UNIVERSIDAD SIMÓN BOLÍVAR

DEPARTAMENTO DE COMPUTACIÓN Y TECNOLOGÍA DE LA INFORMACIÓN

CI5437 – INTELIGENCIA ARTIFICIAL I

Trimestre: Enero – Marzo 2024

Profesor: Carlos Infante

Informe del Proyecto #1

Implementación de Algoritmos de búsqueda de costo mínimo para la Resolución de Problemas jugables

Estudiantes:

José Matías González, 15-10627

Ana Santos, 17-10602

Sartenejas, Febrero del 2023

ÍNDICE

DEFINICIÓN DEL PROBLEMA

El modelo de espacios de estados es una representación matemática para un conjunto de variables parte de un sistema físico con entrada, salida y acciones determinantes de los posibles cambios dentro del sistema. En el mundo de la inteligencia artificial, este modelado hace referencia a la búsqueda de una configuración computacional para encontrar la serie de pasos que llevan a las salidas deseadas, es decir, el desarrollo de una solución con la definición del espacio de estado, las funciones y procedimientos necesarias para resolver el problema.

Por ello, se les solicita a los estudiantes del curso de Inteligencia Artificial I implementar algoritmos de búsqueda para resolver problemas relacionados con juegos sencillos, a través de la comprensión del modelado de espacios de estado y las heurísticas aplicables a estos algoritmos. Uno de los objetivos principales de este proyecto, es realizar implementaciones correctas, eficientes y que cumplan los tiempos propuestos estipulados.

En cuanto a los objetivos más específicos, se quiere analizar la diferencia entre los algoritmos de búsqueda sin y con eliminación de duplicados de ancestros comparando los resultados de nodos abiertos y la profundidad de la solución. También, se desea implementar heurísticas basadas en bases de datos de patrones (PDBs) para resolver los problemas de manera aditiva para los N-puzzle y con los demás problemas se pueden hacer abstracciones Finalmente, se quiere resolver cada uno de los problemas de juegos usando algoritmos informados como A* para la eliminación retardad de duplicados e IDA* con eliminación parcial de duplicados.

Ahora, se dará una pequeña descripción general de los problemas que se desean resolver a través del modelo de espacios de estado:

- N-Puzzle: este problema consiste en un rompecabezas deslizante que involucra mover la piezas hacia el espacio vacío hasta alcanzar una configuración objetivo.
- Torres de Hanoi: este problema es un rompecabezas que consisten en algunas torres y un número de discos de diferentes tamaños que pueden deslizarse en cualquiera de

las torres, sabiendo que solo se puede mover un disco a la vez y que ningún disco puede ser colocado encima de un disco de menor tamaño.

- Cubo de Rubik: este consiste en un cubo don caras de diferentes colores que pueden girar alrededor de un eje central, buscando girar y manipular las piezas hasta que cada cara del cubo sea de un mismo color por cara.
- Top Spin: este problema tiene un sistema parecido al N-puzzle, ya que contiene piezas deslizantes que deben ser ordenadas, nada más que las acciones de configurar son rotar hacia los lados y girar la pieza central.

METODOLOGÍA

Es importan hacer mención del uso de PSVN, como un recurso de definición de estados en el lenguaje C, el cual crea las estructuras de los espacios de estados a cada problema a partir de valores, estados, acciones y metas determinados. Este programa permite, también, realizar la implementación de funciones necesarias para trabajar con los estados y las acciones en cualquier algoritmo que se desee diseñar.

Para poder completar los objetivos planteados en el proyecto, se escogió entre los algoritmos de búsqueda dados en clase para la resolución de los problemas según los objetivos planteados.

Finalmente, se diseñaron una serie de *makefile* y *scripts* para poder acoplar los algoritmos de búsqueda, las estructuras del modelo de espacio de estados y los casos de prueba.

1. Implementación de los Algoritmos de Búsqueda

De que se tratan con foto de las implementaciones principales

a. BFS

El algoritmo de búsqueda en profundidad se implementó como método de búsqueda sin eliminación de duplicados ancestros, por lo que es importante hacer énfasis en que este algoritmo tiene una complejidad de tiempo $O\ (v + e)$, considerando los *v* como la cantidad de vértices y la cantidad de aristas, *e*.

b. BRS with prunning

El algoritmo BFS con poda parcial aplica una modificación al código original donde se implementan colores para evitar revisar los mismos caminos más de una vez en el árbol, por lo que existe una mejora en la complejidad de este algoritmo con respecto al anterior, ya que depende únicamente de la cantidad de vértices, $O\ (v)$.

c. UCS

Se realizó esta implementación esta modificación a BFS con una eliminación tardía de duplicados de costos uniformes, teniendo una complejidad $O(b^{C^*/\epsilon})$, tal que b es un factor de ramificación, C^* es el costo del camino óptimo y ϵ el costo mínimo de cualquier paso. Siendo una mejora en complejidad de tiempo en comparación de los algoritmos anteriores.

La decisión de implementar este algoritmo se debió a que los algoritmos de A* e IDA* no supo completar ninguna corrida óptima entonces se quiso mostrar más material de comparación en la resolución de los problemas.

d. A* e IDA*:

El algoritmo A* se usa para la búsqueda en grafos, verificando tanto la profundidad como el costo de los caminos. Mientras que IDA*, es una implementación del anterior con limitación de memoria, de manera que se profundiza en el grado iterativamente. Realmente, estos algoritmos tienen una mayor complejidad de espacio en comparación a BFS, pero representan una gran ventaja en la evaluación de precisa de los costes del camino y la búsqueda de soluciones.

Estos algoritmos esos los llamados informados ya que necesitan de funciones heurísticas para el cálculo del costo del camino hacia el objetivos. En el caso de este proyecto, se intentaron desarrollar el calculo de heurísticas con PDB para 15 y 24 puzzle.

2. Especificaciones del Equipo usado

Todas las pruebas se corrieron en el mismo equipo, el cual cuenta con las siguientes especificaciones:

- Procesador: Intel i5-12500H, 12 Cores.
- RAM: 64GB, 3100MHz.
- SSD

RESULTADOS Y ANÁLISIS

1. N-Puzzle

El problema de N-Puzzle que se pudo ejecutar fue el de 15 puzzle para el algoritmo de búsqueda *UCS* debido a que los demás algoritmos ocasionaban fugas de memorias, lo cual provocaba que Linux matará el proceso. Para este algoritmo, sólo se pudieron correr 11 de las 100 instancias de prueba. Además, todas fueron interrumpidas a los 15 minutos por el procedimiento del algoritmo. No obstante, se obtuvo un promedio de 65.188.549,08 nodos abiertos durante la búsqueda de solución de estos 15 puzzle.

2. Torres de Hanoi

En el caso de las torres de Hanoi, se debían ordenar en cuatro (4) torres, 12, 14 y 18 discos con casos promedios con soluciones de profundidad de cinco (5), diez (10), 15, 2000 y 2000000 de nodos.

Al aplicar los algoritmos de búsqueda *UCS, BFS, BFS with prunning* para el caso de ordenamiento de 12 discos, se obtuvó los resultados visualizados en el Gráfico 1. En particular, los archivos de los casos grandes como 2000 y 2000000 nodos no arrojaron resultados para los casos de *BFS* y *BFS with prunnig* porque el terminal de Linux captaba una posible fuga de memoria y terminaba el proceso.

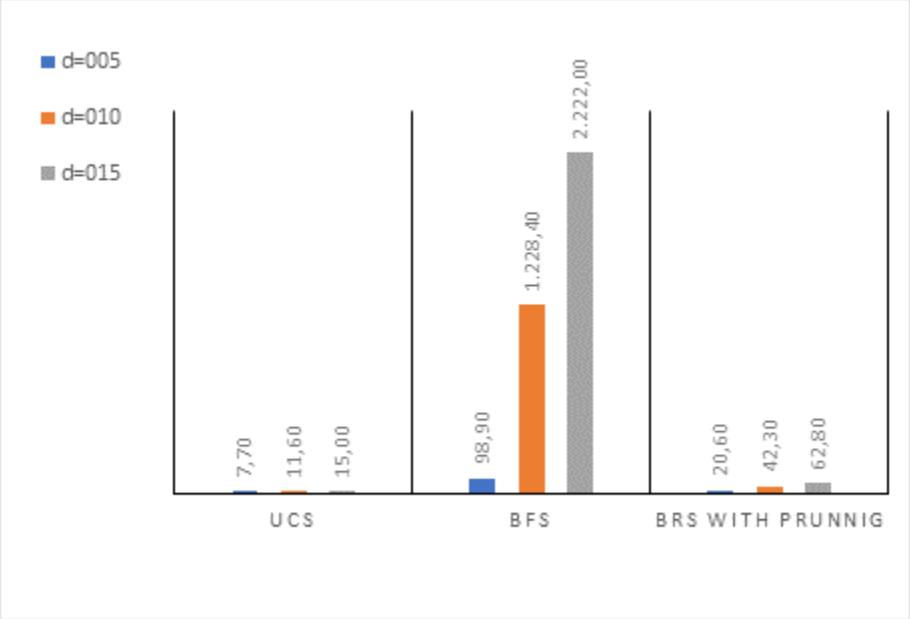

Gráfico 1. *Resultados promedios para 4 torres y 12 discos*

Se pudo apreciar que el tiempo promedio en milisegundos para estos tres (3) casos representados en la gráfica fueron insignificantes, demostrando que para casos de búsquedas cortas los tres (3) algoritmos de búsqueda son igualmente de eficientes. Por otra parte, si se habla de la complejidad del espacio, se observó que *BFS* es altamente ineficiente ya que verifica muchísimos más nodos que los otros algoritmos de búsqueda aplicados

Se destaca la eficiencia del algoritmo de *UCS* tanto en tiempo como en espacio, ya que se pudieron hacer pruebas para casos grandes como se observa en el Gráfico 2.

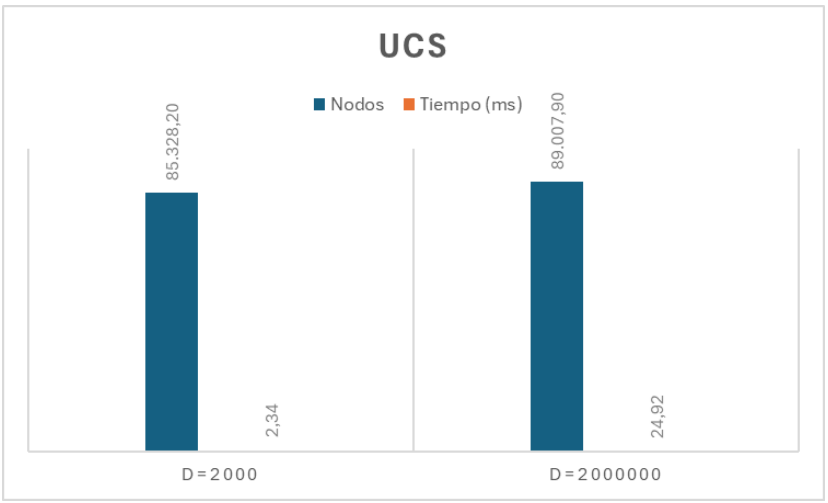

Gráfico 2. *Resultados promedios en UCS para 4 torres y 12 discos para casos grandes*

Al aplicar los algoritmos de búsqueda *UCS*, *BFS*, *BFS with prunning* para el caso de ordenamiento de 14 discos, se obtuvó los resultados visualizados en el Gráfico 3. En particular, los archivos de los casos grandes como 2000 y 2000000 nodos no arrojaron resultados para los casos de *BFS* y *BFS with prunnig* por lo explicado anteriormente.

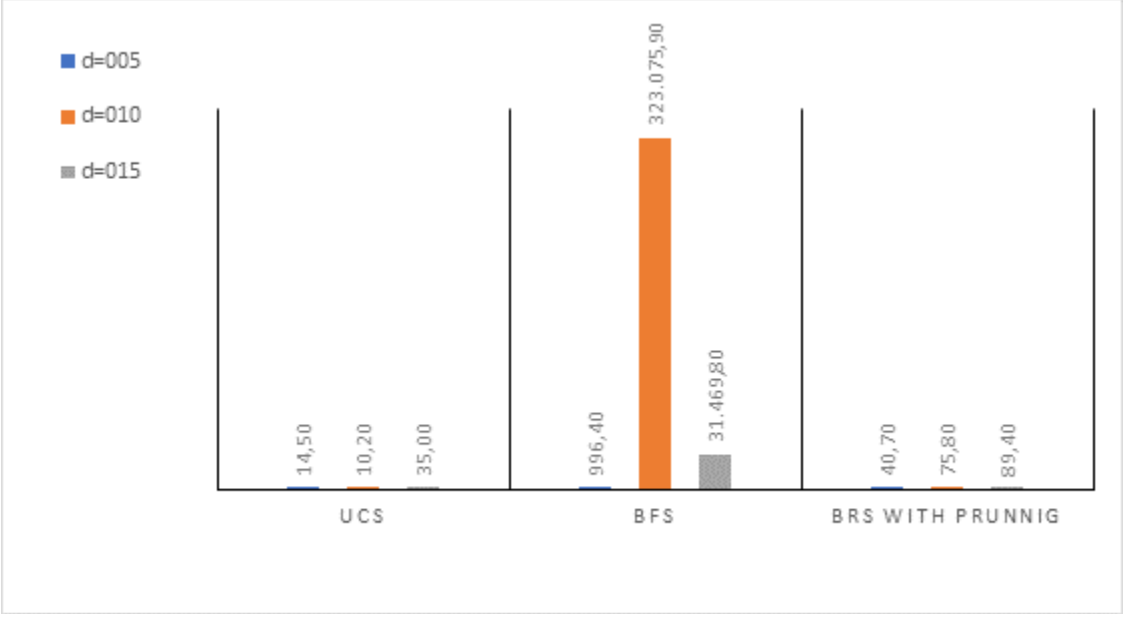

Gráfico 3. *Resultados promedios para 4 torres y 14 discos*

Dentro de los datos resaltantes, se pudo apreciar que el tiempo promedio en milisegundos de nuevo fueron insignificantes. Se destaca la eficiencia del algoritmo de *UCS* ya que tuvo que interrumpir uno de las búsquedas a los 15 minutos, ya que no había encontrado solución y encontró que uno de los casos no tenía solución. Esto último, se observó en el archivo de prueba para 2000000 nodos.

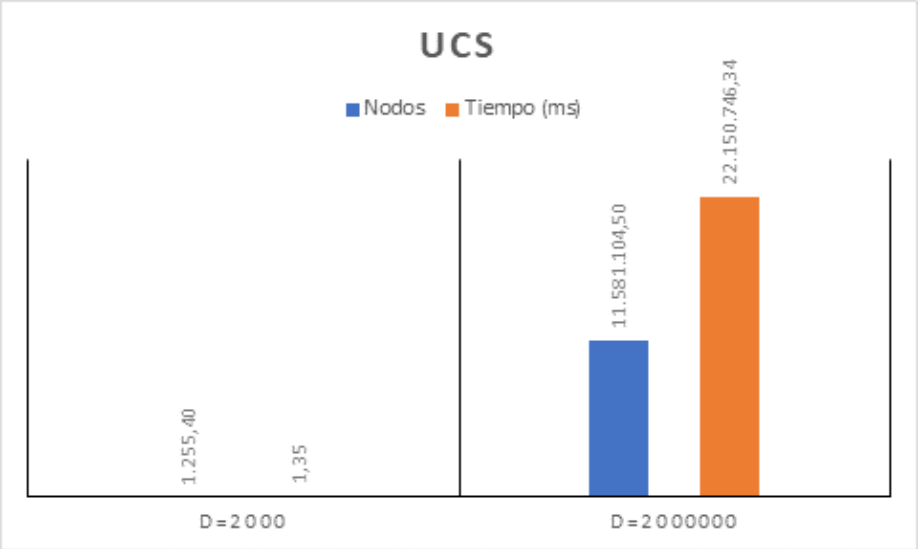

Gráfico 4. *Resultados promedios en UCS para 4 torres y 14 discos para casos grandes*

Como se puede observar para el caso de 18 discos en las torres de Hanoi, que los resultados fueron similares a los casos anteriores, lo que muestra la eficiencia *UCS* como algoritmo de búsqueda para este problema y que *BFS* al no eliminar los duplicados representa un fuga de memoria importante ocasionando que tanto el sistema operativo como el parámetros de lenguaje usado puedan terminar con las corridas de prueba.

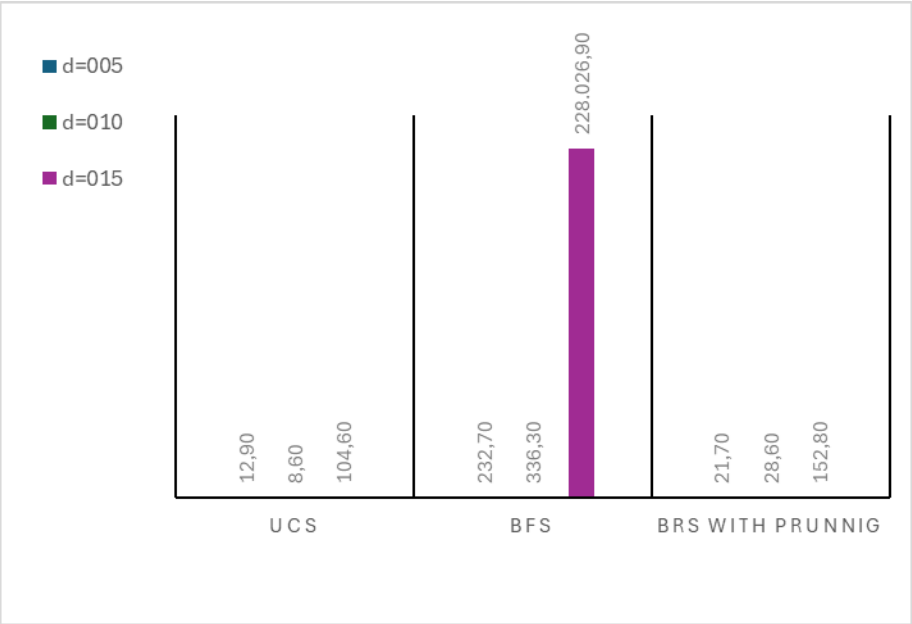

Gráfico 5. *Resultados promedios para 4 torres y 14 discos*

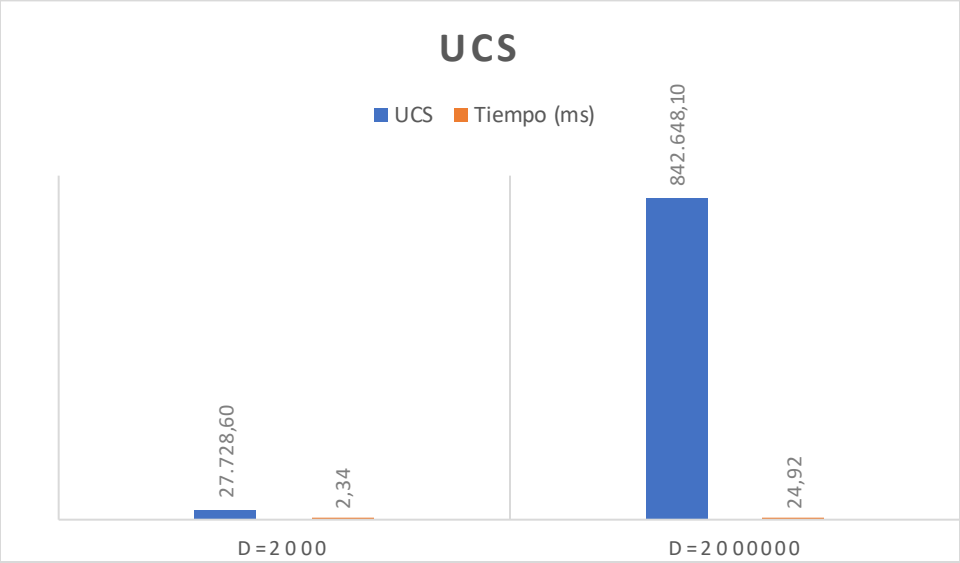

Gráfico 6. *Resultados promedios en UCS para 4 torres y 12 discos para casos grandes*

3. Cubo de Rubik

El problema del cubo de Rubik debía usar casos de prueba para soluciones con una profundidad de cinco (5), diez (10), 15, 20, 40, 80 y 160 nodos. Pero debido a las

implementaciones usadas, sólo se pudo realizar las pruebas con el primer caso sin tener fugas de memoria.

Ahora, viendo el Gráfico 7, se observa que aunque *BFS* abrió sustancialmente muchos más nodos que los otros algoritmos, fue el que obtuvo la solución mucho más rápido. Con ello, se puede concluir que al este algoritmo estar centrado en la búsqueda de la solución de manera más corta sin considerar pasos para evitar la doble verificación, en este problema, dan un resultado más optimo en tiempo.

Lamentablemente al no poder correr los otros casos, no se puede contrastar esta información.

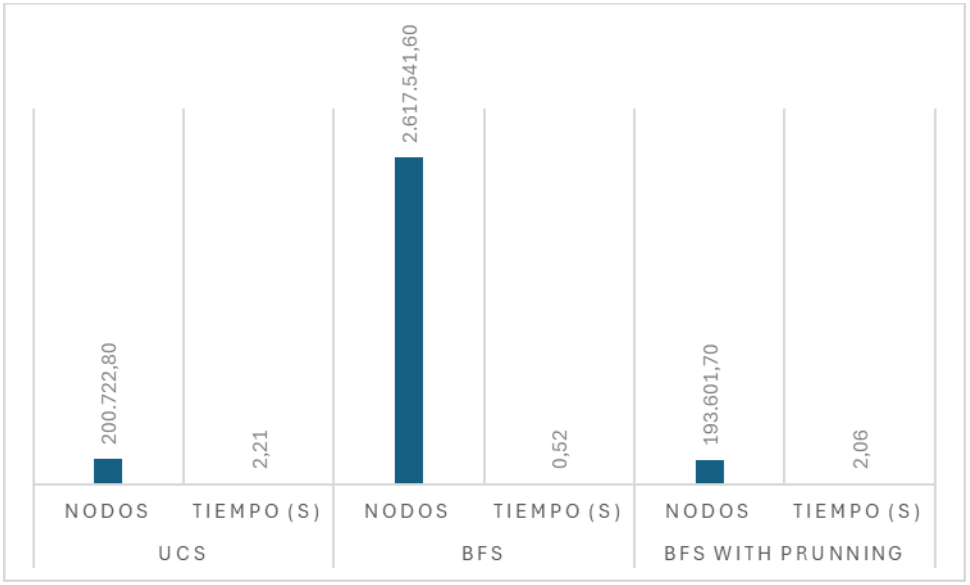

Gráfico 7. *Resultados promedios para un cubo de Rubik 3x3x3*

4. Top Spin

En el caso particular de los problemas de Top Spin, se evaluaron los casos para 12, 14 y 17 números a ordenar con 4 en la pieza movible. De manera general, se observó que los algoritmos no pudieron correr para ninguno de los casos de solución a 2000 nodos y 2000000 nodos de profundidad.

Como se puede observar en el Gráfico 8, los problemas se resuelven realmente rápido para casos pequeños pero el crecimiento en la revisión de nodos al aumentar la dificultad es exponencial. Cabe destacar que los algoritmos de *BFS* y *BFS with prunning*, sólo corrieron el caso más sencillo obteniendo un promedio de 269.468 y 46.869 nodos abietos.

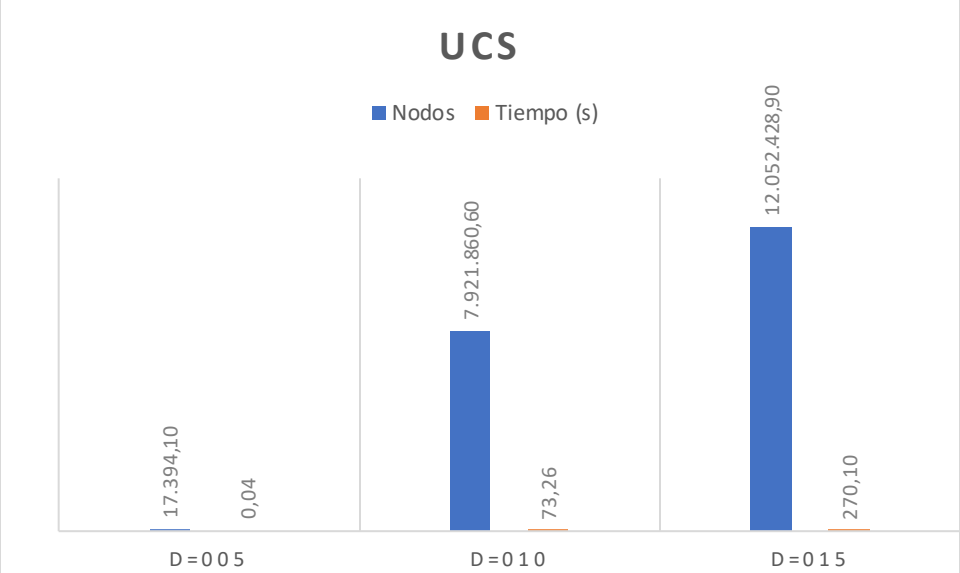

Gráfico 8. *Resultados promedios en UCS para 12 números*

Para el caso de ordenamiento de 14 números, con *UCS* se lograron obtener resultados para los dos primeros casos, es decir, para soluciones de 5 y 10 nodos de profundidad, donde se obtuvo 29.298,80 y 14.216.880,57 nodos abiertos en promedio respectivamente. En profundidad 10, se obtuvo un único caso con solución mientras que los demás fueron interrumpidos. Los algoritmos de *BFS y BFS with prunning* no completaron las corridas de prueba.

Finalmente, en los casos de ordenamiento de 17 números ocurrió lo mismo para *BFS y BFS with prunning*. Asimismo, se encontró para los casos de profundidad 5 y 10 al realizar la búsqueda con *UCS* un promedio en los nodos abiertos de 56.792 y 34.491.583,60, respectivamente. Nuevamente, para la profundidad 10, sólo se halló una solución con 4.813.979 nodos abiertos.

DIFICULTADES EN LA IMPLEMENTACIÓN

Dentro de las dificultades encontradas a la hora de implementar este proyecto, se encuentran las siguientes:

- El equipo está trabajando con un lenguaje de programación desconocido, lo que representa un desafío.
- Han progresado utilizando *psvn,* una herramienta que facilita tareas como la abstracción y las funciones de estados.
- Sin embargo, el uso de *psvn* ha ralentizado su desarrollo debido a su curva de aprendizaje.
- A pesar de las semanas de trabajo, el funcionamiento de *psvn* aún les parece misterioso en ciertos aspectos
- El equipo enfrentó desafíos debido a la complejidad de los problemas y las instancias de prueba.
- La ejecución de los algoritmos fue difícil, incluso con un equipo potente.
- Los problemas generaron un gran número de estados, hasta casi 100 millones, lo que prolongó los tiempos de prueba.
- Experimentaron problemas con el intérprete de C++, que terminó el proceso de la terminal varias veces debido al consumo excesivo de memoria.

FE DE ERRATA: Los tiempos en la gráfica de torres de Hanoi están expresados en segundo y no en ms como se muestra en la leyenda