Revista Integración
Escuela de Matemáticas
Universidad Industrial de Santander
Vol. 24, No. 1, 2006, pág. 1–15

La dificultad de jugar sudoku

J. Andrés Montoya*

Resumen. Se prueba que el juego sudoku es NP–completo, si se consideran tableros de tamaño n^2 para todo número natural n. Esto explica, en parte, por qué es que resulta tan difícil jugar sudoku.

Abstract. It is proven that the play sudoku is NP–complete, if the size of boards are considered to be n^2 for every natural number n. This partly explains why it is difficult to play sudoku.

A Mamadimitriou e Hijodimitriou

1. Introducción

Esta artículo tiene fundamentalmente dos objetivos, a saber:

1. Revisar los conceptos fundamentales de la complejidad computacional.

2. Usar tales conceptos para analizar la complejidad intrínseca de jugar sudoku.

Palabras y frases claves: máquinas de Turing, clases de complejidad, algoritmos eficientes, costos computacionales.

MSC2000: Primaria: 03D15. Secundaria: 68Q17.

* Escuela de Matemáticas, Universidad Industrial de Santander, Bucaramanga, Colombia, *e-mail*: amontoyaa@googlemail.com, juamonto@uis.edu.co

Todo aquel que haya enfrentado el reto de completar un tablero de sudoku ha sentido que no existe una manera inteligente de solucionar tal *puzzle*, esto es, que la única manera de completar el tablero es probar todas las posibles opciones, que son muy numerosas, hasta encontrar una que satisfaga los requerimientos. Lo anterior indica que, cuando un jugador promedio se enfrenta a un tablero de sudoku, la única estrategia a la que puede hechar mano es a la búsqueda exhaustiva en el espacio de posibles soluciones, esto es, a la búsqueda exhaustiva en el conjunto de todos los posibles completamientos del tablero. El problema con esta metodología (de fuerza bruta) es que el espacio de posibles soluciones es enorme, y esto hace que el tiempo utilizado en completar un tablero sea excesivamente largo. Una pregunta natural que podemos hacernos es la siguiente:

¿Existe una estrategia inteligente (por lo demás, aún no descubierta), o por el contrario el sudoku es un juego intrínsicamente difícil?

En este artículo intentamos dar una respuesta a esta pregunta. La respuesta que el lector encontrará en este artículo es una repuesta parcial y es la siguiente:

Si $NP \neq P$, el sudoku es un juego intrínsicamente difícil.

1.1. Organización del artículo

El artículo esta organizado en 6 secciones, incluyendo la introducción.

En la sección 2 se presenta la noción general de reducción entre problemas computacionales y la noción mas específica de reducción de Karp.

En la sección 3 se define la clase NP.

En la sección 4 se introduce la noción de problema NP–completo.

Finalmente, en la sección 5 se enuncia y prueba el teorema principal de este artículo, a saber, que un cierto problema combinatorio al que llamamos sudoku, (y que corresponde de manera apropiada al juego sudoku), es NP–completo.

En la última sección de este artículo, la sección 6, se presentan algunas conclusiones.

2. Reducciones entre problemas computacionales

En esta sección introduciremos las dos nociones básicas de la complejidad computacional, a saber, la noción de reducción entre problemas computa-

cionales y la noción de problema eficientemente resoluble (o lo que es lo mismo, la clase P)[1].

Para empezar daremos una definición formal de problema computacional, para lo cual usaremos una convención usual en teoría de la complejidad, consistente en restringir nuestra definición al caso particular de los problemas de decisión.

Sea $A \subset \{0,1\}^*$. El conjunto A determina un problema de decisión al que llamaremos $\mathcal{L}_A$ y que se define de la siguiente manera

- *Entrada:* $x \in \{0,1\}^*$.
- *Problema: Decida si* $x \in \mathcal{L}_A$.

Todo problema de decisión es un problema de la forma $\mathcal{L}_A$ para algún $A \subset \{0,1\}^*$. Pero no todo problema de decisión es un problema interesante: los problemas de decisión interesantes son aquellos problemas $\mathcal{L}_A$ tales que el conjunto A que los determina es un conjunto infinito, definido por alguna propiedad natural expresable en una lógica adecuada.

Veamos algunos ejemplos:

Ejemplo 2.1 (Primos).

- Entrada: $n \in \mathbb{N}$.
- Problema: Decida si n es un número primo.

Ejemplo 2.2 (Grafoisomorfismo).

- Entrada: (G, H), donde G y H son grafos.
- Problema: Decida si G y H son isomorfos.

Ejemplo 2.3 (SAT[2]).

- Entrada: α, donde α es una fórmula booleana.
- Problema: Decida si α es satisfactible.

[1] La letra P viene de la expresión inglesa *Polynomial time*, referida al tiempo de cómputo necesario.

[2] Por las letras iniciales de SATisfactibilidad booleana o proposicional.

El lector habrá notado que las posibles entradas (*o instancias*) de los tres problemas anteriores no son elementos del conjunto $\{0,1\}^*$. Lo que sucede es que todo objeto finito, por ejemplo todo número natural, o todo grafo, o toda fórmula booleana, puede ser codificado como una cadena de ceros y unos, de manera tal que la codificación es inyectiva y de manera tal que el tamaño del código binario es similar al tamaño del objeto así codificado. En adelante, a lo largo del texto no prestaremos mayor atención a la manera como codificamos en binario un objeto combinatorio finito, o mejor una clase de tales objetos; nos bastará saber que existen codificaciones eficientes. Así pues, cuando definamos un problema computacional o cuando consideremos las posibles instancias de un problema, pensaremos en ellas como lo que son (números naturales, grafos, fórmulas), y no en sus códigos binarios.

Del lector supondremos alguna familiaridad con el análisis de algoritmos y con el modelo de máquina de Turing, lo cual nos permite suponer que maneja el concepto de tiempo de cómputo de un algoritmo.

Definición 2.4 (Reducciones de Karp). Sean $\mathcal{L}_1$ y $\mathcal{L}_2$ dos problemas computacionales; diremos que $\mathcal{L}_1$ es Karp-reducible a $\mathcal{L}_2$ si y sólo si existe un algoritmo $\mathbb{M}$ tal que:

1. el tiempo de cómputo de $\mathbb{M}$ es polinomial;
2. para toda instancia x del problema $\mathcal{L}_1$, el algoritmo $\mathbb{M}$ calcula una instancia y del problema $\mathcal{L}_2$ tal que $x \in \mathcal{L}_1$ si y sólo si $y \in \mathcal{L}_2$.

Para indicar que un problema $\mathcal{L}_1$ es Karp-reducible a un problema $\mathcal{L}_2$, usaremos la notación $\mathcal{L}_1 \preceq_m \mathcal{L}_2$.

Veamos un ejemplo de reducción de Karp. Antes debemos definir un par de problemas.

Definición 2.5. El problema del clique[3] $\mathcal{L}_{clique}$ es el siguiente problema de decisión

- Entrada: (G,k), G es un grafo y $k \in \mathbb{N}$.
- Problema: Decida si G contiene un clique de tamaño k.

[3]En inglés *clique* designa un grupo cerrado de personas con intereses específicos. El préstamo léxico se ha generalizado en español en la teoría de complejidad computacional, así que aquí lo usaremos libremente.

Definición 2.6. El problema del conjunto independiente $\mathcal{L}_{IS}$ es el siguiente problema de decisión

- Entrada: (G, k), G es un grafo y $k \in \mathbb{N}$.
- Problema: Decida si G contiene un conjunto independiente de tamaño k.

Es fácil ver que el problema del clique es Karp-reducible al problema del conjunto independiente. Considérese el siguiente algoritmo $\mathbb{M}$ con entrada (G, k):

1. $\mathbb{M}$ calcula el grafo dual de G, llamémoslo G^*.
2. $\mathbb{M}$ calcula el par (G^*, k).

Primero nótese que G contiene un clique de tamaño k si y sólo si G^* contiene un conjunto independiente de tamaño k, esto es,

$$(G, k) \in \mathcal{L}_{clique} \quad \textit{si y sólo si} \quad (G^*, k) \in L_{IS}.$$

Para terminar, nótese que el tiempo de cómputo del algoritmo es polinomial (incluso lineal) en el tamaño de la entrada, $\|(G, k)\|$, el cual se define de la siguiente manera:

$$\|(G, k)\| := |V(G)| + |E(G)| + k.$$

La teoría de la complejidad podría ser definida como la disciplina que intenta:

1. realizar una distinción adecuada entre problemas fáciles y problemas difíciles;
2. clasificar problemas específicos como fáciles o difíciles, según sea el caso.

Es por ello que una de las nociones fundamentales es la noción de problema eficientemente soluble.

Definición 2.7. Diremos que un problema $\mathcal{L}$ es *eficientemente soluble* si y sólo si existe un algoritmo $\mathbb{M}$ tal que:

1. el tiempo de cómputo de $\mathbb{M}$ es polinomial;

2. el algoritmo $\mathbb{M}$ resuelve el problema $\mathcal{L}$.

Proposición 2.8. *Si $\mathcal{L}_1 \preceq_m \mathcal{L}_2$ y $\mathcal{L}_2$ es eficientemente soluble, entonces $\mathcal{L}_1$ es eficientemente soluble.*

Definición 2.9. La clase P es la clase de problemas eficientemente solubles (*feasible* en inglés), o de manera más precisa, P es la clase de los problemas que pueden ser resueltos en tiempo polinomial.

En la definición anterior hemos identificado la noción de ser eficientemente decidible con la noción de ser decidible en tiempo polinomial. Esta es, por supuesto, una identificación arbitraria, que no obstante asumiremos como la asumen todos los miembros de la *complexity-community*.

Definición 2.10. Una clase de complejidad $\mathcal{C}$ es simplemente un conjunto de problemas de decisión cerrado bajo reducciones de Karp.

Una de las nociones centrales en teoría de la complejidad es la noción de problema completo para una cierta clase. Es posible afirmar que la teoría de la complejidad nace en el año de 1971, cuando Steven Cook publica su famoso artículo [2], en el cual define las nociones de reducción y de completez aplicadas específicamente a la clase *NP* (del inglés *Nondeterministic Polynomial time*).

Definición 2.11. Dada $\mathcal{A}$ un conjunto de problemas de decisión, la Karp-clausura de $\mathcal{A}$ es la clase de complejidad $\langle \mathcal{A} \rangle_m$ definida por

$$\langle \mathcal{A} \rangle_m := \{\mathcal{T} \subset \{0,1\}^* : \text{ Existe } \mathcal{L} \in \mathcal{A} \text{ tal que } \mathcal{T} \preceq_m \mathcal{L}\}.$$

Definición 2.12. Dada una clase de complejidad $\mathcal{C}$, diremos que $\mathcal{L}$ es $\mathcal{C}$-duro si y sólo si $\mathcal{C} = \langle \mathcal{L} \rangle_m$. Si adicionalmente $\mathcal{L} \in \mathcal{C}$, diremos que $\mathcal{L}$ es $\mathcal{C}$-completo.

Muchos de los problemas centrales en teoría de la complejidad caen en una de las siguientes dos categorías:

1. (*Problema de la contenencia*) Dadas $\mathcal{C}_1$ y $\mathcal{C}_2$ dos clases de complejidad, pruébese que $\mathcal{C}_1 \subset \mathcal{C}_2$.

2. (*Problema de la separación*) Dadas $\mathcal{C}_1$ y $\mathcal{C}_2$ dos clases de complejidad, pruébese que $\mathcal{C}_1 \nsubseteq \mathcal{C}_2$.

La noción de problema completo es una noción que puede ayudar a resolver casos particulares de los dos problemas anteriores. Nótese que

1. Si $\mathcal{L}$ es $\mathcal{C}_1$-completo, entonces $\mathcal{C}_1 \subset \mathcal{C}_2$ si y sólo si $\mathcal{L} \in \mathcal{C}_2$.
2. Si $\mathcal{L}$ es $\mathcal{C}_2$-completo, entonces $\mathcal{C}_1 \nsubseteq \mathcal{C}_2$ si y sólo si $\mathcal{L} \notin \mathcal{C}_1$.

La noción de reducción de Karp es una de las muchas posibles nociones de reducción, solo una de las muchas que han sido estudiadas en profundidad. De la casi imnumerable lista de reducciones estudiadas en teoría de la complejidad, se destacan dos nociones naturales y robustas, a saber, la noción de reducción de Karp y la noción de reducción de Turing. En este breve artículo consideraremos únicamente el primer tipo de reducción.

3. *La clase* NP

En esta sección introduciremos la clase NP. Existen muchas maneras de definir NP; nosotros hemos elegido solo una de entre las muchas opciones posibles. Para empezar consideremos los siguientes problemas computacionales.

Problema (CIRCSAT[4]).

- Entrada: C un circuito booleano.
- Problema: Decida si C es satisfactible, esto es, decida si existe un modelo para C.

Definición 3.1. Dado un grafo G, un ajuste en G es una colección $\{e_1, \ldots, e_n\}$ de aristas tal que:

1. si $i \neq j$, e_i y e_j no tienen extremos comunes;
2. todo vértice de G es el extremo de alguna de las aristas en $\{e_1, \ldots, e_n\}$.

Problema (AJUSTE).

- Entrada: G un grafo.
- Problema: Decida si G tiene un ajuste.

Los dos problemas anteriores tiene en común el ser solubles en tiempo polinomial no determinista. ¿Qué quiere decir lo anterior? Considere los siguientes algoritmos.

[4]Del inglés CIRCuit SATisfiability.

Algoritmo 3.2 (CIRCSAT). Con entrada C, donde C es un circuito booleano con k entradas:

1. Adivine $v_1, \ldots, v_k$, donde $v_1, \ldots, v_k \in \{0, 1\}$.
2. Verifique que el vector booleano $(v_1, \ldots, v_k)$ satisface el circuito C.
3. Acepte si este es el caso, en caso contrario rechace.

Algoritmo 3.3 (AJUSTE). Con entrada G.

1. Adivine un conjunto de aristas $\{e_1, \ldots, e_n\}$.
2. Verifique que $\{e_1, \ldots, e_n\}$ es un ajuste.
3. Acepte si $\{e_1, \ldots, e_n\}$ es un ajuste, en caso contrario rechace.

Los dos algoritmos tienen muchas cosas en común, una de ellas es que su tiempo de cómputo es polinomial, otra, quizás la mas importante, es la sospechosa primera instrucción de cada uno de ellos. Es obvio que la instrucción *adivine algo* no puede ser una instrucción de un algoritmo en el sentido tradicional del término, por lo cual decimos que los tres algoritmos son no deterministas, porque no se determina la manera en que la escongencia en el paso 1 debe ser realizada y solo se le pide al algoritmo que adivine un objeto adecuado, un objeto solución, si es que tales objetos (soluciones) existen. El no determinismo incrementa nuestro poder de cómputo, aunque no de manera ilimitada. Los dos problemas anteriores, que pueden ser muy difíciles de resolver sin la ayuda del no determinismo, son fáciles de resolver con su ayuda. Esto es así dado que, para cada uno de los dos problemas, decidir si la entrada x pertenece al problema es equivalente a decidir si existe un objeto y (un modelo, un ajuste) de tamaño polinomial tal que $(x, y) \in R$, donde R es una relación fácil de verificar. En los ejemplos anteriores la relación R corresponde a:

1. $((G, k), \{v_1, \ldots, v_k\}) \in R$ si y sólo si para todo $i, j \leq k$, si $i \neq j$, entonces v_i es adyacente a v_j; es decir, si y sólo si $\{v_1, \ldots, v_n\}$ es un clique en G.
2. $(G, \{e_1, \ldots, e_n\}) \in R$ si y sólo si $\{e_1, \ldots, e_n\}$ es un ajuste en G.

Nótese que en cada uno de los casos anteriores la relación R es fácil de verificar, es decir, dada (x, y), podemos decidir rápidamente si (x, y) pertenece a R.

Dado x y dado y tales que $(x, y) \in R$, diremos que y es un *certificado* para x, porque en cierto sentido y certifica que x pertenece al lenguaje. Un problema $\mathcal{L}$ puede ser resuelto en tiempo polinomial no determinista si el problema determina una noción adecuada de certificado, una noción tal que dada cualquier instancia x del problema es fácil decidir si $x \in \mathcal{L}$, conociendo un certificado para x.

Dado $n \in \mathbb{N}$, $\{0,1\}^n$ es el conjunto de las palabras de longitud n, mientras que $\{0,1\}^{\leq n}$ es el conjunto de las palabras de longitud menor o igual que n.

Sea $\mathcal{L}$ un problema, R una relación y p un polinomio tales que:

1. $x \in \mathcal{L}$ si y sólo si existe $y \in \{0,1\}^{p(|x|)}$ tal que $(x, y) \in R$;
2. R es fácil de verificar.

De R diremos que es una relación p-balanceada, y que el polinomio p es su módulo.

El problema $\mathcal{L}$ es un problema genérico dentro de las clase de los problemas que pueden ser resueltos velozmente con la ayuda del no determinismo. Considere el siguiente algoritmo:

Con entrada x,

1. Adivine $y \in \{0,1\}^{p(|x|)}$;
2. Verifique que $(x, y) \in R$;
3. Acepte si $(x, y) \in R$, en caso contrario rechace.

El tiempo de cómputo de este algoritmo está dado por $p(|x|)$, que es el tiempo requerido para adivinar y, mas el tiempo requerido para verificar que $(x, y) \in R$.

Por otro lado, el problema anterior puede ser resuelto de manera trivial sin el concurso del no determinismo, para lo cual podemos usar un algoritmo de fuerza bruta o de búsqueda exhaustiva. Sea $\mathbb{M}$ el algoritmo dado por:

Con entrada x,

1. Liste los elementos de $\{0,1\}^{p(|x|)}$;
2. Recorra la lista y para cada y en la lista decida si $(x, y) \in R$;
3. Si existe y tal que $(x, y) \in R$ acepte, en caso contrario rechace.

El algoritmo $\mathbb{M}$ es correcto, pero no es un buen algoritmo; el problema es que el espacio de búsqueda es demasiado grande, $\left|\{0,1\}^{p(|x|)}\right| = 2^{p(|x|)}$, por lo que el tiempo de cómputo de $\mathbb{M}$ no podrá ser inferior a $2^{p(|x|)}$. El tiempo de cómputo de $\mathbb{M}$ no es polinomial. Todo algoritmo de búsqueda exhaustiva es ineficiente cuando el espacio de búsqueda es grande; lo que se requiere entonces son algoritmos que usen estrategias inteligentes de búsqueda. La estrategia más inteligente posible es el no determinismo, (corresponde a la estrategia de una inteligencia omnimiscente). Infortunadamente es tan inteligente que no podemos implementarla computacionalmente.

Todo problema de la forma

Dado x determine si x tiene un certificado

puede ser resuelto eficientemente con la ayuda del no determinismo, es decir con la ayuda de una inteligencia omnimiscente.

No siempre el no determinismo es indispensable. Existe un algoritmo polinomial determinista que resuelve el problema $AJUSTE$ [3]. En otras ocasiones, el no determinismo parece indispensable, como en el caso del problema $CIRCSAT$ el cual es NP–completo (es decir, si para él no es indispensable el no determinismo, entonces el no determinismo nunca es indispensable).

Definición 3.4. NP es la clase de todos los problemas de decisión que pueden ser resueltos en tiempo polinomial no determinista.

Para finalizar esta sección introduciremos el problema *sudoku*, y verificaremos que este problema pertenece a la clase NP.

Sea n un número natural; considérese una matriz M de n^2 por n^2, supóngase que $M = [m_{ij}]_{i,j\leq n^2}$ y que sus entradas, es decir los números m_{ij}, pertenecen en el conjunto $\{1,\ldots,n^2\}$.

Dados $i, j \leq n$, el bloque $M_{(i,j)}$ de M es la submatriz de M, de tamaño n por n, definida por

$$M_{(i,j)} = \left[m_{((i-1)n+l)((j-1)n+s)}\right]_{l,s\leq n}.$$

M es un *sudoku* de tamaño n^2 si y sólo si

1. en ninguna fila y en ninguna columna de M ningún número ocurre al menos dos veces;

2. para cualesquiera $i, j \leq n$, ningún número ocurre al menos dos veces en el bloque $M_{(i,j)}$.

Un *sudoku parcial* de tamaño n^2 es una matriz M de n^2 por n^2 tal que sus entradas pertenecen al conjunto $\{1, \ldots, n^2, Ⓢ\}$ y tal que

1. en ninguna fila y en ninguna columna de M ningún número ocurre al menos dos veces;

2. para cualesquiera $i, j \leq n$, ningún número ocurre al menos dos veces en el bloque $M_{(i,j)}$.

Dado M un sudoku parcial de tamaño n^2 diremos que M es *completable* si y sólo si existe una modo de reemplazar todas las ocurrencias del símbolo Ⓢ por elementos del conjunto $\{1, \ldots, n^2\}$, de manera que tras realizar tales sustituciones se obtenga un sudoku.

Problema (Sudoku).

- Entrada: (M, n), donde M es un sudoku parcial de tamaño n^2.
- Problema: Decida si M es completable.

Es fácil verificar que el problema *sudoku* pertenece a *NP*.
Sea $[n^2]$ el conjunto $\{1, \ldots, n^2\}$. Dado un sudoku parcial M de tamaño n^2, un *completado parcial* para M es una función $f : [n^2] \times [n^2] \to [n^2]$ tal que

1. Si $M_{ij} \neq Ⓢ$, entonces $M_{ij} = f((i, j))$;

2. La matriz $[f((i, j))]_{i,j \leq n^2}$ es un sudoku.

Sea R la relación

$$\{(M, f) : M \text{ es un sudoku parcial y } f \text{ es un completado para } M\}.$$

Es claro que R es una relación p-balanceada y también es claro que

$$(M, n) \in \text{ sudoku si y sólo si existe } f \text{ tal que } (M, f) \in R.$$

Lo anterior implica que *sudoku* es un problema en *NP*, para el cual la noción de certificado es desempeñada por los completados.

4. Problemas NP–completos

En esta sección introduciremos la noción de NP–completez. Los problemas NP–completos son problemas difíciles o *duros,* que aunque podemos resolver velozmente con el concurso del no determinismo, por el momento solo sabemos resolver determinísticamente usando búsqueda exhaustiva (fuerza bruta). Los problemas NP–completos son tan difíciles que hoy en día no sabemos resolver alguno de ellos de manera eficiente. Si aprendiéramos a resolver velozmente alguno de estos problemas, aprenderíamos entonces a resolver velozmente todo problema en NP.

Definición 4.1. Un problema $\mathcal{L}$ es NP–completo si y sólo si

1. $\mathcal{L} \in NP$;
2. Si $\mathcal{T} \in NP$, entonces $\mathcal{T}$ es Karp-reducible a $\mathcal{L}$.

Si suponemos que $P \neq NP$, es decir, si suponemos que el no determinismo incrementa efectivamente nuestro poder de cómputo, entonces todo problema NP–completo es demasiado difícil como para que podamos resolverlo eficientemente, o lo que es lo mismo, estos problemas son demasiado difíciles como para poder implementar una estrategia de búsqueda más inteligente y más económica que la búsqueda exhaustiva. Los textos de complejidad computacional (ver, por ejemplo [3]) suelen contener una extensa lista de problemas NP–completos. La NP–completez de ciertos problemas explica, en parte, por qué no podemos resolver estos problemas de manera eficiente. Si nosotros pudiéramos arreglárnoslas para probar que el problema de completar sudokus es NP–completo, explicaríamos, al menos parcialmente, por qué es que ninguna estrategia inteligente parece funcionar a la hora de jugar sudoku. La explicación sería parcial, en cuanto el contenido efectivo de la noción de NP–completez depende de una hipótesis, a saber, que P es diferente de NP. Es importante anotar que existe un gran concenso respecto a que tal hipótesis debe ser cierta. Piénsese en lo siguiente: sea p un polinomio y sea A_p el conjunto de las sentencias α en el lenguaje de la teoría de conjuntos que tienen una prueba de tamaño a lo más $p(|\alpha|)$. Sea R la relación

$$R := \{(\alpha, \Sigma) : \alpha \text{ es una sentencia y } \psi\},$$

donde ψ es la afirmación. *Algún segmento inicial de* $\Sigma \in \{0,1\}^{p(|\alpha|)}$ *es una prueba de* α.

Nótese que R es una relación p-balanceada. Nótese también que

1. A_p contiene casi todos los teoremas de la matemática.
2. Si $NP = P$, existe entonces un algoritmo que con entrada α produce una prueba de α. Esto es, si $P = NP$, casi toda la matemática es automatizable (creemos y esperamos que este no sea el caso).

5. *Sudoku es NP–completo*

En esta sección probaremos que el problema *sudoku* es *NP*–completo, para lo cual mostraremos que existe un problema *NP*–completo que es Karp-reducible a *sudoku*.

Sea n un número natural; considérese una matriz M de n por n tal que sus entradas son números en el conjunto $\{1, \ldots, n\}$.

Una tal matriz será un *cuadrado latino* de tamaño n si y sólo si en ninguna fila y en ninguna columna de M ningún número ocurre al menos dos veces.

Un *cuadrado latino parcial* de tamaño n es una matriz M de n por n tal que sus entradas pertenecen al conjunto $\{1, \ldots, n, Ⓢ\}$, y tal que en ninguna fila y en ninguna columna de M ningún número ocurre al menos dos veces.

Dado M un cuadrado latino parcial de tamaño n, diremos que M es *completable* si y sólo si existe un modo de reemplazar todas las ocurrencias del símbolo Ⓢ por elementos del conjunto $\{1, \ldots, n\}$ de manera que tras realizar tales sustituciones se obtenga un cuadrado latino.

Problema (Latin Square Completion, LSC).

- Entrada: (M, n), donde M es un cuadrado latino parcial de tamaño n.
- Problema: Decida si M es completable.

Es fácil verificar que el problema LSC pertenece a NP.

Teorema 5.1. *LSC es NP–duro.*

Para una prueba el lector puede consultar [1].

Corolario 5.2. *LSC es NP–completo.*

En lo que sigue demostraremos el siguiente teorema:

Teorema 5.3. Sudoku *es NP–completo.*

Antes debemos introducir alguna notación y alguna terminología.

Dada $M = [m_{i,j}]_{i,j\leq n^2}$ una matriz de n^2 por n^2, usaremos el símbolo F_i para denotar la i-ésima fila de M; podemos ver cada fila F_i como constituida por n bloques de n entradas; al j-ésimo bloque lo denotaremos con el símbolo $F_{i,j}$. Para todo $i \leq n^2$ y todo $j \leq n$, el bloque $F_{i,j}$ es el n-vector $\left[m_{i,((j-1)n+l)}\right]_{l\leq n}$.

Dado $v = [v_i]_{i\leq n}$ un n-vector, y $\pi \in \mathcal{S}_n$ una permutación, $\pi(v)$ es el vector $\left[v_{\pi(i)}\right]$. Y dado $m \in \mathbb{N}$, $\pi^{(m)}(v)$ es el vector $\left[v_{\pi^{(m)}(i)}\right]$. En lo que sigue fijaremos la permutación $\nu : [n] \to [n]$ definida por

$$\nu(i) = i + 1 \mod (n).$$

$S^1 = \left[s^1_{i,j}\right]$ es la matriz de $(n-1)$ por n dada por $s^1_{i,j} = (i \cdot n) + j$, para todo $i \leq n-1$, $j \leq n$.

La matriz S^1 esta constituida por las filas $FS^1_1, \ldots, FS^1_{n-1}$ y por las columnas $C_1, \ldots, C_n$.

Para cada $i \in \{2, \ldots, n\}$, la matriz S^i es la matriz constituida por las columnas $C_{\nu^{(i-1)}(1)}, \ldots, C_{\nu^{(i-1)}(n)}$. Esto es, la matriz S^i es un reordenamiento de las columnas de S^1 determinado por la permutacion $\nu^{(i)}$. Nótese que $\nu^{(i)}(FS^1_1), \ldots, \nu^{(i)}\left(FS^1_{n-1}\right)$ son las filas de S^i. A estas filas las denotaremos con los símbolos $FS^i_1, \ldots, FS^i_{n-1}$.

Sea L un cuadrado latino parcial de tamaño n, y sean $L_1, \ldots, L_n$ las filas de L.

En lo que sigue construiremos una matriz $M(L)$ de tamaño n^2 por n^2 con entradas en $\{1, \ldots, n^2, Ⓢ\}$:

1. Para todo $i \leq n$, el bloque $F_{(i-1)n+1,1}$ es igual a L_i.
2. Para todo $i \leq n$ y todo $j \in \{2, \ldots, n\}$, el bloque $F_{(i-1)n+j,1}$ es igual a FS^i_{j-1}.
3. Para todo $i \leq n$ y todo $j \in \{2, \ldots, n\}$, el bloque $F_{(i-1)n+1,j}$ es igual a FS^i_{j-1}.
4. Para todo $i \leq n$ y todo $j \in \{2, \ldots, n\}$, el bloque $F_{(i-1)n+j,j}$ es igual a L_i.
5. Para todo $i \leq n$, todo $l \in \{2, \ldots, n\}$ y todo $j \in \{2, \ldots, n\}$, si $l \neq j, 1$ el bloque $F_{(i-1)n+l,j}$ es igual a $FS^i_{\upsilon^{(j-1)}(l-1)}$, donde $\upsilon : [n-1] \to [n-1]$ es la permutación definida por

$$\upsilon(i) := i + 1 \mod (n-1).$$

Proposición 5.4. *$M(L)$ es un sudoku parcial completable si y sólo si L es un cuadrado latino completable.*

Demostración. La prueba es inmediata y por ello se omite. ☑

Corolario 5.5.

1. *La asignación $L \longmapsto M(L)$ es una reducción de Karp del problema LSC en el problema sudoku.*

2. *Sudoku es NP–completo.*

6. Conclusiones

Jugar sudoku es muy difícil, o mejor, completar un sudoku es muy difícil; quien lo intenta tiene la sensación de que no existe una estrategia inteligente para enfrentar el reto de completar un tablero de sudoku. Los modestos resultados del presente artículo justifican y explican dicha intuición. Cuando jugamos sudoku nos enfrentamos a una instancia de un problema NP–completo, y un problema NP–completo es precisamente un problema que no admite estrategias inteligentes (al menos hasta donde llega nuestro conocimiento).

Lo anterior muestra cómo los conceptos y métodos de la complejidad computacional pueden explicar, o al menos dar nombre, a las limitaciones combinatorias de la mente humana.

Referencias

[1] Colbourn C. "The complexity of completing partial latin squares". *Discrete Applied Mathematics*, **8** (1984), 151–158.

[2] Cook S. "The complexity of theorem proving procedures". *Proceedings, ACM Symposium on Theory of Computing*, (1971), 151–158.

[3] Papadimitriou C. H. *Computational Complexity*. Addison-Wesley, 1994.

J. Andrés Montoya
Escuela de Matemáticas
Universidad Industrial de Santander
Bucaramanga, Colombia
e-mail: amontoyaa@googlemail.com, juamonto@uis.edu.co