Universidad Nacional de Colombia
Facultad de Ciencias - Departamento de Matemáticas
Introducción a la Inteligencia Artificial - 2025 II

Mateo Sebastian Ortiz Higuera mortizhi@unal.edu.co

Reporte Primer Proyecto
Solucionador de Sudoku

1. Introducción

Resolver un sudoku es una tarea de gran interés matemático. A lo largo del tiempo se han propuesto diversos métodos y algoritmos que buscan estrategias más eficientes que la enumeración exhaustiva. Esto es muy importante, ya que el número de posibles configuraciones asciende a $9^{81} \approx 1.996 \times 10^{77}$, una cifra comparable al número estimado de átomos en el universo observable (aproximadamente 10^{80}). Esta magnitud ilustra la complejidad del problema, que puede formalizarse como un problema de satisfacción de restricciones (CSP, por sus siglas en inglés), en la misma categoría que el problema de las n-reinas o el n-coloreo. Bajo este marco teórico, resolver un sudoku se clasifica como un problema NP-Completo[1].

	2			3	9			
			5					
	4			1			7	
	7			2	3		9	
			4				2	
		5						6
		8				5	3	
				9	5	1	4	
			2					7

Figura 1: Ejemplo de una configuración inicial de un juego de Sudoku

Entre los métodos más populares utilizados para solucionar el sudoku se encuentran el Backtracking Cronológico[2], el algoritmo de búsqueda por armonía (Harmony Search en inglés)[4], métodos estocásticos como la búsqueda progresiva estocástica (PSS de sus siglas en inglés)[5] o la optimización del enjambre de partículas repulsivas (RPSO de sus siglas en inglés)[7], además de otras aproximaciones que combinan o que son una variante de los algoritmos anteriores[3][6].

En este trabajo vamos a resolver el problema en un tablero clásico de tamaño 9×9 utilizando dos agentes basados en planificación: el primero es el algoritmo de búsqueda con costo uniforme y el

segundo es el algoritmo A*. Después experimentaremos con ellos para ver la diferencia entre el uso de búsqueda no informada y la búsqueda informada.

2. Estados del problema

En esta implementación vamos a considerar cada estado como la matriz asociada al sudoku, en donde inicialmente las celdas sin información serán rellenadas por ceros. Luego el estado inicial del problema será una matriz de tamaño 9×9 con las pistas iniciales y el resto de elementos en blanco. Por ejemplo, el estado inicial del juego de la figura 1 sería

$$S_1 = \begin{bmatrix} 0 & 2 & 0 & 0 & 3 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 1 & 0 & 0 & 7 & 0 \\ 0 & 7 & 0 & 0 & 2 & 3 & 0 & 9 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 6 \\ 0 & 0 & 8 & 0 & 0 & 0 & 5 & 3 & 0 \\ 0 & 0 & 0 & 0 & 9 & 5 & 1 & 4 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 7 \end{bmatrix}$$

El costo por ir de un estado a otro es de 1, que corresponde al número de celdas rellenadas.

3. Función Sucesora

Para la función sucesora consideramos el método de ordenamiento de mínimos valores restantes (por sus siglas en inglés MRV). En donde cada una de las celdas restantes del tablero es revisada y se toma la que viole menos restricciones, si dos celdas tienen el mismo número mínimo de restricciones se toma la primera en haber sido revisada. Finalmente, se retorna una lista con todos los estados posibles que resultan de las selecciones válidas en dicha celda.

3.1 *Heurísticas*

Para el algoritmo de A* se han probado dos heurísticas:

- *Espacios vacíos*: En esta heurística simplemente evaluamos el número de espacios vacíos que tiene el tablero. Dado que el costo de cada sucesión es 1 entonces esta heurística es admisible. Formalmente

  $$h(n) = |B|$$

  donde $B = \{(i, j) \in S : s_{i,j} = 0\}$, $S = D \times D$ es la matriz que representa un estado del tablero. y $D = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

- *MRV con espacios vacíos*: En esta heurística tenemos tres casos: si no hay espacios en blanco, entonces es 0, si existe una celda que no es válida entonces retornamos ∞, y si no, entonces sumamos el número de espacios vacíos que tiene el tablero con la tasa de asignaciones restantes en cada una de las celdas vacías entre el número total de posibles asignaciones. Formalmente

$$h(n) = \begin{cases} 0, & \text{si } |B| = 0 \\ \infty, & \text{si } \exists (i,j) \in B \text{ tal que } |v_{i,j}| = 0 \\ |B| + \frac{\sum_{i,j \in B} |v_{i,j}|}{|B| \times 9}, & \text{en otro caso} \end{cases}$$

  en donde $v_{i,j} = \{n \in D : n \text{ no viola ninguna restricción en } s_{i,j}\}$ es el conjunto de los valores legales en la celda $s_{i,j}$. De esta forma esta heurística no es admisible, sin embargo, es mucho más informativa que la heurística anterior. En particular, si el estado actual presenta una celda inválida entonces al retornar ∞ este estado nunca será explorado. Por otro lado, entre menos asignaciones posibles tengan todas las celdas, entonces la tasa será más pequeña, penalizando menos la función de costo total.

4. Prueba de Objetivo

Para las pruebas de funcionamiento de los algoritmos se ha utilizado la librería `py-sudoku`[1] para generar tableros con distintos niveles de dificultad, y para después contrastar y verificar las respuestas obtenidas con las respuestas dadas por la misma librería.

```
+-------+-------+-------+
| 2 3   |       |       |
| 7 8 5 | 2 9   |   3   |
|       |       |   5   |
+-------+-------+-------+
| 5     |   6 9 | 4     |
| 4     | 8 5   | 3 7   |
|     8 | 7     |       |
+-------+-------+-------+
|   4 2 | 6     |   8 5 |
| 8 1 6 |     5 | 7 4   |
|       |   8 3 |       |
+-------+-------+-------+
```

Figura 2: Tablero generado con la librería `py-sudoku`.

Las pruebas realizadas para este informe están disponibles en el cuaderno de Jupyter del repositorio del proyecto[2].

[1] `https://pypi.org/project/py-sudoku/`
[2] `https://github.com/MateoOrtiz001/Sudoku-Solver`

La figura 3 muestra cuatro pasos de la solución encontrada por el algoritmo A* para el tablero de la figura 2, generado automáticamente con un nivel de dificultad de 0.6 (32 pistas iniciales).

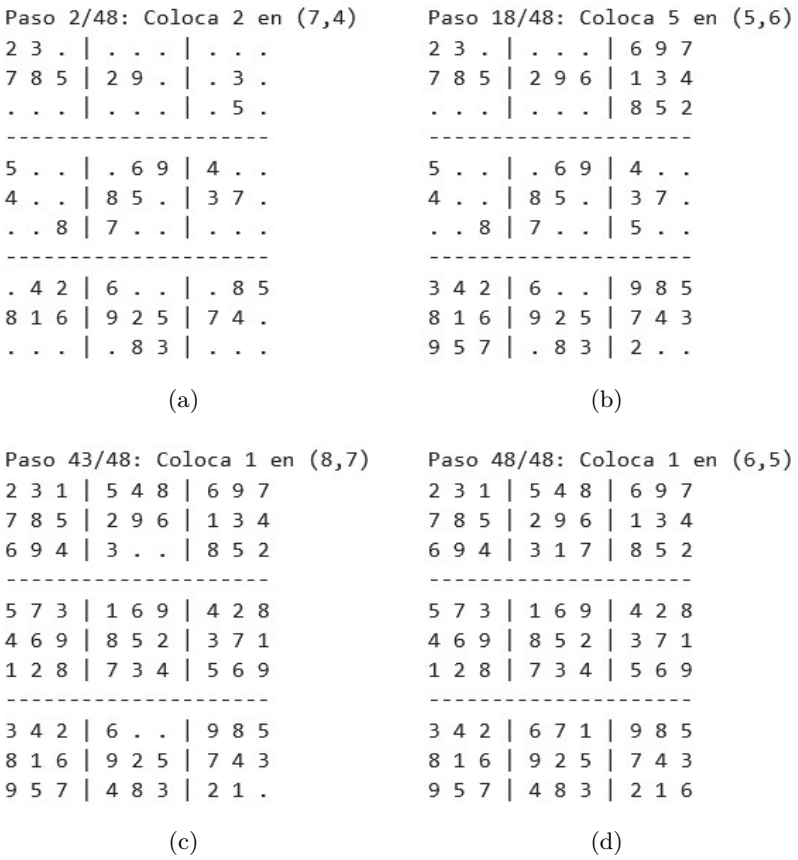
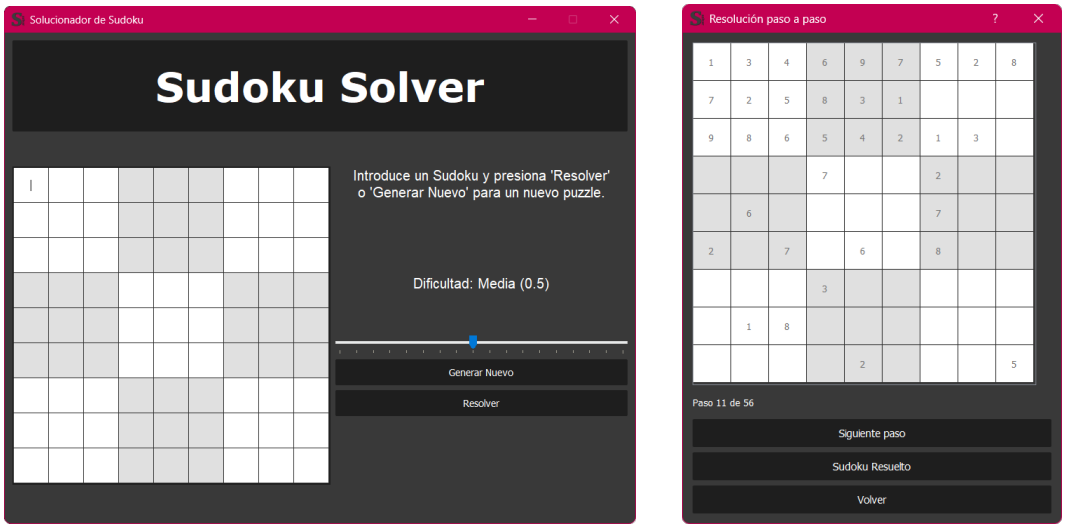


Figura 3: Proceso de solución del Sudoku implementado en el cuaderno con A*. (a) Es el segundo paso del camino a la solución encontrado, (b) el paso 18, (c) el 43 y (d) es el último paso para completar el tablero.

Además, como adicional se ha desarrollado una aplicación que permite introducir al usuario una matriz personalizada y la soluciona utilizando A* paso a paso:



(a) (b)

Figura 4: Vista de las dos ventanas de la aplicación.

5. Resultados

Para la elección del algoritmo de A* con heurística basada en MRV hemos realizado algunos experimentos, comparándolo con el algoritmo de búsqueda uniforme y A* con la heurística de espacios en blanco.

Primero se ha realizado una comparación entre los tiempos medios de ejecución por nivel de dificultad (desde 0.5 hasta 0.95). Los datos obtenidos se pueden visualizar en la figura 5; se puede visualizar que utilizando el algoritmo de A* con heurística basada en MRV se mantiene con un comportamiento asintótico constante a medida que el nivel de dificultad aumenta, mientras que el A* con la heurística simple y el algoritmo de búsqueda uniforme explotan y se hace prácticamente imposible de medir el tiempo cuando la dificultad es mayor a 0.8.

Similarmente ocurre comparando el número de nodos expandidos en la búsqueda, en la figura 5 se puede observar que con el algoritmo de A* el comportamiento es constante mientras que con la heurística simple y con el algoritmo de búsqueda el número de nodos explorados explotan.

De este primer experimento podemos deducir que con el algoritmo A* con heurística MRV optimizamos bastante en tiempo de ejecución y en memoria utilizada. En niveles de dificultad mayores a 0.8 se hace insostenible utilizar A* con la heurística simple o búsqueda uniforme.

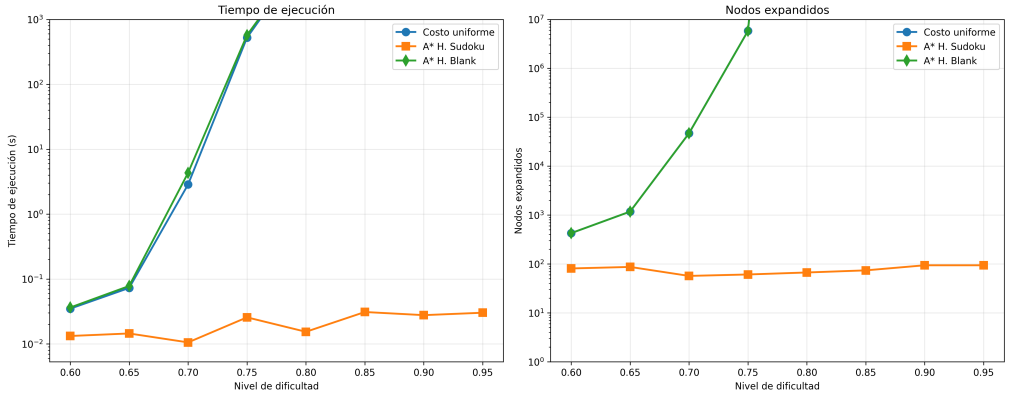
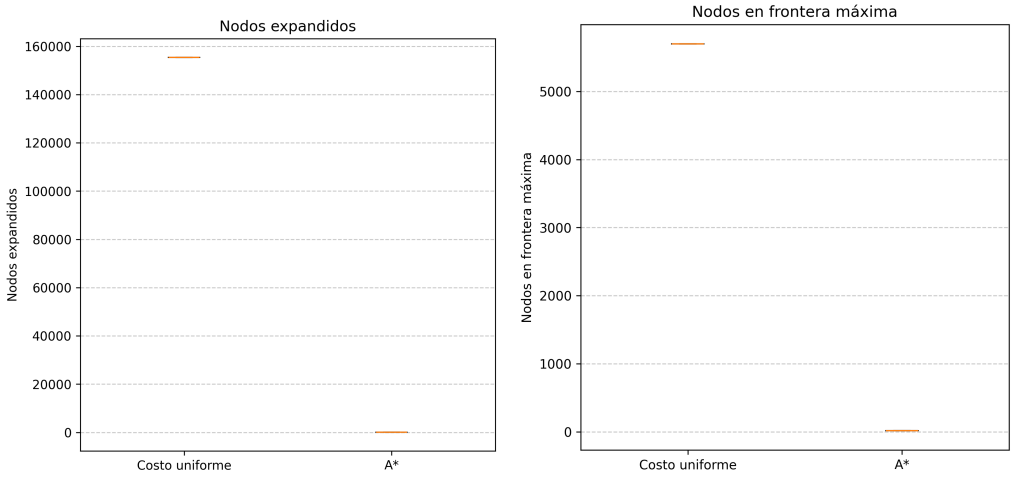


Figura 5: Comparación entre los algoritmos utilizados en tiempo de ejecución y en número de nodos expandidos.

Además, se hace evidente lo poco informativa que resulta ser la primera heurística, al mostrar tiempos similares a las pruebas con el algoritmo de búsqueda uniforme. Razón por la cual el resto de experimentos se harán tan solo entre A* y búsqueda uniforme.

Finalmente, también exploramos más el uso de la memoria visualizando el número de nodos expandidos y el número máximo de nodos en frontera en Sudokus con un nivel de dificultad 0.73, la figura 6 muestra la gran diferencia entre ambos algoritmos. De donde podemos deducir el gran ahorro de memoria que también se obtiene al utilizar A*.



(a) (b)

Figura 6: Comparación entre A* y búsqueda uniforme en (a)nodos expandidos y (b)número máximo de nodos en frontera.

6. Conclusiones

- Se ha demostrado que el uso de heurísticas en agentes de planificación en problemas de búsqueda significa una gran ventaja a nivel computacional, tanto en tiempo como en memoria.
- El algoritmo A* con la heurística basada en mínimos valores restantes logra resolver el Sudoku clásico (9×9) en tiempos razonables para todos los niveles de dificultad.
    - Además, se observa que la calidad de la heurística tiene un impacto directo en la eficiencia del agente. Una heurística más informada no solo reduce el número de nodos explorados, sino que orienta la búsqueda hacia regiones más prometedoras del espacio de estados.
- A pesar de los buenos resultados obtenidos, el enfoque sigue siendo sensible al tamaño del tablero. Extender el modelo a Sudokus mayores implicaría un crecimiento exponencial en el espacio de búsqueda, lo que requeriría estrategias adicionales de poda o paralelización.

Referencias

[1] R. Béjar, C. Fernández, C. Mateu, and M. Valls. The sudoku completion problem with rectangular hole pattern is np-complete. *Discrete Mathematics*, 312(22):3306–3315, 2012.

[2] F. S. Caparrini. Blog, URL: https://www.cs.us.es/ fsancho/Blog/posts/CSP.md.html.

[3] L. Chaimowicz and M. Machado. Combining metaheuristics and csp algorithms to solve sudoku. games and digital entertainment (sbgames). In *2011 Brazilian Symposium on*, pages 124–131, 2011.

[4] Z. W. Geem. Harmony search algorithm for solving sudoku. In *International conference on knowledge-based and intelligent information and engineering systems*, pages 371–378. Springer, 2007.

[5] J. Mangwani and P. Prateek. Using progressive stochastic search to solve sudoku csp. *PRATEEK*, 2012.

[6] E. Onokpasa, D. Bisandu, and D. Bakwa. A hybrid backtracking and pencil and paper sudoku solver. *International Journal of Computer Applications*, 2019.

[7] R. Sonmez and O. H. Bettemir. A hybrid genetic algorithm for the discrete time–cost trade-off problem. *Expert Systems with Applications*, 39(13):11428–11434, Oct. 2012.