INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

INTELIGENCIA ARTIFICIAL


# PRÁCTICA 5

# BÚSQUEDA INFORMADA P.II

NOMBRE DEL ALUMNO: GARCÍA QUIROZ GUSTAVO IVAN

NOMBRE DEL PROFESOR: GARCÍA FLORIANO ANDRES

FECHA DE ENTREGA DEL REPORTE: 04/10/2024

# Índice

# Marco teórico

## Función de Himmelblau

La función de Himmelblau es una función multimodal, definida sobre  y usada para comprobar el rendimiento de los algoritmos de optimización.

La función se define de la siguiente manera:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2.$$

*Figura 1 Función de Himmelblau.*

Tiene un máximo local en  y  donde , y cuatro mínimos locales idénticos (también son mínimos globales):

*Figura 2 Mínimos globales.*

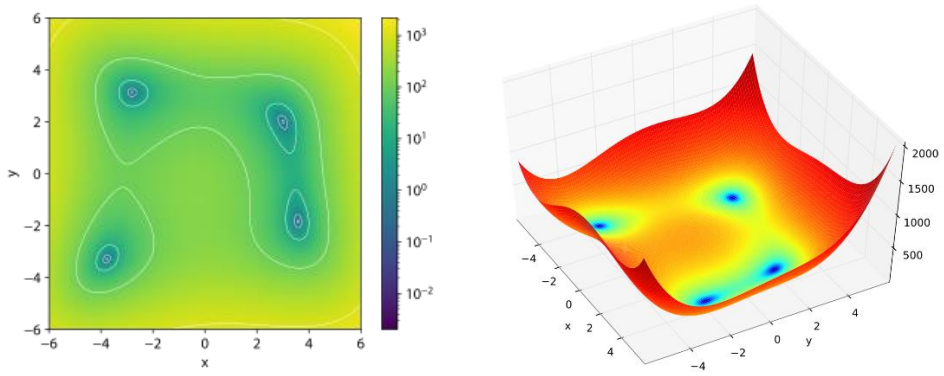


*Figura 3 Gráfica de la función de Himmelblau.*

# Introducción

La práctica de búsqueda informada se enfoca en la aplicación del algoritmo A* para resolver problemas de optimización. En este caso específico, el objetivo es encontrar los valores (x,y) que minimizan la función de Himmelblau, una función no convexa utilizada comúnmente para probar algoritmos de optimización.

El problema planteado requiere la creación de un programa que busque los valores mínimos de la función de Himmelblau dentro de un rango definido. La restricción establecida limita el espacio de búsqueda a valores de x e y entre -5 y 5, lo que acota el área de exploración y permite una implementación más eficiente del algoritmo.

Para llevar a cabo esta práctica, se utilizaron herramientas de software y hardware específicas. En cuanto al hardware, se empleó una computadora para ejecutar el código y realizar los cálculos necesarios. El software utilizado incluyó Visual Studio Code como entorno de desarrollo integrado (IDE) y Python 3 como lenguaje de programación.

Esta práctica busca la aplicación del algoritmo A* y el desarrollo de la formulación de problemas de optimización, la implementación de heurísticas adecuadas y la interpretación de resultados en un contexto de búsqueda informada.

# Material y equipo.

Para esta práctica se usaron las siguientes herramientas de software y hardware necesarias para realizar la práctica.

### Hardware

- Computadora.

### Software

- Visual Studio Code.
- Phyton 3.

# Desarrollo de la práctica.

En esta práctica estudiamos la implementación de la búsqueda informada utilizando el algoritmo A* para resolver un problema de optimización. El objetivo principal fue encontrar los valores (x,y) que minimizan la función de Himmelblau en un espacio de búsqueda definido. El lenguaje de programación elegido fue Python porque puede simplificar el código y utilizar funciones de manera más rápida.

## Actividades desarrolladas

Se desarrolló un programa en Python para implementar el algoritmo A* adaptado al problema de la función de Himmelblau. Los pasos principales fueron:

### Definición de la función de Himmelblau.

```python
# Función de Himmelblau
def f(x, y):
    return (x**2 + y - 11)**2 + (x + y**2 - 7)**2
```

*Figura 4 función de Himmelblau.*

### Diseño de la heurística

Se utilizó como heurística la diferencia absoluta entre el valor actual de la función y el valor objetivo (en este caso, 0 para buscar el mínimo global).

```python
# Heurística (distancia al valor objetivo)
def h(x, y):
    return abs(f(x, y) - objetivo_z)
```

*Figura 5 Heurística.*

### Implementación del algoritmo A*

La implementación del algoritmo A* para resolver el problema de la función de Himmelblau se realizó en Python. El algoritmo se estructuró en una función llamada a_estrella_himmelblau que toma como parámetros las coordenadas iniciales, el valor objetivo, el tamaño del paso y el número máximo de iteraciones.

```python
def a_estrella_himmelblau(inicio_x, inicio_y, objetivo_z, paso, max_iteraciones):
    # Nodo inicial
    inicio = (inicio_x, inicio_y)
    f_inicio = f(inicio_x, inicio_y)

    # Lista abierta y cerrada
    abierta = [(f_inicio, 0, h(inicio_x, inicio_y), inicio)]
    cerrada = set()

    for _ in range(max_iteraciones):
        if not abierta:
            return None  # No se encontró solución

        # Obtener el nodo con menor f = g + h
        actual_f, actual_g, actual_h, (actual_x, actual_y) = min(abierta)
        abierta.remove((actual_f, actual_g, actual_h, (actual_x, actual_y)))

        # Verificar si hemos alcanzado el objetivo
        if abs(actual_f - objetivo_z) < 1e-6:
            return (actual_x, actual_y)

        cerrada.add((actual_x, actual_y))

        # Generar vecinos
        for dx, dy in [(paso, 0), (-paso, 0), (0, paso), (0, -paso)]:
            vecino_x, vecino_y = actual_x + dx, actual_y + dy
            if (vecino_x, vecino_y) in cerrada:
                continue

            vecino_g = actual_g + paso
            vecino_h = h(vecino_x, vecino_y)
            vecino_f = f(vecino_x, vecino_y)

            for i, (_, g, _, (x, y)) in enumerate(abierta):
                if (x, y) == (vecino_x, vecino_y):
                    if g <= vecino_g:
                        break
                    del abierta[i]
                    continue
            else:
                abierta.append((vecino_f, vecino_g, vecino_h, (vecino_x, vecino_y)))

    return None  # No se encontró solución en las iteraciones máximas
```

*Figura 6 Función llamada a_estrella_himmelblau.*

El algoritmo inicializa una lista abierta con el nodo inicial y un conjunto cerrado vacío. La lista abierta contiene tuplas con la información de cada nodo: el valor de la función f, el costo acumulado g, el valor heurístico h y las coordenadas (x, y).

El bucle principal del algoritmo itera hasta alcanzar el número máximo de iteraciones o encontrar una solución. En cada iteración, se selecciona el nodo con el menor valor de f de la lista abierta. Si este nodo alcanza el objetivo (dentro de una tolerancia), se devuelve como solución.

Si no se alcanza el objetivo, el nodo actual se mueve a la lista cerrada y se generan sus vecinos. Los vecinos se crean aplicando el tamaño del paso en las cuatro direcciones cardinales. Para cada vecino, se calcula su costo g, su valor heurístico h y el valor de la función f.

Los vecinos se añaden a la lista abierta si no están en la lista cerrada y si no existe un camino mejor hacia ellos en la lista abierta. Si ya existe un nodo para las mismas coordenadas en la lista abierta con un costo g menor, se mantiene el nodo existente.

Si el algoritmo no encuentra una solución dentro del número máximo de iteraciones, devuelve None. Fuera de la función principal, se llama al algoritmo con los parámetros iniciales y se imprime el resultado si se encuentra una solución.

## Resultados obtenidos

```
Mínimo encontrado en x = 3.579999999999967, y = -1.8699999999999999
Valor de la función: 0.008786570000019912
Mínimo encontrado en x = 3.0099999999999794, y = 1.9800000000000009
Valor de la función: 0.006452169999992478
Mínimo encontrado en x = -2.7999999999999843, y = 3.1199999999999775
Valor de la función: 0.005903360000025415
Mínimo encontrado en x = -3.790000000000026, y = -3.2900000000000365
Valor de la función: 0.0066536200000382045
```

*Figura 7 Resultados obtenidos.*

## Conclusiones.

La implementación del algoritmo A* para minimizar la función de Himmelblau demostró la eficacia de las técnicas de búsqueda informada en problemas de optimización continua. Se aprendió la importancia del diseño de una heurística adecuada y el impacto de los parámetros como el tamaño del paso y el número máximo de iteraciones en el rendimiento del algoritmo. La práctica reforzó conceptos clave como la gestión de listas abiertas y cerradas, la generación de nodos vecinos en espacios continuos y la convergencia hacia mínimos locales y globales. Para mejorar la comprensión y aplicación del algoritmo A* en este contexto, se debió repasar los fundamentos de la búsqueda heurística, las propiedades de las funciones no convexas y las estrategias para evitar la convergencia prematura en mínimos locales.

## Referencias.

[1] 4_Informed_Search at main · GUSTAVOIVANGQ/AI. .

## Código

https://github.com/GUSTAVOIVANGQ/AI/tree/main/4_Informed_Search