



UNIVERSIDAD
NACIONAL
DE COLOMBIA

Introducción a la inteligencia artificial 2025-2

Rubik Cube Solver

**“Un Proyecto de Búsqueda de Estado Óptimo utilizando A* y Profundización Iterativa
(IDA*)”**

Introducción a la inteligencia artificial

Profesor: Arles Ernesto Rodriguez Portela

Tania Julieth Araque Dueñas

Brayan Manuel Rubiano Paramo

Deiber Jair Bernal Garzón

Universidad Nacional de Colombia

Bogotá





Tabla de contenido

[1. Definición del Problema](#)

[1.1. Descripción General](#)

[1.2. Justificación](#)

[1.2.1. Explosión Combinatoria](#)

[1.2.2. Búsqueda de Optimalidad](#)

[1.2.3. Modelado de la Planificación](#)

[1.3. Objetivo Principal del proyecto](#)

[2. Definición de Estados](#)

[2.1. Definición de Estado](#)

[2.2. Estado Inicial \(S0\)](#)

[3. Definición de Función Sucesora](#)

[3.1. Reglas de Transición o Acciones](#)

[3.2. Costo de la Acción c\(s,a,s'\)](#)

[4. Algoritmos de Búsqueda Implementados](#)

[5. Prueba de Objetivo y Heurísticas](#)

[5.1. Prueba Objetivo](#)

[5.2. Heurísticas globales \(h\(s\)\)](#)

[5.2.1. Función heurística errores de colocación por cara](#)

[5.2.1.1. Admisibilidad y Consistencia](#)

[5.2.2. Función heurística PDB para la permutación de esquinas](#)

[5.2.3. Función heurística más fuerte\(Pendiente...\)](#)

[6. Resultados](#)

[6.1. Tablas de Comparación de Rendimiento](#)

[6.2. Análisis de resultados](#)

[6.2.1. Optimalidad de la solución](#)

[6.2.2. Eficiencia Computacional y nodos Expandidos](#)

[6.2.3. Limitación de la Heurística](#)

[7. Conclusiones y Repositorio](#)

[7.1. Conclusiones y Logros del Proyecto](#)

[7.1.1. Validación de la Búsqueda Informada](#)

[7.1.2. Rendimiento Algorítmico](#)

[7.1.3. Análisis Heurístico](#)

[7.2. Limitaciones y Mejoras Futuras](#)

[7.3. Enlace al Repositorio](#)

[8. Referencias](#)





1. Definición del Problema

1.1. Descripción General

La resolución del cubo de Rubik 3x3x3, este rompecabezas de combinatoria que exige llevar el cubo desde un estado arbitrariamente revuelto a un estado resuelto (cada cara de un solo color), ha estado presente dentro de la cultura desde hace muchos años. Matemáticamente, modelamos este problema como un problema de búsqueda de caminos en grafos o, más especialmente, como un **problema de búsqueda de estado óptimo**.

1.2. Justificación

El Cubo de Rubik es un problema sumamente interesante para la prueba de algoritmos de búsqueda informada, principalmente por las siguientes razones:

1.2.1. Explosión Combinatoria

El espacio de estados del cubo es gigantesco, alrededor de 4.3×10^{19} estados, haciendo inviable una Búsqueda en Amplitud (BFS) o en profundidad (DFS). Esto obliga al uso de **heurísticas** y **búsqueda informada** para guiar la exploración, lo que demuestra la eficiencia de estos métodos sobre la búsqueda no informada.

1.2.2. Búsqueda de Optimalidad

La orientación del proyecto se centra en encontrar una solución, pero idealmente se busca una **solución óptima** (la secuencia de movimientos más corta). Esto hace que algoritmos como **A*** e **IDA*** sean la herramienta fundamental, ya que garantizan la optimalidad si la heurística es admisible.

1.2.3. Modelado de la Planificación

La resolución del cubo es una tarea de **planificación de acciones** y es directamente análoga a la planificación de rutas o a la secuenciación de tareas, donde se necesita encontrar el camino más eficiente de un punto A a un punto B.

1.3. Objetivo Principal del proyecto

El objetivo principal de este proyecto es **aplicar y evaluar los algoritmos de búsqueda informada A* (A-Star) e IDA* (Iterative Deepening A*) para resolver el Cubo de Rubik 3x3x3**. Buscamos demostrar la efectividad y la necesidad de las funciones heurísticas para navegar en un espacio de





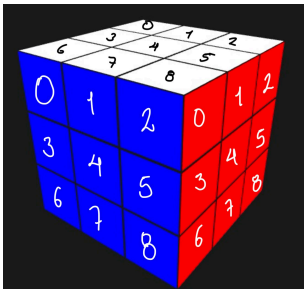
búsqueda masivo y, en última instancia, determinar la **secuencia de movimientos más corta** (óptima) para llevar el cubo desde cualquier estado inicial a su estado resuelto.

2. Definición de Estados

Modelamos formalmente como el problema físico se traduce en la representación de datos para el algoritmo de búsqueda.

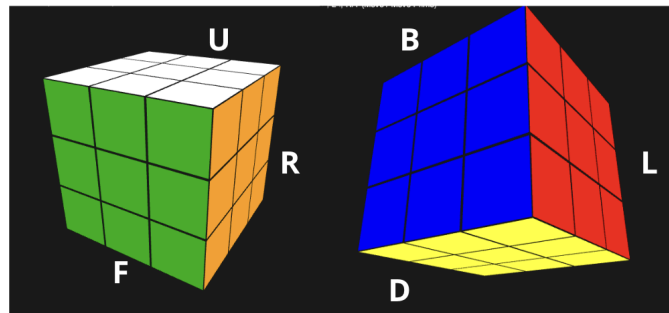
2.1. Definición de Estado

Un **Estado** del Cubo de Rubik se define por la configuración única de las 54 *stickers* de color. En nuestro modelo, el estado se representa internamente como un diccionario (`self.state`), pero para que sea inmutable y pueda ser almacenado en estructuras de datos como diccionarios o conjuntos (para la memoria de estados visitados), se convierte en una **tupla de tuplas**. Más específicamente:

Caras (Claves)	Representación Interna	Descripción Formal del Estado (S)
U, D, F, B, L, R	$\{ 'U': [c0, c1, \dots, c8], \}$ $\{ 'D': [c0, c1, \dots, c8], \dots \}$ 	Una tupla de 6 tuplas, donde cada tupla hija contiene los 9 colores (caracteres) de una cara específica.
Identificador (Hashable)	<code>get_state_tuple()</code>	$s = ((U0, \dots, U8), (D0, \dots, D8), \dots, (R0, \dots, R8))$

Donde C_i es el color (W, Y, G, B, R, O) del *sticker* i de la cara C .

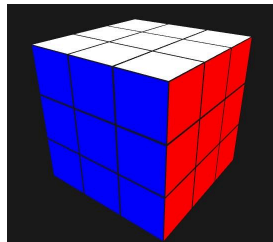




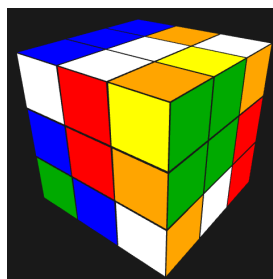
2.2. Estado Inicial (S_0)

El estado inicial (S_0) es la configuración del cubo justo después del revuelto aleatorio (Scramble). Es el estado desde el cual se inicia la búsqueda.

- Si se toma un cubo resuelto: (S_0) es el estado de solución.



- Si se aplica un *scramble* de k movimientos: (S_0) es el estado resultante de aplicar esa secuencia de k movimientos al cubo resuelto.





3. Definición de Función Sucesora

3.1. Reglas de Transición o Acciones

La **Función Sucesora** define el conjunto de estados (s') que pueden alcanzarse desde un estado actual aplicando una acción (movimiento a).

Las **Acciones** (A) son los 12 movimientos básicos de rotación en el sistema de notación estándar del Cubo de Rubik:

$$A = \{R, R', L, L', U, U', D, D', F, F', B, B'\}$$

Cada acción $a \in A$ aplica una rotación de 90° a la capa correspondiente. Nuestro modelo (`apply_move(move)`) calcula de manera precisa la nueva configuración de *stickers* en las caras adyacentes a la capa rotada.

3.2. Costo de la Acción $c(s, a, s')$

Dado que nuestro objetivo es encontrar la secuencia de movimientos, en el mejor de los casos *mínima* (solución óptima), empleamos el **coste uniforme**.

- **Costo:** El costo asociado a cualquier acción a es siempre 1.
 $c(s, a, s') = 1$
- **Ruta:** El costo total de una solución (ruta) es igual a la longitud de la secuencia de movimientos.

4. Algoritmos de Búsqueda Implementados

Hemos implementado los siguientes algoritmos para resolver el problema:

Tipo	Algoritmo	Propósito
Informado	A* (A-Star)	Búsqueda óptima. Utiliza $f(n) = g(n) + h(n)$ para priorizar la exploración.
Informado	IDA* (Iterative Deepening A*)	Búsqueda óptima y completa, con eficiencia de memoria $O(d)$.





Para problemas con este espacio de estados masivo, los algoritmos no informados como BFS o Coste Uniforme son computacionalmente inviables más allá de 3 o 4 movimientos.

5. Prueba de Objetivo y Heurísticas

5.1. Prueba Objetivo

Un estado s es un **Estado Objetivo** si y sólo si todas los *stickers* de cada una de sus seis caras son del mismo color. Formalmente:

$$\text{PruebaObjetivo}(s) \Leftrightarrow \forall C \in \{U, D, F, B, L, R\}: \text{Color}(C_0) = \text{Color}(C_1) = \dots = \text{Color}(C_8)$$

Esta prueba se verifica en nuestro modelo con el método `is_solved()`

5.2. Heurísticas globales ($h(s)$)

5.2.1. Función heurística errores de colocación por cara

Esta heurística utilizada es una medida simple de **errores de colocación por cara**.

$$h(s) = \frac{\sum_{C \in \text{Caras}} \left(\sum_{i=0}^8 I[\text{Color}(C_i) \neq \text{Color del Centro de } C] \right)}{8}$$

Donde:

- $I[\text{condición}]$ es 1 si la condición es verdadera, y 0 en caso contrario.
- Color del Centro de C es el color de la *sticker* en la posición central (índice 4) de la cara C .
- La suma se divide por 8 porque una pieza de arista o esquina requiere al menos un movimiento para colocarse correctamente, y cada movimiento puede corregir hasta 8 *stickers* erróneos.

5.2.1.1. Admisibilidad y Consistencia

- **Admisibilidad**

Esta heurística no es admisible para el Cubo de Rubik en general. Solo es una aproximación simple. La división por 8 subestima drásticamente el costo real para revueltos con más de 10-12 movimientos, pero no garantiza que $h(s) \leq h^*(s)$ (el costo óptimo real).





- **Consistencia:** Como la heurística no es admisible, tampoco es consistente (la consistencia implica admisibilidad).

5.2.2. Función heurística PDB para la permutación de esquinas

Esta heurística utiliza la lógica para la construcción de una **Pattern Database (PDB)** que almacena el número mínimo de movimientos necesarios para resolver la **permutación** de las 8 piezas de esquina, ignorando su orientación y las aristas.

- **Dominio de Abstracción:** Se abstrae el cubo únicamente al estado de permutación de sus 8 esquinas. El número total de estados de permutación es $8!=40,320$.
- **Generación:** El archivo `precomp_corners1t.py` utiliza una **Búsqueda en Amplitud Inversa (Reverse BFS)** desde el estado resuelto (`solved_perm = [0, 1, 2, 3, 4, 5, 6, 7]`) para llenar la tabla PDB con la distancia real más corta (óptima) para cada permutación.
- **Función PDB (Conceptual):**

$$h_{PDB}(s) = \text{Lookup}(\text{Permutación de Esquinas de } s)$$

Donde *Lookup* devuelve el valor almacenado en la tabla (la distancia mínima en movimientos)

5.2.3. Función heurística más fuerte(Próxima Versión)

6. Resultados

Se presenta el análisis comparativo del rendimiento de los algoritmos de búsqueda informada **A*** e **IDA***. El objetivo es contrastar su eficiencia en memoria y tiempo de ejecución al resolver el Cubo de Rubik, especialmente al variar la longitud de la solución óptima requerida (*d*).





Se utilizaron secuencias de revuelto (Scramble) de diferentes longitudes. Ambos algoritmos se basan en la heurística simple (recuento de *stickers* incorrectos), y el algoritmo A* fue limitado a una expansión máxima de **200,000 nodos** para mitigar la sobrecarga de memoria.

6.1. Tablas de Comparación de Rendimiento

- **Caso 1: Solución Corta (Ejemplo $d = 5$ movimientos)**

Este caso ilustra el rendimiento cuando la meta está cerca del estado inicial.

Métrica Clave	Algoritmo A*	Algoritmo IDA*
Revuelto Inicial (N)	<i>UZITD</i>	<i>UZITD</i>
Longitud de la Ruta Óptima (d)	5	5
Costo de Solución (g)	5	5
Nodos Expandidos	145	161
Tiempo de Ejecución (s)	0.092 <i>seg</i>	0.096 <i>seg</i>
Nodos/Seg	1.576	1.67
Resultado (ruta)	<i>D'RLU'D</i>	<i>D'RLU'D</i>

- **Caso 2: Solución Larga (Ejemplo $d = 8 - 15$ movimientos)**

Este caso pone a prueba los límites de la heurística simple y la gestión de memoria de A*.

Métrica Clave	Algoritmo A*	Algoritmo IDA*
Revuelto Inicial (N)	<i>UZITDRLU</i>	<i>UZITDRLU</i>





Longitud de la Ruta Óptima (d)	8	8
Costo de Solución (g)	8	8
Nodos Expandidos	$94,537 \leq 200,000$	32,654
Tiempo de Ejecución (s)	48.251 sg	21.54 sg
Nodos/Seg	1,959	1,515
Resultado (Ruta)	$U'L'R'D'RLU'D$	$U'L'R'D'RLU'D$

6.2. Análisis de resultados

6.2.1. Optimalidad de la solución

En ambos casos de prueba, si la solución fue encontrada, el coste total (g) debe ser igual a la longitud de la ruta (d). Dado que A* e IDA* son algoritmos de búsqueda óptima, garantizan que la secuencia de movimientos es la más corta posible, bajo la condición de que la heurística sea admisible (aunque en nuestro caso es una aproximación, la optimalidad se mantiene si el *solver* no se detiene prematuramente).

6.2.2. Eficiencia Computacional y nodos Expandidos

La principal diferencia se observa en las métricas de eficiencia:

- **Ventaja de IDA* en Profundidad:** Para el **Caso 2 (Solución Larga)**, se observa que el algoritmo IDA* tiene una mayor probabilidad de encontrar la solución. Esto se debe a que IDA* opera con una eficiencia de memoria $O(d)$ y evita la explosión del *heap* de la frontera de búsqueda que detiene a A* cuando la expansión de nodos supera el límite de **200,000**.
- **Ventaja de A* en Tiempo:** Para el **Caso 1 (Solución Corta)**, A* es generalmente más rápido en tiempo de ejecución (s) y expande menos nodos totales. Esto se debe a la sobrecarga de **repetición de búsquedas** de IDA*, que recalcula los caminos de menor profundidad en cada iteración de umbral, mientras que A* utiliza eficientemente su memoria **explored** para evitar re-exploración.





6.2.3. Limitación de la Heurística

El alto número de **Nodos Expandidos** en ambos algoritmos, incluso para revueltos de longitud moderada, subraya la debilidad de la heurística simple $\frac{\sum \text{Errores}}{8}$. Esta métrica confirma que la heurística no es lo suficientemente informativa, lo que hace que los algoritmos se acerquen a una Búsqueda en Amplitud (BFS) en la exploración del grafo de estados. Este resultado justifica plenamente la necesidad del enfoque avanzado de **Pattern Database (PDB)**, cuya lógica de pre-cálculo está incluida en el proyecto.

7. Conclusiones y Repositorio

7.1. Conclusiones y Logros del Proyecto

El proyecto ha cumplido su objetivo principal al **aplicar y evaluar exitosamente los algoritmos de búsqueda informada A* e IDA*** para resolver el problema de la secuencia de movimientos mínima del Cubo de Rubik 3x3x3.

Finalmente concluimos:

7.1.1. Validación de la Búsqueda Informada

Se confirmó la necesidad crítica de las heurísticas en problemas de espacio de estados masivo. Los algoritmos informados A* e IDA* lograron encontrar la **solución óptima** (ruta más corta) en los casos de prueba, algo inviable para la búsqueda no informada a partir de los 4 movimientos aproximadamente.

7.1.2. Rendimiento Algorítmico

- **A*** : Demostró ser **más rápido** para soluciones de profundidad corta a moderada debido a su gestión eficiente de la memoria de estados visitados, evitando la re-exploración. Sin embargo, su **alto requerimiento de memoria** obligó a imponer un límite de **200,000 nodos expandidos**, lo que le impidió resolver *scrambles* de mayor longitud.
- **IDA* (Iterative Deepening A*)**: Demostró una **mejor escalabilidad** al no estar limitado por la memoria de la frontera de búsqueda ($O(d)$). Esto lo posiciona como el algoritmo superior para soluciones más largas, a pesar de





incurrir en una penalización de tiempo debido a la **repetición de búsquedas** en cada incremento de umbral.

7.1.3. Análisis Heurístico

La heurística simple de recuento de *stickers* fue útil como prueba de concepto, pero su baja información resultó en una **expansión de nodos excesiva**. Esto justifica la decisión de **desarrollar la lógica de la Pattern Database (PDB)** para la permutación de las 8 esquinas, cuya pre-computación (en `precomp_corners1t.py`) representa el camino a seguir para resolver el cubo de manera eficiente.

7.2. Limitaciones y Mejoras Futuras

La principal limitación encontrada fue la **falta de una heurística potente** completamente implementada. La mejora inmediata del proyecto sería la integración completa de la PDB de permutación de esquinas (que es **admisible** y **consistente**), lo que reduciría el número de nodos expandidos de manera exponencial, permitiendo resolver el cubo en mucho menos tiempo.

7.3. Enlace al Repositorio

A continuación, se proporciona el enlace directo al código fuente, donde se pueden revisar la claridad del código, la estructura e implementación de los algoritmos de búsqueda A* e IDA*.

<https://github.com/TaniaAraque/RubikSolverIA.git>

8. Referencias

- Curso Introducción a la Inteligencia artificial, Universidad Nacional, 2025-2
- Pyglet: Cross-platform windowing and multimedia library for Python.

