización en enteros.

Existen una gran cantidad de problemas de este tipo que pertenecen a la categoría de NP-Complejos, haciendo que sean decidibles por una máquina de Turing no determinista en un tiempo polinomial, pero todavía no se han encontrado algoritmos que lo solucionen asintóticamente en dicha complejidad. Para muchos de estos problemas se suelen usar estrategias como algoritmos de aproximación, los cuales dan una respuesta aproximada en función de la solución óptima o del tamaño de la entrada, o mediante metaheurísticas, que consiste en utilizar técnicas de búsqueda que guían la exploración del espacio de soluciones de manera más eficiente que los métodos tradicionales.

Sin embargo, no es descabellado buscar o utilizar algoritmos que proporcionen soluciones exactas, ya que, si bien los algoritmos exactos pueden tener un tiempo de ejecución exponencial en el peor caso, eso no significa que lo sean también para el caso promedio o instancias específicas de dicho problema. La eficiencia de los métodos computacionales utilizados para esto depende, en primer lugar, de cómo se construyen los modelos y, en segundo lugar, de los métodos utilizados.

Para esto podemos seguir tres estrategias diferentes: La satisfacción booleana, la programación lineal en enteros mixta y la programación de satisfacción de restricciones

## Satisfacción booleana

Centrándonos en la optimización discreta, no siempre es obvio, con un solo problema, hasta qué punto se utiliza la lógica o hasta qué punto se utilizan métodos más tradicionales. Sin embargo, hay una gran ventaja en poder moverse entre los dos y reconocer las relaciones entre ellos. En este sentido, la optimización discreta (generalmente conocida como programación entera) y la lógica son simbióticas.

En este contexto, el problema de satisfacibilidad booleana (SAT) emerge como un caso paradigmático donde la lógica y la optimización discreta se cruzan. El Teorema de Cook, propuesto por Stephen Cook en 1971, es un hito fundamental en la teoría de la complejidad computacional, pues plantea que todo problema NP es reducible a un problema SAT. Este problema consiste en saber dado una fórmula booleana, si existe una interpretación de la misma tal que esta sea verdadera.

Como forma general, todo problema SAT se representa en su forma normal conjuntiva(CNF) debido a las ventajas que ésta posee, y todos los cuantificadores se sitúan al principio de la expresión (prenex normal form). Es necesario señalar que toda expresión válida de la lógica de predicados puede llevarse a dicha forma. Por eso, muchos algoritmos toman CNF como entrada. Entre los algoritmos más utilizados para resolver de forma completa un SAT están el algoritmo de Davis-Putnam(DP) y Davis-Logemann-Loveland (DLL/DPLL).

El algoritmo de Davis-Putnam es un precursor de los algoritmos modernos para resolver SAT, el cual utiliza el principio de resolución. Sea una instancia de SAT en CNF, sea  $p$  una variable proposicional y sean  $C_1 = p \vee Q_1$  y  $C_2 = \neg p \vee Q_2$  cláusulas del problema, con  $Q_1$  y  $Q_2$  disyunciones de literales. Como  $(p=1) \sqcap Q_2$  y  $(p=0) \sqcap Q_1$  se puede deducir  $Q_1 \vee Q_2$ . Al aplicar iterativamente resolución, podemos deducir posibles valores de variables o una contradicción. En este último caso, se dice que el problema es insatisfacible.

Veamos el siguiente ejemplo: La siguiente formula sera satisfacible:

$$((a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee c) \wedge (\neg a \vee \neg c))$$

Al aplicar la regla de resolución entre las primeras dos cláusulas obtenemos la nueva restricción  $(a \vee a)$ , la cual es lógicamente equivalente a  $(a)$ . Si aplicamos nuevamente resolución entre esta cláusula y las dos ultimas, deducimos  $(c)$  y  $(\neg c)$ . Si aplicamos resolución somos capaces de ver que llegamos a un absurdo, por lo que la formula nunca será satisfacible.

Por otra parte, Davis-Logemann-Loveland (DLL/DPLL) se centra en asignar iterativamente valores a las variables y deshaciendo dichas asignaciones en caso de conflicto. Se basa en tres hechos: todo literal puro (se dice puro si el literal opuesto no esta presente) es asignado como cierto, si una cláusula tiene todos sus literales negados excepto uno este ultimo debe ser cierto, y si todos los literales de una cláusula están negados, entonces la asignación hecha hasta dicho punto es falsa. Las etapas del algoritmo son: preprocesamiento (donde se asignan todos los literales puros), ramificación (donde se asigna un valor a un literal), propagación unitaria (llamado en ingles Unit Propagation, en esta etapa se asignan valores a aquellos literales cuyo valor se puede deducir), análisis de conflicto (donde se busca agregar restricciones adicionales basada en la asignación parcial en caso de hallar una contradicción) y retroceso (comúnmente llamado Backtracking, deshace asignaciones hechas en caso de darse una contradicción, para así explorar nuevos casos).

Si bien estos algoritmos son exponenciales de forma general, para ciertas familias de problemas su complejidad puede reducirse a tiempo polinomial. Aquellos SAT donde cada cláusula tiene a lo sumo 2 literales (los llamados 2-SAT) es soluble en tiempo polinomial. Otro ejemplo es Horn-SAT, que son aquellos donde cada cláusula tiene, a lo sumo, un literal positivo (las llamadas cláusulas de Horn). Dichas característica permite acelerar el proceso de inferencia, permitiendo definir con mayor facilidad si a partir de las restricciones se puede deducir una fórmula lógica. Muchos lenguajes declarativos como Prolog restringen a poner todas sus restricciones como cláusulas de Horn.

Todo esto permite resaltar la gran importancia que cobra la lógica en este tipo de problemas, pues es la que permite deducir enunciados a partir de otros en función de las reglas de deducción que lo conforman. Más específicamente la lógica proposicional y la lógica de predicados son consistentes y completos, por lo que proporcionan un marco teórico robusto para abordar los problemas SAT.

Se dice que un sistema es consistente si no se pueden derivar contradicciones dentro de él, es decir, no se puede demostrar que un enunciado sea verdadero y falso simultáneamente. Un sistema es completo si se puede deducir la veracidad o falsedad de cualquier enunciado que sea verdadero en el modelo del sistema. La consistencia y la completitud son propiedades fundamentales en la lógica y la teoría de sistemas formales. Sin embargo, estas dos propiedades no siempre pueden coexistir en todos los sistemas. Este dilema es especialmente relevante en el contexto de la teoría de Gödel, que establece que en cualquier sistema formal suficientemente poderoso (como la aritmética), si el sistema es consistente, no puede ser completo. Esto significa que existen enunciados que son verdaderos pero que no pueden ser probados dentro del sistema.

A la hora de encarar un problema de optimización usando lógica de predicados, es necesario añadir funciones, constantes y reglas que la involucren. Aunque la aritmética completa sea no decidible, hay "teorías" más pequeñas dentro de ella que son decidibles. Entre estas están la aritmética sin multiplicación y la teoría de orden lineal denso. Estas bastan para resolver cualquier modelo de optimización lineal.

Veamos lo anteriormente planteado en el siguiente ejemplo:

**Maximizar:**

$$2x_1 + 3x_2 - x_3$$

*subject to:*

$$x_1 + x_2 \leq 3$$

$$-x_1 + 2x_3 \geq -2$$

$$-2x_1 + x_2 - x_3 = 0$$

$$x_1, x_2, x_3 \in \mathbb{R}$$

Esto planteado en lógica de predicados sería:

$$\exists z, x_1, x_2, x_3 (z - 2x_1 - 3x_2 + x_3 = 0 \wedge x_1 + x_2 - x_3 = 0 \wedge -2x_1 + x_2 - x_3 = 0 \wedge x_1 \geq 0 \wedge x_2 \geq 0 \wedge x_3 \geq 0)$$

Luego, podríamos despejar  $x_3$  en la cuarta restricción y sustituir en el resto, eliminando así una variable del problema.

$$\exists z, x_1, x_2 (z - 2x_1 - 3x_2 + (-2x_1 + x_2) = 0 \wedge x_1 + x_2 - (-2x_1 + x_2) = 0 \wedge -2x_1 + x_2 = 0 \wedge x_1 \geq 0 \wedge x_2 \geq 0)$$

De forma homóloga, podríamos despejar la variable  $x_2$  en la primera restricción:

$$\exists z, x_1 (z - 2x_1 + 2(-2x_1 + x_2) = 0 \wedge -5x_1 + 2(z - 2x_1) = 0 \wedge x_1 \geq 0 \wedge z - 2x_1 \geq 0 \wedge -2x_1 + z \geq 0)$$

Luego despejemos la variable  $x_1$  en todas las restricciones

$$\exists z (\exists x_1 (z - 3x_1 = 0 \wedge -5x_1 + 2(z - 2x_1) = 0 \wedge x_1 \geq 0 \wedge z - 2x_1 \geq 0 \wedge -2x_1 + z \geq 0))$$

Notar que aquí deducimos que:

$$\exists z (z - 3z + 9 = 0 \wedge z - 3z + 4 = 0 \wedge z - 3z + 8 = 0 \wedge 0 \leq z \leq 9 \wedge 0 \leq z \leq 4 \wedge 0 \leq z \leq 8)$$

Concluyendo que  $-2 \leq z \leq 8$ . Y como el objetivo es maximizar. Se toma  $z = 8$ . De aquí vemos que  $0 \leq x_1 \leq 1$ . Que tomando a  $x_1 = 1$  nos queda que  $x_2 = 2$  y  $x_3 = 0$

El procedimiento anteriormente planteado teóricamente demuestra muchas propiedades de este tipo de problemas de optimización. Sin embargo, es ineficiente en comparación con los otros métodos utilizados actualmente.

## Programación en enteros

La programación en enteros es un conjunto de herramientas tradicionales hechas para resolver el problema: Cual es el máximo/mínimo que alcanza la función  $c^T x + d^T y$  sujeto a las restricciones:  $Ax \leq p$ ,  $By \leq q$ ,  $x \geq 0$ ,  $y \geq 0$ ,  $x \in \mathbb{R}^n$ ,  $y \in \mathbb{Z}^m$ .

Se dice que es programación entera pura si todas las variables son enteras, de lo contrario se clasifica en programación entera mixta (MIP). Un caso específico de la programación entera pura es cuando todas las variables involucradas son binarias. Todo problema de programación entera pura puede ser reformulado como un problema de programación entera binaria, aunque esto significa aumentar el número de variables del problema.

Como se puede ver, este paradigma es una extensión de la programación lineal, y de forma general, resolver un problema de programación en enteros pasa por hallar el óptimo de un problema lineal mediante métodos como el algoritmo simplex.

Un ejemplo de lo anteriormente planteado es el algoritmo de ramificación y acotación. El primer paso de este método es eliminar el requerimiento de que las variables sean enteras. Esto permite trabajar con una relajación del conjunto de soluciones factibles del problema. Una vez obtenida la solución óptima  $x^*$ , si  $x^* \in \mathbb{Z}^n$  entonces el óptimo ha sido

encontrado. Si no,  $\exists x_i$  componente de la solución tal que  $x_i = n + p$ , tal que  $n \in \mathbb{Z}$  y  $p \in [0, 1)$ . Luego podemos dividir el conjunto de soluciones factibles en dos: aquellos donde  $x_i \leq n$  y aquellos donde  $x_i \geq n + 1$ . A continuación, podríamos resolver dos ramas del mismo problema original tomando alguna de las dos restricciones, descartando la solución  $x^*$ . Finalmente, el óptimo será el menor (si se está minimizando; de lo contrario, será el mayor) de los óptimos de ambas ramificaciones. Veámoslo en el siguiente ejemplo:

Maximizar

$$x_1 + x_2$$

sujeto a:

$$2x_1 + 2x_2 \geq 3$$

$$-2x_1 + 2x_2 \leq 3$$

$$4x_1 + 2x_2 \leq 19$$

$$x_1, x_2 \geq 0$$

$$x_1, x_2 \in \mathbb{Z}$$

Solución óptima:  $x_1 = 2.67, x_2 = 4.16, Objective = 6.83$

Entonces, el problema se divide en dos subproblemas distintos: uno con la restricción extra  $x_1 \leq 2$  (Caso 1) y otro con la restricción extra  $x_1 \geq 3$  (Caso 2).

Para el caso 1 la solución óptima es:  $x_1 = 2, x_2 = 3.5, Objective = 5.5$

Para el caso 2 esta es:  $x_1 = 2, x_2 = 3.5, Objective = 6.5$ .

En este caso se puede seguir ramificando por ambas ramas. Específicamente, si ramificamos el caso 2, este se dividiría en el caso donde  $x_2 \geq 4$  y el caso donde  $x_2 \leq 3$ .

Finalmente. Tras otras dos ramificaciones se puede llegar a que el óptimo es  $x_1 = 3, x_2 = 3, Objective = 6$ .

Otros métodos son los llamados planos cortantes, que consisten en buscar una parte de la menor envoltura convexa que contiene todas las soluciones enteras. Un ejemplo de planos cortantes es el corte de Gomory. Sea  $ax \leq b$  una de las restricciones del problema, entonces toda solución entera  $x$  cumple que  $[a]x \leq [b]$ , los valores que cumplen la primera condición pero no la segunda son aquellos en los que  $\frac{a}{b}x < \frac{a}{b}b$ . Por tanto, siempre será válido adicionar la restricción  $\frac{a}{b}x \geq \frac{a}{b}b$ . Esto permite seguir trabajando con problemas relajados descartando soluciones no enteras.

Para ambas estrategias de solución, es difícil encontrar formas generales de aplicarlas sistemáticamente, debido al indeterminado número de veces que se necesitan aplicar para llegar a la solución óptima. Dicho número se conoce como rango de Chvátal, y es una buena forma de medir la complejidad de estos métodos de solución sobre un modelo en específico.

Es fundamental identificar qué problemas pueden ser representados como problemas de programación entera. Para ello, resulta interesante explorar cómo diversas restricciones de la lógica de predicados pueden ser modeladas en este contexto. Al comprender esta relación, podemos traducir enunciados lógicos complejos en formulaciones matemáticas que se pueden resolver mediante técnicas de optimización discreta, ampliando así el alcance de los problemas que podemos abordar.

Supongamos que queremos introducir las siguientes restricciones:

$$\sum_j a_{ij}x_j \leq b_i \quad \delta_i = 1$$

$$\delta_i = 1 \quad \sum_j a_{ij}x_j \leq b_i$$

Si estas restricciones se logran, se podría saber cuántas restricciones se cumplen en un

modelo, haciendo bisección entre una restricción y una variable binaria.

Para la primera fórmula, al aplicar contrarrecíproco, se quiere que  $\sum_j a_{ij}x_j > b_i$ , pero si  $\exists i: \sum_j a_{ij}x_j \geq m$ , entonces se puede crear la restricción  $\sum_j a_{ij}x_j \geq b_i + \epsilon + (m - b - \epsilon)\delta_i$ . De esa forma, si  $\delta_i = 1$  es una restricción redundante, si  $\delta_i = 0$  entonces fuerza a incumplir la restricción objetivo.

Para la segunda, si  $\exists i: \sum_j a_{ij}x_j \leq M$  entonces se puede introducir la restricción  $\sum_j a_{ij}x_j \leq M - (M - b)\delta_i$ .

Luego, si no existieran dichos valores, entonces se dice que el problema no es MIP representable. Un ejemplo de esto sería:  $x = 0 \vee y = 0$ .

Una vez haciendo biyección entre variables lógicas y restricciones lineales, se pueden hacer operaciones lógicas elementales:

$$\begin{aligned} \delta_1 \vee \delta_2 : \delta_1 + \delta_2 &\geq 1 \\ \delta_1 \wedge \delta_2 : \delta_1 + \delta_2 &= 2 \\ \delta_1 \rightarrow \delta_2 : \delta_1 &\leq \delta_2 \\ \delta_1 \oplus \delta_2 : \delta_1 &\neq \delta_2 \end{aligned}$$

De esta forma se podrían modelar problemas escritos en formas normales conjuntivas y disyuntivas. Se puede demostrar que una formulación basada en forma normal disyuntiva (siempre que se utilicen los mismos límites  $M$  y  $m$  para cada restricción) siempre será al menos tan estricta, y a veces más estricta, que una formulación basada en CNF. debido a que requerirá más variables 0-1 que la formulación DNF, ya que necesitamos  $r$  (o estrictamente  $\lceil \log_2 r \rceil$ ) variables 0-1 para modelar cada disyunción. Si tenemos una conjunción de  $n$  tales disyunciones, entonces necesitaremos un total de  $m$  (o  $\lceil \log_2 r \rceil n$ ) 0-1 variables, mientras que en DNF una conjunción de  $m$  disyunciones podría modelarse con  $m$  variables 0-1. En la práctica, pueden ser posibles simplificaciones sustanciales en cualquiera de los dos tipos de formulación, y el tamaño de la representación CNF o DNF dependerá del problema. La hermeticidad de la IP resultante suele ser de mayor importancia que la compacidad del modelo en términos de número de variables (y restricciones).

## Programación de satisfacción de restricciones

Una forma de analizar un problema de optimización es como un problema de satisfacción de restricciones, que consiste en una tupla  $(V, D, C)$  donde  $V$  es un conjunto de variables,  $D = \{D_v | v \in V\}$  es el conjunto de los conjuntos de los posibles valores que pueden tomar cada variable, y  $C$  es un conjunto finito de restricciones de la forma  $(R_i, S_i)$ , con  $S_i$  es un subconjunto ordenado de  $V$  y  $R_i$  es una relación de tamaño  $|S_i|$ . Una solución es una asignación a cada variable que pertenece a  $V$  con uno de sus correspondientes valores en  $D$  tal que se cumplan todas las restricciones en  $C$ . Cuando el número de soluciones es exponencial, se dice que es un problema combinatorio, que es donde entran los problemas relacionados con la optimización.

La programación de satisfacción de restricciones (CSP) es aquella especializada en resolver este tipo de problemas. Dado que no se basa en la rica metodología computacional de la programación lineal, carece de la sofisticación matemática que ésta y la programación en enteros presenta. Sin embargo, es mucho más rico en sus capacidades de modelado y más flexible en sus estrategias de solución. No obstante, algunas de las operaciones (por ejemplo, la ramificación) utilizadas son similares a las de IP, y tiene muchas características en común con los procedimientos de reducción que ahora se usan comúnmente para preprocesar modelos. Este enfoque no está concebido como un método de optimización propiamente, aunque se puede adaptar a él haciendo que el objetivo, con límites cada vez más estrictos, sea una restricción.

A diferencia de la programación en enteros, que restringe su modelado a expresiones lineales, En la programación por restricciones, los modelos suelen expresarse en forma de predicados, que si bien pudieran ser convertidos a modelos lineales, dicha conversión puede ser engorrosa. Dichos predicados suelen depender del software utilizado, y en muchos casos se da la oportunidad al usuario de definir predicados locales. Pero de forma general existen restricciones globales que suelen ser semánticamente redundantes y permiten filtrar el dominio de las variables.

La mayoría de los solvers recaen en la propagación de restricciones (constraint propagation) y se realiza mediante la comprobación de consistencia entre los valores de las variables. Este proceso implica analizar las restricciones que vinculan diferentes variables y ajustar sus dominios en consecuencia, lo que implica eliminar aquellos que violen alguna restricción. Entre las formas de comprobar consistencia está la consistencia de nodo, que reduce el dominio de una variable a aquellos valores que cumplen con todas las restricciones unarias.

También se habla de la consistencia de arco, centrada en eliminar aquellos valores  $a$  de una variable  $x$  si no existen valores  $b$  de una variable  $y$  tales que  $(a,b)$  satisfagan a todas las restricciones entre  $x$  y  $y$ . Uno de los algoritmos más utilizados para comprobar consistencia de arco es el algoritmo AC-3, el cual guarda todos los pares ordenados de variables en una cola. Luego saca iterativamente cada uno de estos pares  $\langle x,y \rangle$  hasta que la cola se quede vacía, y comprueba la consistencia de arco para cada posible valor de  $x$ . Si un valor no cumple la consistencia de arcos, este valor es eliminado del dominio de  $x$ , y todos los pares de variables de la forma  $\langle z,x \rangle$  son reinsertados en la cola. El algoritmo tiene una complejidad de tiempo en el peor de los casos de  $O(ed^3)$ , donde  $e$  es la cantidad de pares y  $d$  es el tamaño de dominio más grande. Tras aplicar la consistencia de arco, pueden surgir tres posibles escenarios: si todos los dominios de las variables quedan con exactamente 1 valor (en cuyo caso tenemos la asignación factible), si un dominio queda vacío (en cuyo caso ocurriría una contradicción y se debe hacer backtrack en una asignación) o si al menos un dominio queda con más de un posible valor, en cuyo caso se le debe asignar un valor y volver a realizar consistencia de arco.

Otras formas de consistencia existentes son la consistencia de camino y la  $k$ -consistencia. La consistencia de camino considera no solo las restricciones binarias entre pares de variables, sino también las relaciones a través de secuencias más largas de variables. Aquí,  $u$  es un valor consistente de  $x$  si para todo  $y$  existe un  $w$  tal que dado cualquier secuencia de variables  $a_1, a_2, \dots, a_n$ , con  $a_1 = x$  y  $a_n = y$  tenga la secuencia de valores  $v_1, v_2, \dots, v_n$  con  $v_1 = u$  y  $v_n = w$  de forma que el par  $\langle v_i, v_{i+1} \rangle$  cumpla con todas las restricciones binarias entre  $a_i$  y  $a_{i+1}$ , con  $1 \leq i \leq n$ . Si bien la aplicación de la consistencia de camino garantiza un mayor nivel de consistencia que la consistencia del arco, todavía no es suficiente para resolver CSP en general. Esto significa que garantizando dicha consistencia, no todas las asignaciones garantizadas por esta son necesariamente soluciones factibles. Por otra parte, la  $k$ -consistencia, se logra al garantizar que cualquier asignación válida de valores a  $k-1$  variables garantiza la posibilidad de asignación de un valor a otra cualquier otra variable. Se dice que se es fuertemente  $k$ -consistente si para todo  $j < k$  se es  $j$ -consistente. Ambos tipos de consistencias son bastante costosos computacionalmente por lo que no es muy utilizado en la práctica en comparación con la consistencia de arco.

Ahora, si se desea optimizar usando CSP, una forma de lograrlo es hacer búsqueda binaria sobre la función objetivo. Sea  $f(x)$  la función objetivo a maximizar y sean  $m$  y  $M$  tales que  $\forall x : m \leq f(x) \leq M$ . Esto permite hacer un problema de factibilidad adicionando la restricción  $f(x) \geq \frac{M+m}{2}$ . Si el problema es satisfacible con  $f(x) = m'$  entonces se puede resolver el modelo nuevamente, pero esta vez con la restricción  $f(x) \geq \frac{M+m'}{2}$ . En caso contrario se puede volver a realizar la búsqueda con la restricción  $f(x) \geq \frac{M'+m}{2}$  con  $M' = \frac{M+m'}{2}$ . El caso de parada es cuando  $M = m$ , haciendo que la respuesta final sea la última solución encontrada.