



Métodos de Búsqueda

Integrantes

*Juan Pablo Oriana - 60621
Tomas Cerdeira - 60051
Santiago Garcia Montagner - 60352*



Objetivos del trabajo

- Crear un programa que implemente los métodos de búsqueda visto en clase para solucionar, de ser posible, un rompecabezas de 8 números.
- Analizar las diferencias en los resultados que presentan los métodos de búsqueda informados de los no informados
- Implementar 3 heurísticas distintas y analizar su rendimiento. Dos de ellas deben ser admisibles y la otra no admisible



Reglas del juego

Reglas del juego

W. E. Story, Note on the "15" puzzle, Amer. J. Math. 2 (1879) 399-404

El estado inicial debe ser un estado posible

1	8	2
	4	3
7	6	5

Estado posible

8	1	2
	4	3
7	6	5

Estado imposible

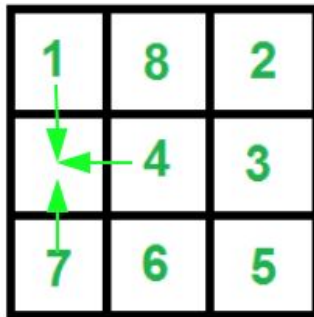
Observación: La cantidad de estados posibles para este problema es $9!/2$

Reglas del juego



Movimientos

- Las piezas solo pueden desplazarse horizontal o verticalmente
- Las únicas que pueden desplazarse son aquellas que comparten una arista con el espacio vacío



Reglas del juego



Finalización del juego

- El juego termina cuando se llega al estado final en el cual todos los números están ordenados de menor a mayor y el espacio vacío en la esquina inferior derecha.

1	2	3
4	5	6
7	8	



Estrategia de implementación



Algunas consideraciones de implementación

- Se verifica la win condition en la expansión a frontera, no en la exploración.
- La heurística local no aplica retroceso.
- El BPPV prioriza encontrar una solución rápida por sobre encontrar la óptima.



Algoritmos utilizados

- No informados:
 - BPP (Búsqueda por Profundidad)
 - BPPV (Búsqueda por Profundidad Variable)
 - BPA (Búsqueda por Anchura)
- Informados:
 - Búsqueda Heurística Local
 - Búsqueda Heurística Global
 - A*

Estrategia de implementación

Pseudocódigo vs. Implementación

Sean **A** un árbol, **F** un conjunto de nodos frontera de **A** y **Ex** el conjunto de nodos explorados.

Algoritmo de búsqueda

1. Crear **A**, **F** y **Ex** inicialmente vacíos.

2. Insertar el nodo raíz n_0 en **A** y **F**.

3. Mientras **F** no esté vacía

 Extraer el primer nodo **n** de **F**

 Si **n** no está en **Ex**, agregarlo

 Si **n** está etiquetado con un estado objetivo

 Devolver la solución, formada por los arcos entre

 la raíz n_0 y el nodo **n** en **A**

 Termina Algoritmo.

 Expandir el nodo **n**, guardando los sucesores de **n** en **A** y, en **F**, si es que no están en **Ex**.

 Reordenar **F** según el método de búsqueda.

4. Termina Algoritmo sin haber encontrado una solución.

```
name = "DFS"

def solve_internal(self):
    self.frontier.append(self.start_node)
    while self.frontier:
        node = self.frontier.pop()
        if node.state in self.visited:
            continue
        self.visited.add(node.state)
        self.analytics.expanded_count += 1
        for new_state in node.possible_moves():
            if new_state not in self.visited:
                new_node = SearchableNode(new_state, node, node.depth + 1, node.cost + 1)
                if new_state.is_solved():
                    return new_node
                self.frontier.append(new_node)
    return None
```



Searchers

Todos los buscadores se ramifican de una clase abstracta “Searcher” que determina el comportamiento de un buscador algorítmico. Hay otra clase abstracta que la hereda llamada “InformedSearcher” para los *algoritmos informados*. Las distintas clases inyectan su comportamiento particular en el método **solve_internal**, pero pueden usarse de forma indiferente y transparentemente entre sí.



SearchableNode

Los Searchers buscan en base a árboles que van construyendo. El nodo se llama SearchableNode. Se le debe incluir un estado (de interfaz Searchable, no importa su lógica interna) y acarrea información como su padre, su costo, profundidad, estimación al final, etc.



EightState

Es la representación del estado del tablero. Fundamentalmente una matriz $N \times N$. También tiene funcionalidades como calcular los movimientos posibles y determinar si llego a la win-condition (métodos necesarios para que sea Searchable). Se le puede pasar la posición de la celda en blanco para optimizar su inicialización.



Heurísticas implementadas

Heurísticas implementadas

Heurística “basica”

- Se basa en la cantidad de números que aún no están en la posición final
- Durante el juego, su valor va de 0 a 8.
- Es **admisible** ya que su valor nunca sobreestima el de la solución óptima, por que si el valor de la heurística basica es n , lo que significa que n números están fuera de su lugar objetivo, entonces al menos voy a tener que hacer n movimientos para alcanzar la solución. Por lo tanto, el valor de esta heurísticas siempre es menor o igual al de la óptima.

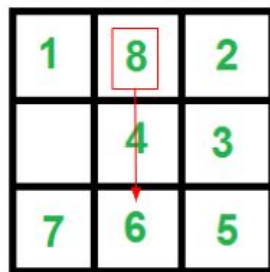
1	8	2
	4	3
7	6	5

En este caso, el valor de la heurística es 6

Heurísticas implementadas

Heurística en profundidad (“Deep”)

- Utiliza la suma de las distancias manhattan/taxi de cada número a su posición original para estimar la cantidad de movimientos que cada pieza debe realizar para llegar a su lugar objetivo. La misma se calcula sin restricciones, es decir, cada pieza puede moverse libremente en el tablero hasta llegar a su objetivo.
- Es **admisible** ya que para calcular esta heurística, se elimina la restricción que imponen los demás números del tablero. De esta forma, se obtiene un valor menor al óptimo por omitir los movimientos extras que se requieren hacer de contar con los números “bloqueando” el camino.



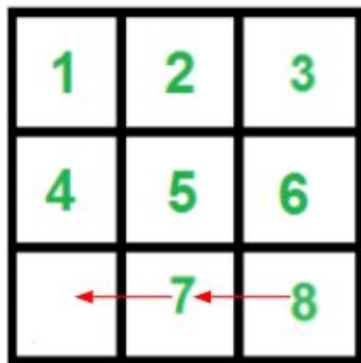
Ejemplo

La distancia manhattan para la pieza del 8 es dos. Para calcular el valor de la heurística se tendría en cuenta la suma de todas las distancias.

Heurísticas implementadas

Heurística no admisible (“Fat”)

- Utiliza la heurística básica al cuadrado
- Su rango de valores va de 0 a 64
- Es **no admisible** ya que, por ejemplo, la heurística óptima dará como resultado 2, pero esta heurística estimará el valor a 4, sobreestimando el costo óptimo.



1	2	3
4	5	6
	7	8



Visualización

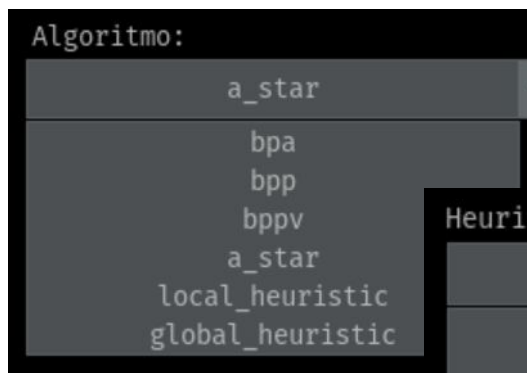
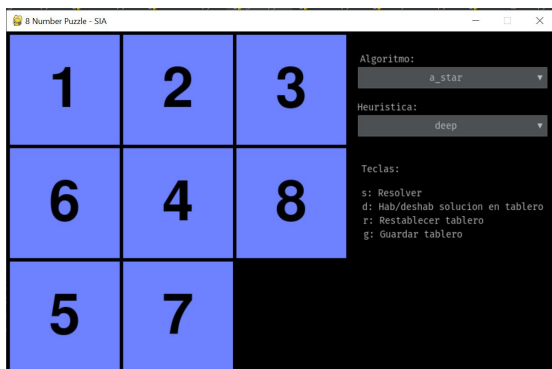


Consola

- Se puede hacer una corrida del programa con visualización en terminal.
- Es mejor para cálculos rápidos, puede correr en cualquier lado.
- Usado principalmente para armar stats.

Pygame

- Se desarrolló una GUI en pygame para probar los distintos algoritmos y heurísticas.
- Permite guardar estados, habilitar/deshabilitar repeticiones, y mezclar el tablero a mano.
- Muestra la solución paso por paso de forma muy visual



Teclas:

s: Resolver
d: Hab/deshab solución en tablero
r: Restablecer tablero
g: Guardar tablero

Heuristica:

deep

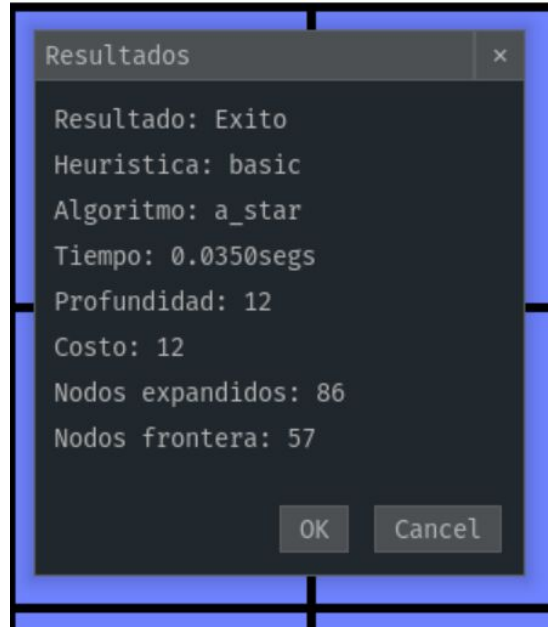
basic

deep

fat



Estadísticas





Demo

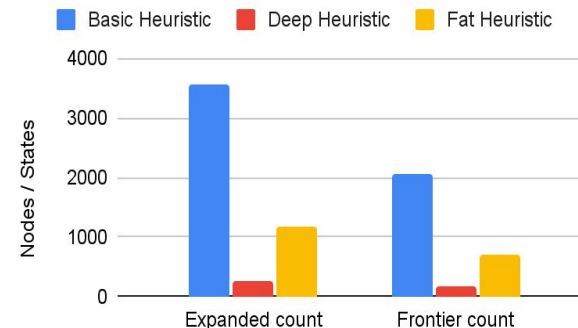
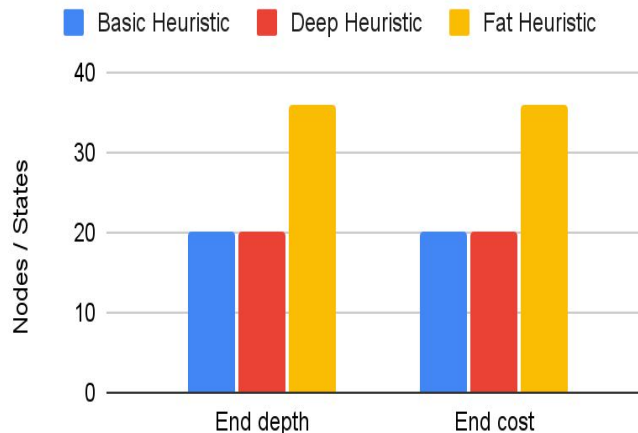
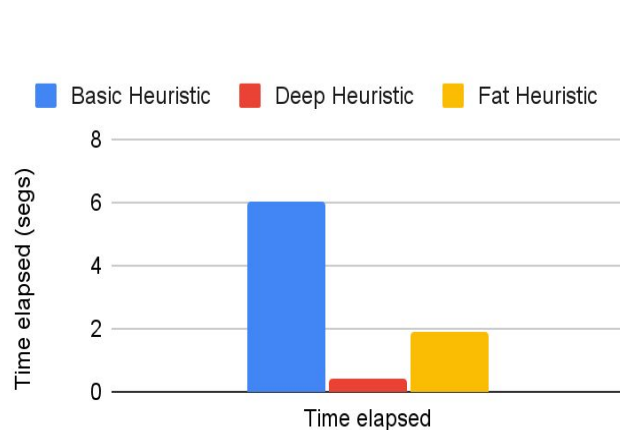


Análisis de los resultados

7	2	4
5		6
8	3	1

Diferencia de resultado entre heurísticas

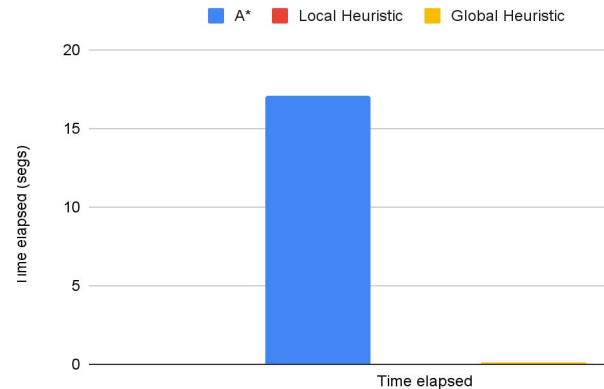
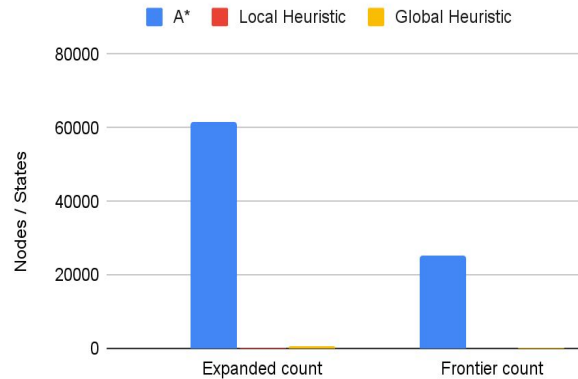
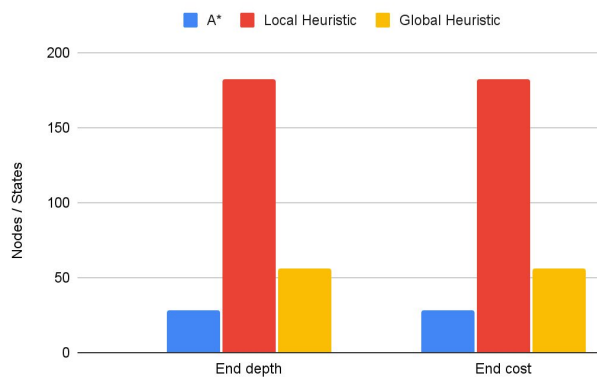
- En todos los casos, se puede observar un gran aumento en la performance del algoritmo al utilizar la heurística *deep* en vez de la *basic*. Esto se debe a que la *deep* se acerca más a la “realidad”.



Análisis de los resultados

8	7	6
1	4	3
5	2	

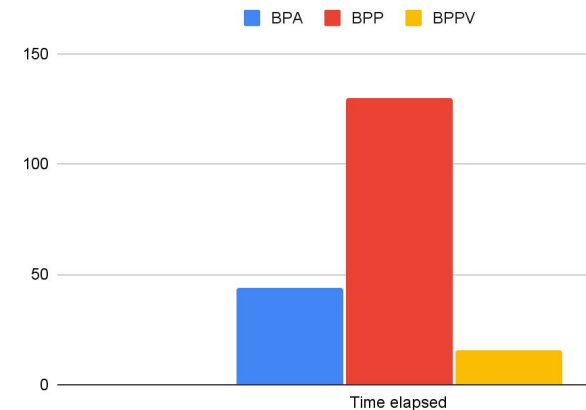
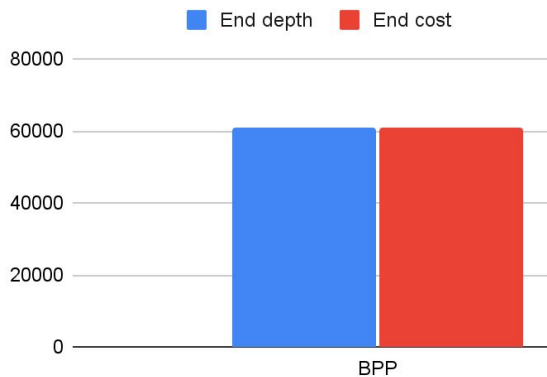
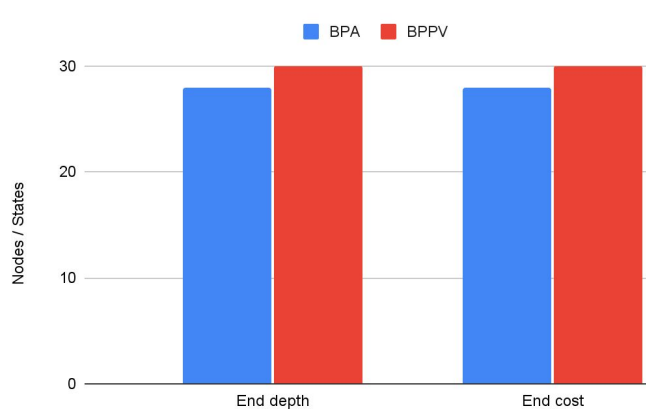
Diferencias entre informados



Análisis de los resultados

7	2	4
5		6
8	3	1

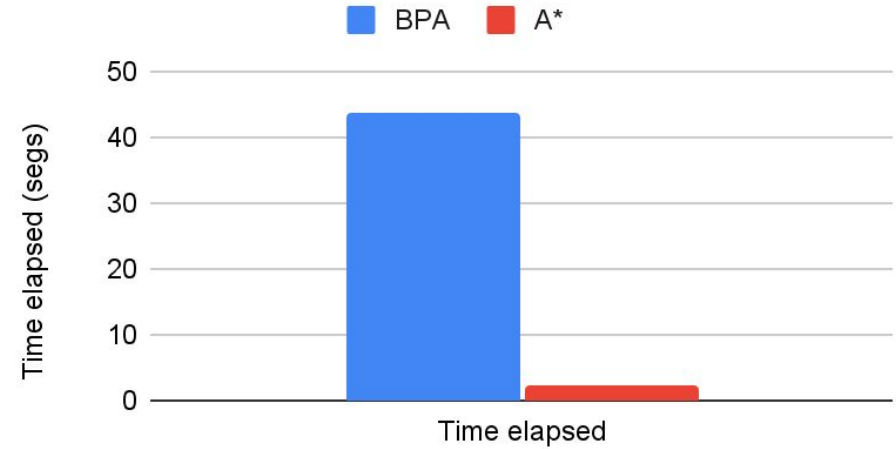
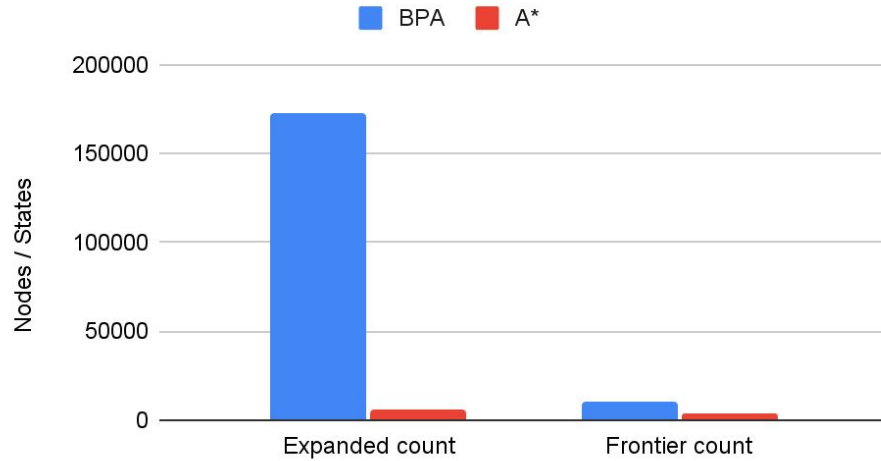
Diferencias entre no informados



Análisis de los resultados

8	7	6
1	4	3
5	2	

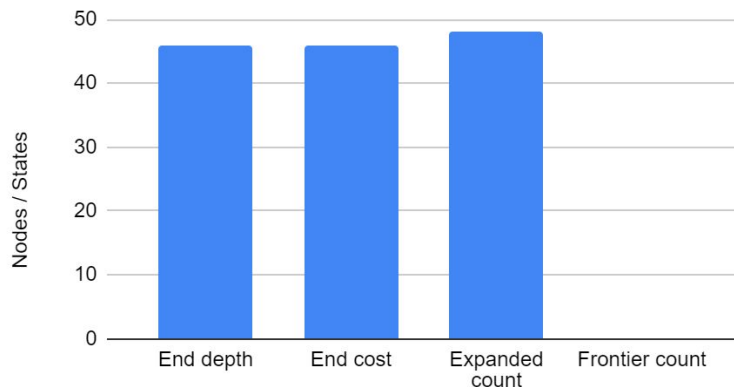
BPA VS A*



7	2	4
5		6
8	3	1

Caso en que heurística local no llega a la solución

- La heurística local no tiene retroceso, por lo que en algunas situaciones no logra hallar una solución a pesar de que exista
- Con la configuración inicial de la esquina derecha y utilizando la heurística *basic* como función, se evidencia lo mencionado.
- Pero, si se utiliza la heurística *deep* como función, se obtienen los siguiente resultados





Conclusión

Teniendo en cuenta lo estudiado en clase y puesto en práctica en este trabajo, creemos poder comprender cómo es que funcionan los distintos algoritmos usados y sus resultados. Analizando el tiempo de ejecución, los nodos expandidos y frontera, además del funcionamiento de cada algoritmo, la respuesta de “¿Qué algoritmo es el mejor?” no tiene una única respuesta...


Al momento de tomar esa decisión, hay que cuestionarse qué factor (o factores) se quiere priorizar y la información que se tiene de antemano sobre el problema. De contar con alguna, pensando en una heurística admisible se reduce la complejidad temporal y espacial de forma significativa.

No Informados	Velocidad	Uso de memoria	Solución Óptima
BPP	B	A	C
BPPV	B	B	B
BPA	C	C	A

A → C
mejor → peor

Informados	Velocidad	Uso de memoria	Solución Óptima
Local	A	A	C
Global	A	B	C
A*	A	B	A*

*si cumple con: acciones finitas, costo > 0 y heurística admisible



**Gracias por su
atencion.**

