Problemas de satisfacción de restricciones

Miguel Romero
Estudiante de Doctorado en Ciencias de la Computación, Universidad de Chile. Ingeniero Civil Matemático y Magíster en Ciencias mención Computación, Universidad de Chile. Áreas de interés: Bases de Datos, Lógica para la Computación.
mromero@dcc.uchile.cl

Un problema de *satisfacción de restricciones* (o CSP, por sus siglas en inglés), consiste en buscar una asignación de valores para un conjunto de variables que respete ciertas restricciones. Estos tipos de problemas son fundamentales en Ciencia de la Computación, ya que permiten modelar problemas de distintas áreas, como Inteligencia Artificial, Bases de Datos y Teoría de Grafos, por nombrar algunas.

En los años setenta, partiendo con el trabajo de Montanari [30], la comunidad de Inteligencia Artificial comenzó una intensa investigación en esta clase de problemas. El área de CSP ocupa un lugar prominente en Inteligencia Artificial y ha sido fundamental en el estudio de satisfacibilidad booleana, *scheduling*, razonamiento temporal y visión computacional, entre otros [11, 18, 29]. En general, es NP-Completo resolver un CSP, por lo que los esfuerzos se han enfocado en buscar casos tratables y heurísticas para la búsqueda de soluciones [11, 18].

Por otra parte, a partir de los influyentes trabajos de Feder, Kolaitis y Vardi [15, 27, 28] a finales de los noventa, se inició un estudio sistemático de CSP, desde un punto de vista teórico. En este enfoque, hay un énfasis en comprender la diferencia entre casos tratables y no tratables, utilizando herramientas de complejidad computacional y NP-completitud. La pregunta básica que ha guiado esta línea de investigación ha sido la siguiente:

¿Qué tipos de CSP son tratables y qué tipos no lo son?

Durante el último tiempo, ha habido avances importantes y se han establecido conexiones interesantes con otras áreas como Bases de Datos, Lógica, Álgebra Universal y Complejidad computacional. Es importante recalcar la relevancia de este enfoque. Primero, nos ayuda a entender conceptualmente qué es lo que hace a un CSP admitir un algoritmo eficiente. Segundo, en términos prácticos, conocer los límites entre lo tratable y lo intratable, nos ayuda a desarrollar mejores algoritmos y heurísticas.

En este artículo daremos una breve mirada a los aspectos teóricos de satisfacción de restricciones. Introduciremos el concepto de CSP y revisaremos brevemente los avances más importantes en los últimos quince años.

FORMULANDO UN CSP

Pensemos en el problema de 2-coloración: nuestro input es un grafo G y queremos saber si cada nodo puede ser pintado con un color entre {Azul, Rojo}, de manera que nodos adyacentes reciben colores diferentes. Podemos formular este problema como un CSP de la siguiente manera:

1. Nuestras variables son los nodos de G.
2. El espacio de valores posibles para las variables es {Azul, Rojo}.
3. Por cada arco (x,y), tenemos una restricción indicando que x e y reciben valores distintos.

Como podemos ver, cada restricción determina los valores que pueden tomar cierto conjunto de variables. La manera estándar de escribir una restricción, es indicar primero las k variables que se ven afectadas por ella y, luego, una relación k-aria, indicando los posibles valores que pueden tomar estas variables. En el caso de 2-coloración, por cada arco (x,y), tendremos una restricción de la forma ((x,y), Neq), donde Neq indica los pares de colores distintos:

Neq={(Azul, Rojo), (Rojo, Azul)}

Luego, cada asignación de valores a las variables que respeta las restricciones, efectivamente corresponde a un 2-coloreo.

Consideremos ahora el famoso problema 3-SAT: Nuestro input es una fórmula proposicional en 3CNF (conjunción de cláusulas con 3 literales, ver ejemplo en Figura 1) y debemos decidir si es satisfacible. Nuestras variables son simplemente las variables de la fórmula, los valores posibles son {V,F}, y cada cláusula genera una restricción. Por ejemplo, si tomamos la primera cláusula ($\neg x \vee \neg y \vee z$) de la fórmula φ en la Figura 1, entonces tendremos la restricción ((x,y,z), R), donde R indica los posibles valores que hacen verdadera esta cláusula, es decir, todos los triples excepto (V,V,F), como muestra la primera restricción de la Figura.

Nuevamente es claro que una asignación que respeta todas las restricciones, corresponde a una asignación que hace verdadera a la fórmula. A modo de ejemplo, consideremos la asignación x=V, y=F, z=F, u=F, v=V, para la fórmula φ de la Figura 1. Notemos que la primera restricción afecta a ($x, y,$ z). Si reemplazamos cada variable por el valor asignado,

$\varphi = (\neg x \vee \neg y \vee z) \wedge (x \vee z \vee u) \wedge (\neg z \vee y \vee u) \wedge (\neg u \vee \neg v \vee y) \wedge (\neg x \vee \neg z \vee \neg v)$

Variables=$\{x, y, z, u, v\}$
Dominio={V,F}
Restricciones

$x\ y\ z$	$x\ z\ u$	$z\ y\ u$	$u\ v\ y$	$x\ z\ v$
F F F	F F V	F F F	F F F	F F F
F F V	F V F	F F V	F F V	F F V
F V F	F V V	F V F	F V F	F V F
F V V	V F F	F V V	F V V	F V V
V F F	V F V	V F V	V F F	V F F
V F V	V V F	V V F	V F V	V F V
V V V	V V V	V V V	V V V	V V F

Figura 1 • Formulación de 3-SAT como un CSP.

obtenemos la tupla (V,F,F), la cual es una tupla permitida, como muestra el color azul en la Figura. Lo mismo sucede con todas las restricciones, lo que implica que nuestra asignación es una solución. Es fácil verificar además que esta asignación efectivamente satisface a la fórmula.

Ahora estamos en condiciones de dar una formulación más general. Una instancia de un CSP queda definida por tres componentes:

1. Un conjunto finito de variables V.
2. Un dominio finito de valores D.
3. Un conjunto finito de restricciones C. Cada restricción es de la forma $((x_1, x_2, \ldots, x_k), R)$, donde $\{x_1, x_2, \ldots, x_k\}$ son variables en V y R es una relación k-aria sobre D. Como ya mencionamos, cada restricción determina los posibles valores que pueden tomar ciertas variables.

En un CSP nuestra instancia es un triple (V,D,C) y el objetivo es decidir si existe una *solución*, es decir, una asignación h: $V \rightarrow D$ de valores a las variables, que respeta todas las restricciones. Esto último significa que para cada restricción $((x_1, x_2, \ldots, x_k), R)$ en C, tenemos que $(h(x_1), h(x_2), \ldots, h(x_k))$ está en la relación R.

Observemos que en el caso de 2-coloración, nuestras instancias tienen sólo restricciones binarias y éstas sólo usan la relación Neq, mientras que en 3-SAT las restricciones son ternarias y usamos varios tipos de relaciones dependiendo de la forma de la cláusula. Por tanto, 2-coloración y 3-SAT son distintos tipos de CSP, en el sentido que las instancias tienen distinta estructura. Más aún, sus complejidades son radicalmente diferentes: mientras 2-coloración se puede resolver eficientemente mediante un algoritmo *greedy*, 3-SAT es NP-Completo [2].

Lo anterior ilustra el hecho de que satisfacción de restricciones es NP-Completo en general, pero en algunos casos, dependiendo de la estructura de las instancias, podemos encontrar una solución de manera eficiente.

CSP y homomorfismos

Uno de los principales aportes conceptuales del trabajo de Feder y Vardi [15] fue formular un CSP como un problema de homomorfismos entre estructuras. Esta formulación dio origen a un amplio estudio teórico y permitió establecer conexiones con otras áreas más abstractas. A continuación definimos la noción de estructura y homomorfismo.

Un *esquema* es un conjunto de símbolos de relación $R_1, \ldots, R_p$. Cada símbolo Ri tiene una aridad $k_i>0$ asociada. Una *estructura* A sobre el esquema $R_1, \ldots, R_p$, es básicamente un conjunto de relaciones $R_1^A, \ldots, R_p^A$ (de la aridad correspondiente), sobre cierto dominio finito dom(A). Es decir, la estructura A interpreta los símbolos de relación en dom(A).

Por ejemplo, cada grafo dirigido puede ser visto como una estructura, donde el dominio son los nodos y tenemos una relación binaria indicando cada arco. En el caso de un grafo no dirigido o grafo a secas, esta relación será simétrica. En este contexto, el esquema es un símbolo binario E. Similarmente, cada fórmula en 3CNF puede ser vista como una estructura. Notemos que (debido a que la disyunción conmuta) podemos asumir sin pérdida de generalidad que sólo tenemos cuatro tipos de cláusulas: $(x \vee y \vee z)$, $(\neg x \vee y \vee z)$, $(\neg x \vee \neg y \vee z)$ y $(\neg x \vee \neg y \vee \neg z)$. Por tanto, podemos considerar un esquema $T_1, \ldots, T_4$, en donde cada símbolo es ternario y cada uno representa cierto tipo de cláusula. De esta manera, el dominio de la estructura son las variables de la fórmula y tenemos 4 relaciones $T_1, \ldots, T_4$, las cuales indican los triples de variables que conforman cada tipo de cláusula. En la Figura 2, la estructura A^{φ} representa a φ.

Un *homomorfismo* entre dos estructuras (con el mismo esquema) A y B es un mapeo de dom(A) a dom(B), el cual preserva las relaciones de A. Formalmente, si el esquema es $R_1, \ldots, R_p$, entonces h: dom(A) → dom(B) es un homomorfismo, si para cada $1 \leq i \leq p$ y cada tupla $(t_1, \ldots, t_k)$ en R_i^A, se tiene que $(h(t_1), \ldots, h(t_k))$ está en R_i^B. Si existe homomorfismo de A a B, diremos que A es homomorfo a B.

Como ya podemos imaginar, existe una fuerte simbiosis entre homomorfismos y CSPs. De alguna forma, la característica de "preservar las relaciones" que posee un homomorfismo se alinea con "respetar las restricciones" en una solución de un CSP. Para ilustrar esto, consideremos el esquema E (.,.) para representar grafos y la estructura K_2 con dominio {Azul, Rojo} y relación E^{K2}={(Azul, Rojo),

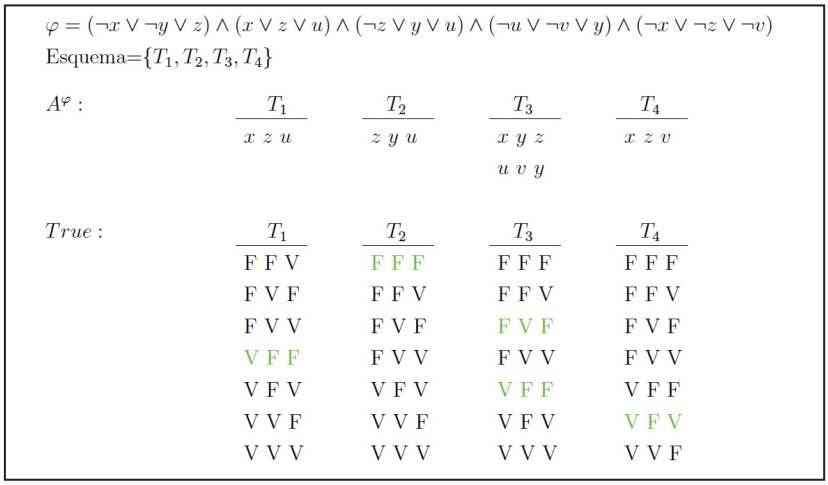

Figura 2 • El problema 3-SAT como existencia de homomorfismos.

(Rojo, Azul)}. Entonces es claro que un grafo G tiene un 2-coloreo si y sólo si G (visto como estructura) es homomorfo a K_2.

Este fenómeno no sólo se restringe a problemas de coloración, sino que es inherente a cualquier CSP. Para ver esto, consideremos una instancia genérica I = (V,D,C). Construiremos dos estructuras A(I) y B(I) como sigue:

1. En nuestro esquema tenemos un símbolo de relación por cada relación que aparece en alguna restricción de C.
2. El dominio de A(I) es el conjunto de variables V. Si tenemos una restricción $((x_1,\ldots, x_k), R)$ en C, entonces la tupla $(x_1,\ldots, x_k)$ está en $R^{A(I)}$.
3. El dominio de B(I) es el conjunto de valores D. Si R aparece en alguna restricción, entonces $R^{B(I)}$ es precisamente R.

Intuitivamente, A(I) codifica las variables y la interacción de los conjuntos de variables que aparecen restringidos. La estructura B(I) codifica los posibles valores que pueden tomar las variables restringidas. Luego, cada mapeo de A(I) a B(I) que respete las relaciones de A(I) (es decir, un homomorfismo), respeta a su vez las restricciones en I, y viceversa. Dicho de otra manera, existe un homomorfismo de A(I) a B(I) si y sólo si I tiene una solución.

Para ilustrar la construcción, consideremos el problema 3-SAT. En este caso, el esquema consta de cuatro relaciones ternarias $T_1,\ldots,T_4$, representando cada tipo de cláusulas, como vimos anteriormente. Para una fórmula φ, la estructura A(I) es la representación de la fórmula A^φ y B(I) es la estructura True, como muestra la Figura 2.

No es difícil ver que una asignación satisface φ si es un homomorfismo de A^φ a True. Por ejemplo, si consideramos nuevamente la asignación x=V, y=F, z=F, u=F, v=V, la cual hace verdadera a φ, podemos ver que define un homomorfismo. En efecto, la tupla (x,z,u), en la relación T_1, es mapeada a la tupla (V,F,F), la cual pertenece a T_1 en True, como podemos apreciar de color verde en la figura. Las demás tuplas en A^φ también son respetadas, por tanto tenemos un homomorfismo.

Lo anterior nos dice que podemos identificar cualquier CSP con un problema de existencia de homomorfismo entre estructuras. Por tanto, de ahora en adelante pensaremos que toda instancia a un CSP es un par de estructuras A y B.

Tipos de CSPs

Para cada CSP, asumiremos que tenemos un esquema fijo a priori, y que las instancias están definidas sobre este esquema (por ejemplo, el esquema de los grafos o de las fórmulas). Cada tipo de CSP queda determinado por la forma de sus instancias, es decir la forma de las estructuras A y B.

Para formalizar esto, supongamos que tenemos dos conjuntos de estructuras C y D, potencialmente infinitos, sobre cierto esquema. Denotaremos por CSP(C,D) al CSP cuyas instancias A y B cumplen que A está en C y B está en D. Por ejemplo, 2-coloración es precisamente CSP(Grafos, $\{K_2\}$), donde Grafos es el conjunto de todos los grafos. A su vez, 3-SAT es equivalente a CSP(Fórmulas, {True}), donde Fórmulas son todas las estructuras que representan fórmulas en 3CNF.

Ahora podemos formular la pregunta fundamental que ha motivado todo este estudio:

¿Para qué conjuntos C y D, CSP(C, D) admite un algoritmo polinomial?

En la literatura, a este tipo de conjuntos C y D se les llama “islas tratables”. Como podemos imaginar, ésta es una pregunta bastante ambiciosa y aún no se tiene respuesta para ella. Debido a esto, la investigación se ha enfocado en casos particulares [7, 21].

Un caso interesante, el cual estudiaremos en las siguientes secciones, es tomar D como el conjunto de todas las estructuras. Es decir, para un conjunto C, estudiaremos el problema CSP(C,-), en donde las instancias A y B, cumplen que A está en C, y no hay restricción sobre B. A continuación, abordaremos la siguiente pregunta: ¿para qué conjuntos C, CSP(C,-) admite un algoritmo eficiente? Como veremos más adelante, esta interrogante tiene importantes aplicaciones en teoría de Bases de Datos.

CASOS TRATABLES: ÁRBOLES

Gran parte de los CSPs tratables se basan

en una simple idea algorítmica. Supongamos que tenemos dos grafos dirigidos T y B. Asumamos además que T tiene forma de árbol, como vemos en el ejemplo de la Figura 3. Queremos saber si existe un homomorfismo de T a B.

Para esto, construiremos para cada nodo t de T, un conjunto B_t de nodos de B. Intuitivamente, cada B_t indicará los posibles valores que puede tomar la variable t, en una solución parcial acotada al subárbol "debajo" de t. Más precisamente, cada B_t contiene exactamente los nodos b de B tal que existe un homomorfismo de T_t (el subárbol de T enraizado en t) a B, el cual mapea t a b. Si somos capaces de computar estos conjuntos, nuestro problema está resuelto: que exista homomorfismo de T a B es equivalente a que B_r sea no vacío, donde r es la raíz de T.

La observación clave es que, debido a que T tiene forma de árbol, es posible calcular los conjuntos B_t's eficientemente. Para esto, comenzamos por las hojas y vamos computando recursivamente B_t hasta llegar a la raíz de la siguiente manera:

1. Para cada hoja h de T, B_h contiene todos los nodos de B.
2. Si t es un nodo interno y t_1,…, t_r son sus hijos, entonces para determinar si un nodo b está en B_t, basta ver que existan nodos b1,…,br contenidos en B_{t1},…,B_{tr}, respectivamente, tal que para cada $1 \le i \le r$, hay un arco entre b y b_i, en la misma dirección que el arco entre t y t_i (y así respetar este arco o restricción). Por ejemplo, para el nodo t de la Figura 3, un nodo b estará en B_t, si existen b_1, b_2, b_3, cada uno en B_{t1}, B_{t2}, B_{t3}, respectivamente, tal que los arcos (b,b_1), (b_2,b) y (b,b_3) están en B.

Observemos que cada vez que visitamos un nodo interno t y verificamos si b está en B_t, nos basta visitar cada nodo en B_{t1},…,B_{tr} una sola vez, por tanto podemos computar Bt en a lo más $O(|B|^2 H_t)$, donde H_t es la cantidad de hijos de t. Lo anterior nos dice que nuestro algoritmo toma tiempo a lo más $O(|B|^2|T|)$. Luego, la clase Trees, de todos los grafos dirigidos con forma de árbol, es una isla tratable.

Esta idea básica ha sido generalizada a estructuras arbitrarias y más aún a estructuras que tienen una forma "parecida" a un árbol, utilizando el concepto de *treewidth* [10, 17]. El treewidth de una estructura mide qué tanto se parece ésta a un árbol: mientras más pequeño el treewidth, más parecida a un árbol. Por ejemplo, todos los árboles tienen treewidth 1.

Para finalizar, enunciamos el resultado que describe las islas tratables definidas por la noción de treewidth:

Teorema [10, 17]. Sea $k \ge 1$ y C un conjunto de estructuras. Si cada estructura en C tiene treewidth a lo más k, entonces CSP(C,-) se puede resolver en tiempo polinomial.

Por tanto, toda clase C con "treewidth acotado" es una isla tratable. Un caso particular es la clase Trees, descrita previamente.

Juegos de fichas

El resultado anterior, que involucra la noción de treewidth, puede ser explicado conectando CSP con otros conceptos como definibilidad en Datalog, test de k-consistencia o lógicas con variables finitas [12]. Otra forma interesante de entender estos resultados es mediante ciertos tipos de juegos combinatoriales llamados *juegos de fichas* [12, 26, 28]. Como veremos en la siguiente sección, estos juegos nos permitirán definir nuevas islas tratables.

Para empezar, definiremos un juego sobre dos grafos dirigidos A y B. En este juego, hay dos jugadores: *Spoiler* y *Duplicator*. Una posición del juego es un par (a,b) en dom(A)×dom(B). En el primer round, Spoiler escoge un elemento a_0 de A y Duplicator responde con un elemento b_0 en B. Después de esto, la posición del juego es (a_0, b_0). En los rounds subsiguientes, si la posición actual es (a,b), entonces la siguiente posición (a',b') queda determinada como sigue: Spoiler escoge un vecino a' de a y Duplicator res-

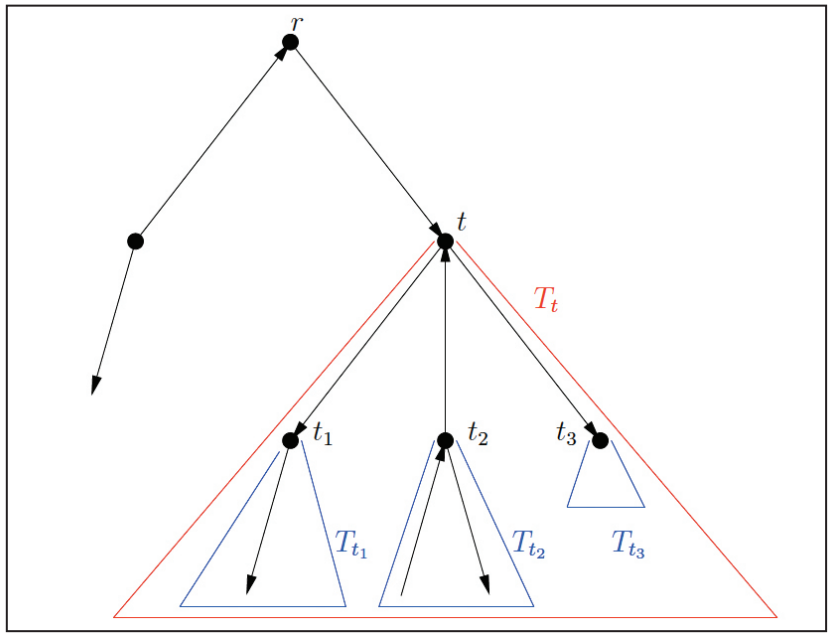

Figura 3 • El árbol T y algunos de sus subárboles.

ponde con un vecino b' de b, de manera que la dirección del arco entre a y a', es la misma que la del arco entre b y b'. Por ejemplo, si (a',a) está en A, entonces (b',b) debe estar en B. Si Duplicator tiene una estrategia que le permite jugar para siempre con Spoiler, sin importar como juegue este último, entonces decimos que Duplicator gana este juego sobre A y B.

Supongamos por un momento que A es un árbol, como el de la Figura 3. No es difícil ver que, si existe un homomorfismo h de A a B, entonces Duplicator gana este juego sobre A y B (de hecho, esto es cierto incluso para un A arbitrario). En efecto, en cada round, si Spoiler juega un elemento a, entonces Duplicator responde con h(a). Como h respeta la dirección de los arcos, esto siempre será una jugada válida para Duplicator. Por otra parte, supongamos que Duplicator gana este juego sobre A y B. Tampoco es difícil ver que esto implica la existencia de un homomorfismo de A a B. Intuitivamente, debemos "simular" una partida de este juego y hacer que Spoiler juegue desde la raíz hacia las hojas (escogiendo un nodo y luego todos sus hijos, continuando recursivamente). Las respuestas dadas por Duplicator en esta partida (cuando éste juega con su estrategia ganadora) constituirán el homomorfismo. Como A es un árbol no hay conflictos en los valores asignados y el homomorfismo queda bien definido. Por tanto, cuando A es un árbol, verificar si Duplicator gana este juego entre A y B es un método correcto y completo para existencia de homomorfismos [12].

Como podemos sospechar, la definición original de este juego, introducido por Vardi y Kolaitis en 1990 [26], llamado el *juego existencial de k fichas*, es más general y aplica para estructuras arbitrarias. El resultado anterior aplica para la definición original de la siguiente manera: si A tiene treewidth a lo más k, entonces decidir acaso Duplicator gana el juego existencial con k+1 fichas sobre A y B es un método correcto y completo para existencia de homomorfismos.

Pero aún falta lo más importante: ¿cómo decidimos si Duplicator puede ganar el juego con k fichas? Afortunadamente, esto se puede hacer en tiempo polinomial en A y B (k está fijo) [12, 25]. Por tanto, esto nos da una explicación alternativa del hecho de que toda clase de treewidth acotado es una isla tratable.

Más allá de treewidth acotado

Una pregunta natural que surge es la siguiente: ¿existen islas tratables más generales que treewidth acotado? ¿Es treewidth acotado la noción óptima? Como veremos a continuación, podemos definir islas tratables aún más generales.

Diremos que A y A' son *equivalentes* si existe un homomorfismo de A a A' y de A' a A. Notemos que si A y A' son equivalentes, entonces, para cualquier estructura B, A es homomorfo a B si y sólo si A' es homomorfo a B (ya que los homomorfismos componen). Luego, A y A' son realmente "indistinguibles" para cualquier CSP.

Pensemos por un momento en grafos dirigidos. Supongamos que queremos decidir si A es homomorfo a B. Asumamos, además, que A es equivalente a un árbol T. Es claro que si tuviéramos disponible T junto con la entrada, podríamos verificar eficientemente si A es homomorfo a B, ya que bastaría verificar acaso T es homomorfo a B. Ahora bien, ¿qué sucede si T no está disponible con la entrada? ¿Es posible aún resolver el problema eficientemente?

A primera vista esto es imposible, ya que no hay manera obvia de computar tal árbol T [12]. Sorprendentemente, utilizando juegos de fichas aún es posible resolver el problema eficientemente sin necesidad de computar el árbol equivalente T. De igual forma, esto extiende a treewidth acotado. Por tanto, si A es equivalente a una estructura de treewidth a lo más k, decidir si Duplicator gana el juego de k+1 fichas sigue siendo correcto y completo para existencia de homomorfismos.

Esto implica que podemos expandir nuestras islas tratables de treewidth acotado a treewidth acotado "módulo equivalencia", como queda enunciado a continuación:

Teorema [12]. Sea k≥1 y C un conjunto de estructuras. Si cada estructura en C es equivalente a una estructura de treewidth a lo más k, entonces CSP(C,-) se puede resolver en tiempo polinomial.

Es importante destacar que, para cualquier k≥1, existen estructuras de treewidth arbitrariamente grande que son equivalentes a estructuras de treewidth a lo más k. Por tanto, estas nuevas islas tratables no sólo contienen a las anteriores, sino que son una generalización interesante y no trivial.

La frontera entre lo tratable y lo intratable

El concepto de treewidth acotado genera islas tratables. Como ya vimos, esto se puede extender a treewidth acotado módulo equivalencia. Pero, ¿podemos ir más allá? ¿Es treewidth acotado módulo equivalencia el límite entre lo tratable y lo intratable?

Sorprendentemente, Grohe en el 2003 dio una respuesta afirmativa a esta pregunta [20]. En un resultado matemáticamente profundo, demostró que si asumimos ciertos supuestos de complejidad parametrizada [16] (supuestos que se creen ciertos, al estilo P≠NP), entonces una clase de estructuras C es una isla tratable si y sólo si C tiene treewidth acotado módulo equivalencia, es decir, existe k≥1 tal que cada estructura en C es equivalente a otra que tiene treewidth a lo más k.

Por tanto, lo que hace que CSP(C,-) admita un algoritmo polinomial es esencialmente que C tenga treewidth acotado módulo equivalencia.

APLICACIONES EN BASES DE DATOS

Calcular el join natural de un conjunto de tablas está en el corazón de todo sistema de manejo de base de datos relacionales. Como este problema es intratable en general [1], definir tipos de joins que admitan evaluación eficiente es un problema fundamental en Computación.

Consideremos por un momento el problema 3-SAT y el ejemplo de la Figura 1. Para computar las soluciones de φ debemos buscar asignaciones a las variables {x,y,z,u,v} que respeten cada restricción. Si observamos la figura, la conexión con base de datos relacionales es clara: podemos pensar las restricciones, como un conjunto de tablas relacionales sobre los atributos {x,y,z,u,v}. Es fácil ver que el conjunto de soluciones para φ es precisamente el join natural de estas 5 tablas.

Esta relación fue observada en 1977 por Chandra y Merlin [8], quienes mostraron que evaluar una consulta J del álgebra relacional que sólo utiliza join natural, sobre una base de datos D es equivalente a verificar existencia de homomorfismos entre un par de estructuras A y B. Más aún, la estructura A es esencialmente la consulta J codificada de manera natural como una estructura. Por tanto, las siguientes dos preguntas son equivalentes:

1. ¿Qué tipos de joins admiten evaluación eficiente?
2. ¿Qué clases C, hacen que CSP(C,-) sea tratable?

A partir de las secciones anteriores, podemos deducir que los tipos de joins relacionales que admiten evaluación eficiente son precisamente los joins con treewidth acotado módulo equivalencia.

Esto muestra una de muchas aplicaciones en Bases de Datos. Más aún, esta transferencia de herramientas no es unidireccional: muchas técnicas de Bases de Datos son utilizadas en satisfacción de restricciones, lo cual ha generado una interesante simbiosis entre estas dos áreas [1, 8, 12, 19, 27].

LA CONJETURA DE LA DICOTOMÍA DE CSPs

No todos los problemas de satisfacción de restricciones tienen la forma CSP(C,-), para un conjunto C de estructuras. Pensemos en 2-coloración: este problema corresponde a CSP(-,{K_2}), por tanto la estructura "de la mano derecha" está fija y es K_2. En este caso, nuestra instancia es una estructura (grafo) G y debemos determinar si es homomorfo a K_2. Lo mismo sucede con 3-SAT, ya que corresponde a CSP(-,{True}).

Por tanto, mucho problemas interesantes tienen la forma CSP(-,{B}), para una estructura fija B, lo cual ha generado mucho interés en este tipo de CSPs [3, 7, 23, 31]. De hecho, los trabajos más profundos matemáticamente acerca de la complejidad de CSP tienen que ver con esta clase de problemas. El motor detrás de este estudio es una conjetura hecha por primera vez en 1993 por Feder y Vardi [15]:

Conjetura (Dicotomía CSP). Para toda estructura B, el problema CSP(-,{B}), o bien puede ser resuelto en tiempo polinomial, o bien es NP-Completo.

A primera vista, puede sonar que esta conjetura es obviamente cierta. Notemos que cada CSP está en NP. Sin embargo, la clase NP probablemente no satisface esta dicotomía, ya que existen problemas que se cree no están en P ni son NP-Completos, como es el caso de Isomorfismos de Grafos y de Factorización [2]. Esto se ve reforzado por el teorema de Ladner [2], que dice que si P≠NP, entonces existen problemas que no están en P y que no son NP-Completos. Luego, nada impide que algunos de estos problemas "intermedios" puedan ser formulado como un CSP de la forma CSP(-,{B}). Por tanto, no hay ninguna razón a priori para pensar que la dicotomía CSP es cierta.

Esta conjetura sigue abierta hasta hoy. Sin embargo, muchos casos importantes han sido resueltos. Algunos de estos casos han sido la dicotomía de Schaefer [31], quien demostró la conjetura para estructuras B con dos elementos, y el trabajo de Hell y Nesetril [23], quienes demostraron la dicotomía para el caso en que B es un grafo no dirigido. En particular, demostraron que CSP(-,{B}) es tratable si B es bipartito, y NP-Completo en caso contrario.

En los últimos años, ha tomado fuerza una línea de ataque a la dicotomía CSP que utiliza herramientas de álgebra universal [3, 4, 6, 7], la cual ha logrado importantes avances. En [4], Bulatov demostró la conjetura para estructuras B con 3 elementos y en [6], para una clase interesante de estructuras llamadas *conservativas*. Además, en [3], Bulatov, Krokhin y Jeavons enriquecieron la conjetura de la dicotomía CSP, conjeturando el borde exacto que separa los casos tratables de los intratables.

CONCLUSIONES

Los problemas de satisfacción de restricciones ocupan un lugar privilegiado en Ciencia de la Computación y son de alto interés tanto teórico como práctico. El estudio teórico de la complejidad de CSPs ha sido muy fructífero, no sólo porque nos permite entender a fondo este tipo de problemas, sino porque en el camino se han desarrollado potentes herramientas teóricas, que son interesantes por sí mismas, y se han establecido fuertes conexiones con otras áreas de interés.

En este artículo hemos cubierto una pequeña fracción del tema. Por ejemplo,

hemos asumido que nuestro esquema está fijo a priori, no obstante muchos problemas interesantes requieren que el esquema sea parte de las instancias. En este contexto, es posible encontrar islas tratables más generales que treewidth acotado módulo equivalencia, basadas en los conceptos de *hypertree width* [19], *coverwidth* [9] y *fractional hypertree width* [22]. Se ha conjeturado en [21], que fractional hypertree width acotado módulo equivalencia es la noción óptima, sin embargo, esto permanece abierto hasta hoy. Otros trabajos relevantes incluyen: el estudio de CSP como un problema de optimización (satisfacer la mayor cantidad de restricciones posibles) y el respectivo análisis de aproximabilidad que esto conlleva [14, 24], y el estudio de la complejidad de contar las soluciones de un CSP [5, 13]. Por ejemplo, es interesante destacar que la dicotomía CSP ha sido resuelta en su versión con respecto a complejidad de conteo [5]. BITS

Referencias

[1] S. Abiteboul, R. Hull, V. Vianu. Foundations of Databases. Addison-Wesley, 1995.

[2] S. Arora, B. Barak. Complexity Theory: A Modern Approach, Cambridge University Press, Cambridge, UK, 2009.

[3] A. Bulatov, P. Jeavons, A. Krokhin. Classifying the Complexity of Constraints Using Finite Algebras. SIAM J. Comput. 34(3): 720-742 (2005).

[4] A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. J. ACM 53(1): 66-120 (2006).

[5] A. Bulatov. The Complexity of the Counting Constraint Satisfaction Problem. ICALP (1) 2008: 646-661.

[6] A. Bulatov. Complexity of conservative constraint satisfaction problems. ACM Trans. Comput. Log. 12(4): 24 (2011).

[7] A. Bulatov. On the CSP Dichotomy Conjecture. CSR 2011: 331-344.

[8] A. Chandra, P. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. STOC 1977: 77-90.

[9] H. Chen, V. Dalmau. Beyond Hypertree Width: Decomposition Methods Without Decompositions. CP 2005: 167-181.

[10] R. Dechter, J. Pearl. Tree Clustering for Constraint Networks. Artif. Intell. 38(3): 353-366 (1989).

[11] R. Dechter. Constraint processing. Morgan Kauman, 2003.

[12] V. Dalmau, Ph. Kolaitis, M. Vardi. Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics. CP 2002: 310-326.

[13] V. Dalmau, P. Jonsson. The complexity of counting homomorphisms seen from the other side. Theor. Comput. Sci. 329(1-3): 315-323 (2004).

[14] V. Deineko, P. Jonsson, M. Klasson, A. Krokhin. The approximability of MAX CSP with fixed-value constraints. J. ACM 55(4) (2008).

[15] T. Feder, M. Vardi. Monotone monadic SNP and constraint satisfaction. STOC 1993: 612-622.

[16] J. Flum, M. Grohe. Parameterized Complexity Theory. Springer-Verlag, 2006.

[17] E. Freuder. Complexity of K-Tree Structured Constraint Satisfaction Problems. AAAI 1990: 4-9.

[18] E. Freuder, A. Mackworth. Constraint-based reasoning. MIT Press, 1994.

[19] G. Gottlob, N. Leone, F. Scarcello. Hypertree Decompositions and Tractable Queries. J. Comput. Syst. Sci. 64(3): 579-627 (2002).

[20] M. Grohe. The Complexity of Homomorphism and Constraint Satisfaction Problems Seen from the Other Side. FOCS 2003: 552-561.

[21] M. Grohe. The Structure of Tractable Constraint Satisfaction Problems. MFCS 2006: 58-72.

[22] M. Grohe, D. Marx. Constraint solving via fractional edge covers. SODA 2006: 289-298.

[23] P. Hell, J. Nesetril. On the complexity of H-coloring. J. Comb. Theory, Ser. B 48(1): 92-110 (1990).

[24] P. Jonsson, M. Klasson, A. Krokhin. The Approximability of Three-valued MAX CSP. SIAM J. Comput. 35(6): 1329-1349 (2006).

[25] Ph. Kolaitis, J. Panttaja. On the Complexity of Existential Pebble Games. CSL 2003: 314-329.

[26] Ph. Kolaitis, M. Vardi. On the Expressive Power of Datalog: Tools and a Case Study. PODS 1990: 61-71.

[27] Ph. Kolaitis, M. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. PODS 1998: 205-213.

[28] Ph. Kolaitis, M. Vardi. A Game-Theoretic Approach to Constraint Satisfaction. AAAI/IAAI 2000: 175-181.

[29] V. Kumar. Algorithms for Constraint-Satisfaction Problems: A Survey. AI Magazine 13(1): 32-44 (1992).

[30] U. Montanari. Networks of Constraints. Fundamental Properties and Application to Picture Processing. Information Science, 7:95-132, 1974.

[31] T. Schaefer. The Complexity of Satisfiability Problems. STOC 1978: 216-226.