En esta práctica se busca resolver el problema de las 8 reinas el cual consiste en que, a partir de un tablero de ajedrez con una reina ubicada en cada columna, estas se deben colocar de manera que cada reina no ataque a ninguna otra.

Estado inicial: una reina en cada columna.

La función de costo heurística **h** es el número de pares de reinas que se atacan la una a la otra.

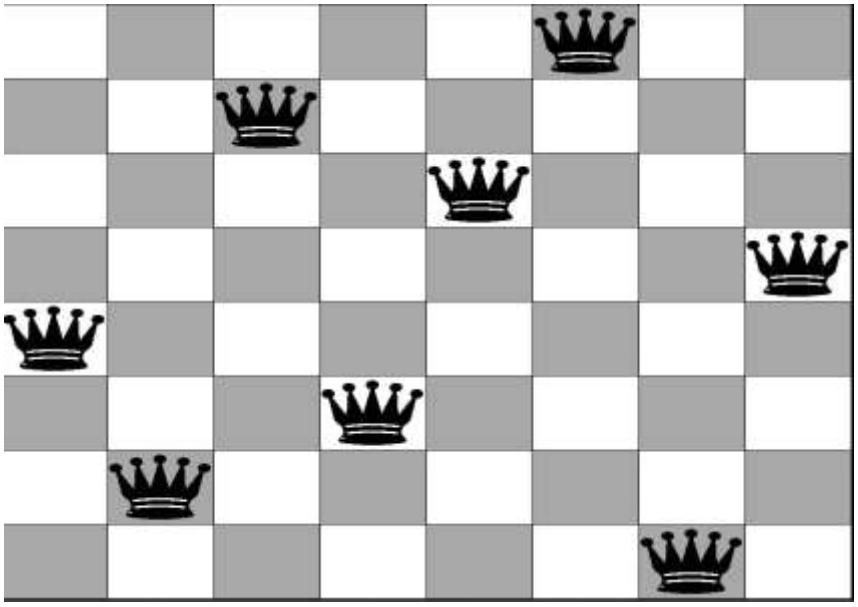


Imagen 1. Ejemplo del problema

Para su resolución se debe codificar algún algoritmo de búsqueda que permita encontrar la solución óptima al problema.

Algunos de los algoritmos de búsqueda que se pueden aplicar son:

Escalada simple o ascensión

Busca maximimizar (o minimizar) una función objetivo f(x), donde x es un vector de valores discretos y/o continuos.

Para entornos continuos, encuentra un valor máximo o mínimo variando gradualmente el estado inicial y evaluando la heurística.

Para entornos discretos, compara los nodos hijos.

Finaliza la búsqueda cuando encuentra un máximo o mínimo local. No garantiza la solución óptima.

El primero mejor

Algoritmo heurístico que presenta una medida, similitud o diferencia **h** entre el estado actual y el estado objetivo.

Se analizan todas las posibilidades de los sucesores con similares características para elegir la mejor y expandirla.

Código

```
import numpy as np
import matplotlib.pyplot as plt
import random


#Se crea el tablero a partir de una semilla
def crear_tablero(semilla):
    plt.figure(figsize=(5,5))
    np.random.seed(semilla)
    tablero = np.zeros((8,8))
    reinas = np.zeros(8)

    #Se llenan los valores del tablero
    for i in range(0,8):
        fila=np.random.randint(0,8)
        tablero[fila,i] = 1
        reinas[i] = fila
    return tablero

"""Para calcular la heuristica del tablero,se aplican 2 funciones,
mov_horizontales y mov_diagonales. En el caso de mov_horizontales,
recibe como
parámetro el tablero y devuelve le heuristica de cada fila, el
procedimiento que realiza es analizar los valores de cada fila a
partir de diferentes
condicionales que le permiten ir incrementando dicho valor"""
def mov_horizontales(tablero):
  for i in range(8):
    contador=0 #En esta variable se va a guardar la heuristica total
de todas las filas
    for j in range(8): #Mediante este ciclo se van a ir analizando las
8 filas del tablero
        reinas_conteo = np.count_nonzero(tablero[j] == 1) #En esta
variable se guarda en forma de lista los valores que contiene cada
fila del tablero para poder analizarlos
        if(reinas_conteo == 2):
            contador += 1
        elif(reinas_conteo == 3):
            contador += 3
        elif (reinas_conteo == 4):
            contador += 6
        elif (reinas_conteo == 5):
            contador += 10
        elif (reinas_conteo == 6):
            contador += 15
        elif (reinas_conteo == 7):
            contador += 21
        elif (reinas_conteo == 8):
            contador += 28

  return contador

"""Para el caso de la función mov_diagonales, se realiza el mismo
procedimiento que en mov_horizontales, pero en este caso devuelve la
heuristica de cada diagonal del tablero"""
def mov_diagonales(tablero):
    contador = 0
    for i in range(-7, 7): #Se hace la iteración para las diagonales
que hay en un extremo del tablero
```

```
        tablero_filas_izq = np.diag(tablero, i)
        reinasConteo = np.count_nonzero(tablero_filas_izq == 1)
        if(reinasConteo == 2):
            contador += 1
        elif(reinasConteo == 3):
            contador += 3
        elif (reinasConteo == 4):
            contador += 6
        elif (reinasConteo == 5):
            contador +=10
        elif (reinasConteo == 6):
            contador += 15
        elif (reinasConteo == 7):
            contador += 21
        elif (reinasConteo == 8):
            contador += 28
    return contador


"""La función heuristica recibe como parámetros los valores de las
heuristicas para las filas, diagonales izquierdas y diagonales
derechas, con ello
hace la suma para obtener la heuristica total del tablero"""
def heuristica(c1, c2, c3):
    return c1+c2+c3

"""La función heuristicas_col, recibe como parámetro el tablero y a
partir de él va evaluando cada columna para encontrar la heuristica
menor de cada una, al
final devuelve las heuristicas menores de cada columna y la fila en la
que se encontraron"""
def heuristicas_col(tablero):
    tablero_aux = np.copy(tablero) #Se crea una copia del tablero para
no alterar al original
    lista_h = list() #Se crea la lista para guardar las heuristicas de
las 8 columnas
    filas = list() #Se crea la lista para guardar las filas
correspondientes a cada heuristicas
    for i in range(8): #Mediante este ciclo se va iterando por cada
columna
        for j in range(8): #Mediante este ciclo se va iterando por
cada fila
            heuristica_pasada =
heuristica(mov_diagonales(np.flip(tablero_aux, axis=1)),
mov_horizontales(tablero_aux), mov_diagonales(tablero_aux)) #Se guarda
la heuristica original
            tablero_aux[:,i] = np.roll(tablero_aux[:,i], 1)  #Se va
moviendo verticalmente la reina de la columna correspondiente
            heuristica_actual =
heuristica(mov_diagonales(np.flip(tablero_aux, axis=1)),
mov_horizontales(tablero_aux), mov_diagonales(tablero_aux)) #Se
obtiene la nueva heuristica a partir del movimiento realizado
anteriormente
            if(heuristica_actual <= heuristica_pasada): #Se comprueba
cual heuristica es la menor entre la nueva y la original
                heuristica_menor = heuristica_actual #Se guarda la
heuristica menor
                fila = j #Se guarda la fila en donde se encontró la
heuristica menor
                if(heuristica_menor == 0): #Si la heuristica menor es
cero, entonces se termina el ciclo ya que no puede haber resultados
```

```
menores
                break
        filas.append(fila) #Se agrega la fila correspondiente a la
heuristica menor a la lista de filas
        lista_h.append(heuristica_menor) #Se agrega la heuristica
menor obtenida a la lista de heuristicas menores

    return lista_h, filas #Se devuelve la lista de heuristicas menores
con la lista de sus filas correspondientes

"""Mediante la función actualizar_tablero que recibe como parámetros
la lista de heuristicas menores, las filas correspondientes y el
tablero se
mueve la reina que está ubicada en la columna que resultó con menor
heuristica a partir de la función heuristicas_col y con ello se
devuelve el
tablero actualizado"""
def actualizar_tablero(heuristicas, movimientos, tablero):
    heuristica_min = min(heuristicas) #Se obtiene la menor heuristica
entre la lista de heuristicas
    columna = heuristicas.index(heuristica_min) #Se obtiene la columna
en donde está la mínima heuristica
    movs = movimientos[columna] #Se obtiene la fila o movimientos que
debe realizar el tablero en la columna anterior para poder obtener la
mínima heuristica
    tablero_aux = np.copy(tablero) #Se realiza una copia del tablero
para no afectar al original
    tablero_aux[:,columna] = np.roll(tablero_aux[:,columna], movs+1)
#Se realiza el movimiento de la reina para obtener la nueva heuristica
menor
    return tablero_aux #Se regresa el tablero actualizado

"""Mediante la función busqueda que recibe como parámetro el tablero,
se realiza el procedimiento para encontrar la menor heuristica del
mismo, para ello
implementa las funciones heuristicas_col y actualizar_tablero, hasta
llegar al valor más bajo que pueda alcanzar"""
def busqueda(tablero):
    heuristica_original = heuristica(mov_diagonales(np.flip(tablero,
axis=1)), mov_horizontales(tablero),mov_diagonales(tablero))
    print(f'Heuristica original: {heuristica_original}')
    tablero_movs = np.copy(tablero) #Se crea una copia del tablero
para no alterar al original
    heuristica_pasada = 1 #Se define la variable para almacenar a la
heuristica pasada
    heuristica_actual= 0 #Se define la variable para almacenar a la
heuristica actual
    moves = 0 #Se define la variable para almacenar los movimientos
que le toma al algoritmo encontrar la menor heuristica
    while(1):
        heuristica_pasada =
heuristica(mov_diagonales(np.flip(tablero_movs, axis=1)),
mov_horizontales(tablero_movs), mov_diagonales(tablero_movs)) #Se
calcula la heuristica que tiene el tablero antes de realizar los
movimientos
        heus, movs = heuristicas_col(tablero_movs) #Se invoca a la
función heuristicas_col y se almacenan los valores que retorna en unas
variables
        tablero_movs = actualizar_tablero(heus,movs,tablero_movs) #Se
invoca a la función tablero_movs y se le pasan las variables
anteriores como parámetros
```

```
        heuristica_actual =
heuristica(mov_diagonales(np.flip(tablero_movs, axis=1)),
mov_horizontales(tablero_movs), mov_diagonales(tablero_movs)) #Se
calcula la heuristica que tiene el tablero después de los movimientos
        if(heuristica_actual < heuristica_pasada): #Si se cumple que
la heuristica actual es menor que la pasada, entonces se sigue
ejecutando el ciclo
            moves += 1 #Se aumenta en una unidad los movimientos del
tablero
            print('---------------------------------')
            print(f'Movimiento {moves}')
            plt.pcolor(tablero_movs, edgecolors='k', linewidths=3)
            plt.savefig('tablero.png', dpi=300)
            plt.show()
            print(f'Heuristica actual: {heuristica_actual}')
        else: break #En el caso de que la heuristica pasada es mayor,
eso significa que el algoritmo ya encontró la menor heuristica que
pudo, por lo que el ciclo se termina
    print('---------------------------------')
    print(f'Heuristica final: {heuristica_actual}') #Se muestra la
heuristica final encontrada
    print(f'Movimientos realizados: {moves}') #Se muestra el total de
movimientos que le tomó al algoritmo para llegar a la heuristica final

if __name__ == "__main__":
    num_semilla = int(input("Ingrese el valor de la semilla: "))
    tablero_copia = crear_tablero(num_semilla) #Se crea el tablero a
partir de la semilla ingresada
    plt.pcolor(tablero_copia, edgecolors='k', linewidths=3)
    plt.savefig('tablero.png', dpi=300)
    plt.show()
    busqueda(tablero_copia) #Se invoca a la función busqueda para
hallar la solución al problema con base al tablero creado
anteriormente
```

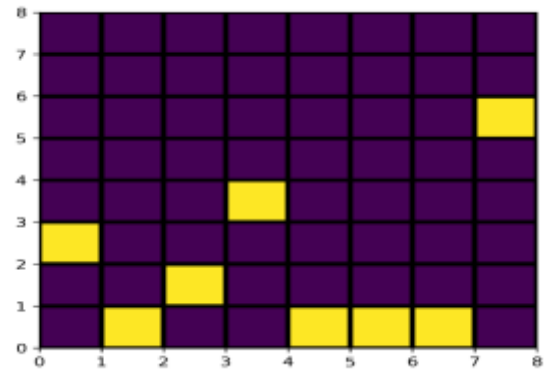
A continuación, se presentan los resultados obtenidos para 10 diferentes semillas {3, 11, 18, 31, 39, 50, 65, 72, 87, 107} con el objetivo de comprobar el comportamiento del algoritmo para distintos estados del tablero.

Semilla 3

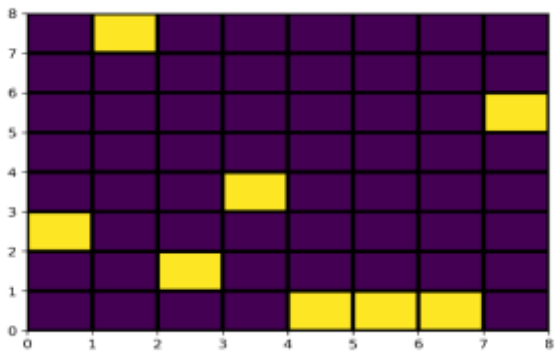
```
Heuristica original: 8
--------------------------------
Movimiento 1
Heuristica actual: 4
--------------------------------
Movimiento 2
Heuristica actual: 2
--------------------------------
Movimiento 3
Heuristica actual: 1
--------------------------------
Heuristica final: 1
Movimientos realizados: 3

Process finished with exit code 0
```

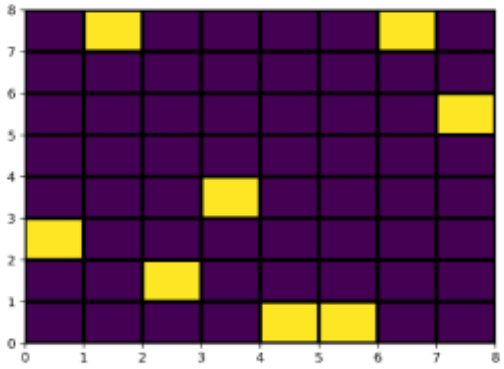
Resultados de la semilla 3



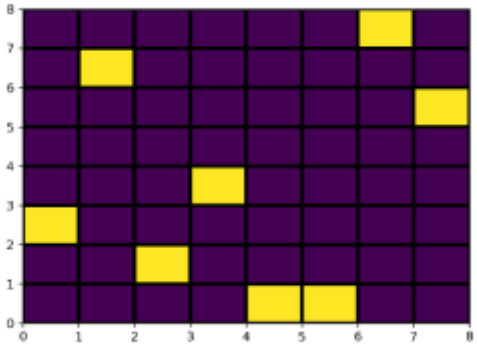
Estado inicial. Heurística = 8



Primer movimiento. Heurística = 4



Segundo movimiento. Heurística = 2



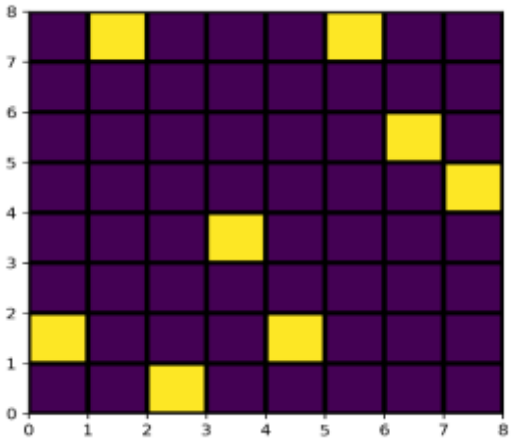
Tercer movimiento. Heurística = 1

Semilla 11

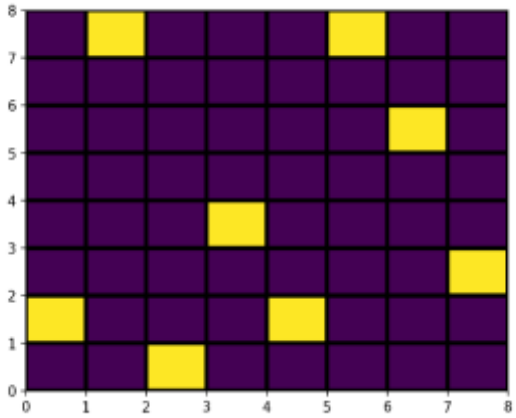
```
Heuristica original: 4
--------------------------------
Movimiento 1
Heuristica actual: 2
--------------------------------
Movimiento 2
Heuristica actual: 1
--------------------------------
Movimiento 3
Heuristica actual: 0
--------------------------------
Heuristica final: 0
Movimientos realizados: 3

Process finished with exit code 0
```

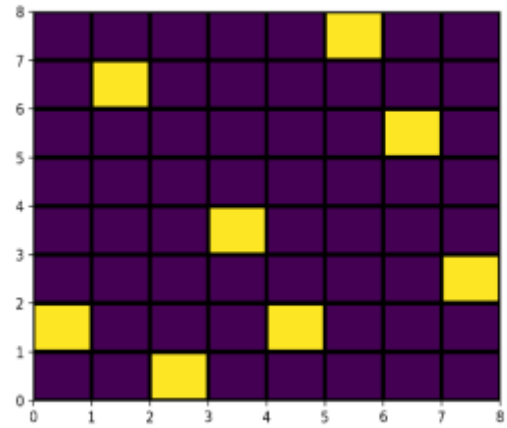
Resultados de la semilla 11



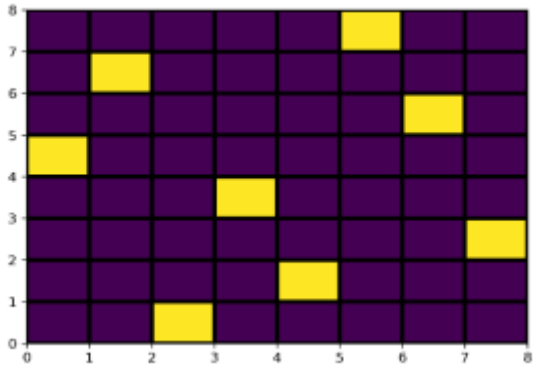
Estado inicial. Heurística = 4



Primer movimiento. Heurística = 2



Segundo movimiento. Heurística = 1

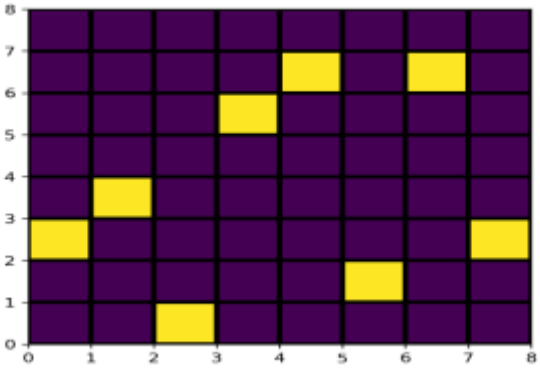


Tercer movimiento. Heurística final = 0

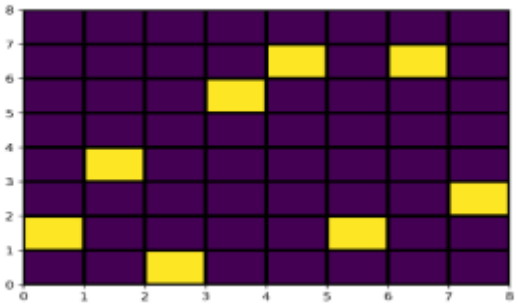
Semilla 18

```
Heuristica original: 9
---------------------------------
Movimiento 1
Heuristica actual: 5
---------------------------------
Movimiento 2
Heuristica actual: 4
---------------------------------
Movimiento 3
Heuristica actual: 3
---------------------------------
Movimiento 4
Heuristica actual: 2
---------------------------------
Heuristica final: 2
Movimientos realizados: 4
```

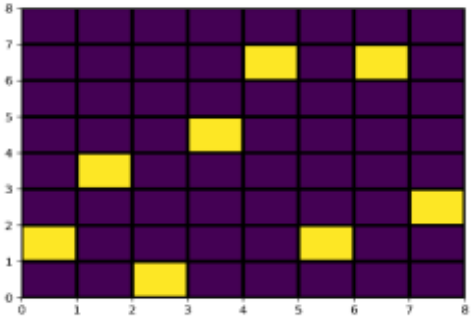
Resultados de la semilla 18



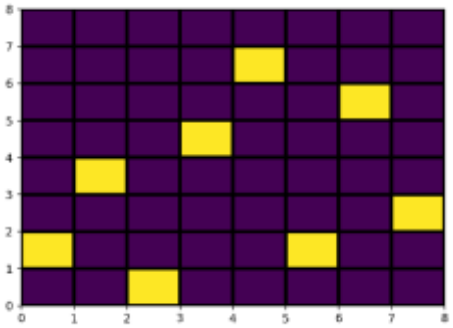
Estado inicial. Heurística = 9



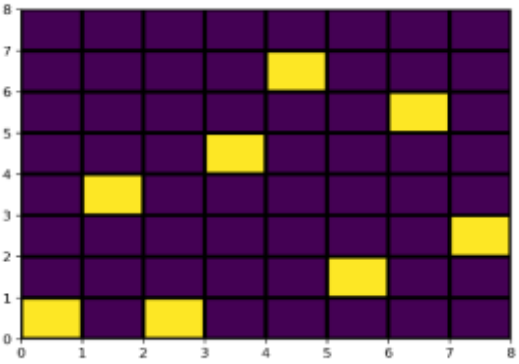
Primer movimiento. Heurística = 5



Segundo movimiento. Heurística = 4



Tercer movimiento. Heurística = 3



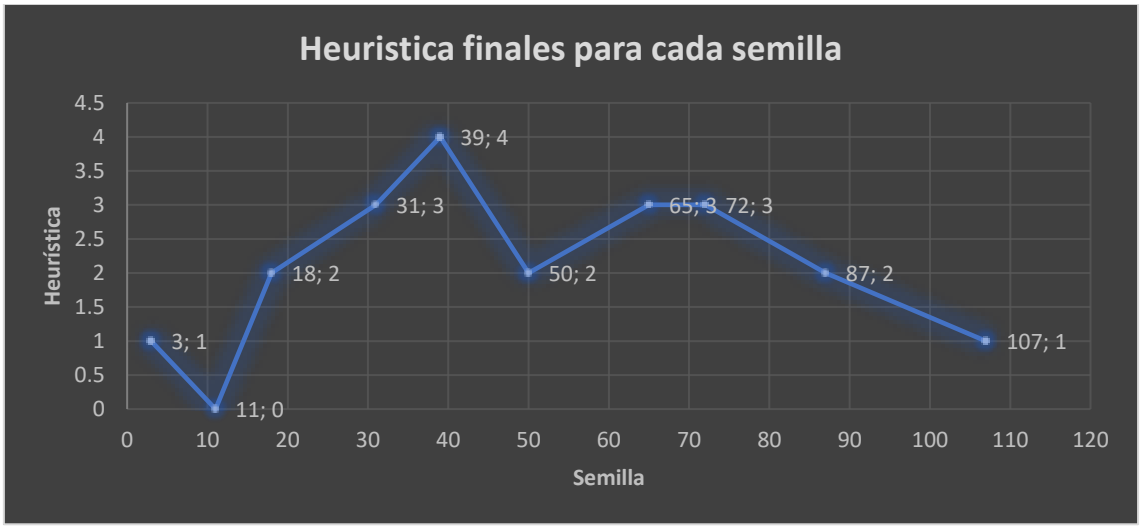
Cuarto movimiento. Heurística final = 2

Resultados

Al final, se obtuvieron los siguientes resultados para las semillas contempladas:

Semilla	Heurística original	Heurística final	Movimientos realizados
3	8	1	3
11	4	0	3
18	9	2	4
31	8	3	2
39	8	4	3
50	5	2	2
65	10	3	3
72	6	3	1
87	10	2	4
107	7	1	2

Tabla 1. Resultados de las heurísticas de cada semilla evaluada



Gráfica 1. Resultados de las heurísticas de cada semilla evaluada.

Se observa que en la mayoría de los casos el algoritmo llega a encontrar valores mínimos de heurística entre 1 y 3, aunque también hay casos en los que puede llegar a 0.

Conclusiones

Mediante esta práctica pudimos resolver el problema de las 8 reinas, aplicando un algoritmo basado en la búsqueda de “el primero mejor”, para ello fueron muy útiles los ejercicios revisados en clase sobre este tema, ya que nos dio la idea para idear el algoritmo que contemplara todas las posibilidades que puede haber al mover de posición cada una de las reinas. Aunque se debe destacar que el tipo de búsqueda que usamos no es la más eficiente, ya que su complejidad en tiempo y espacio es bastante alta, pues requiere del uso de mucha memoria y el tiempo que tarda en realizar las búsquedas es un poco alto en comparación a otros tipos de búsqueda como la escalada simple, además de que no se asegura que la solución encontrada sea la óptima, ya que siempre busca la primera que considera como la mejor, aunque en realidad no lo sea.