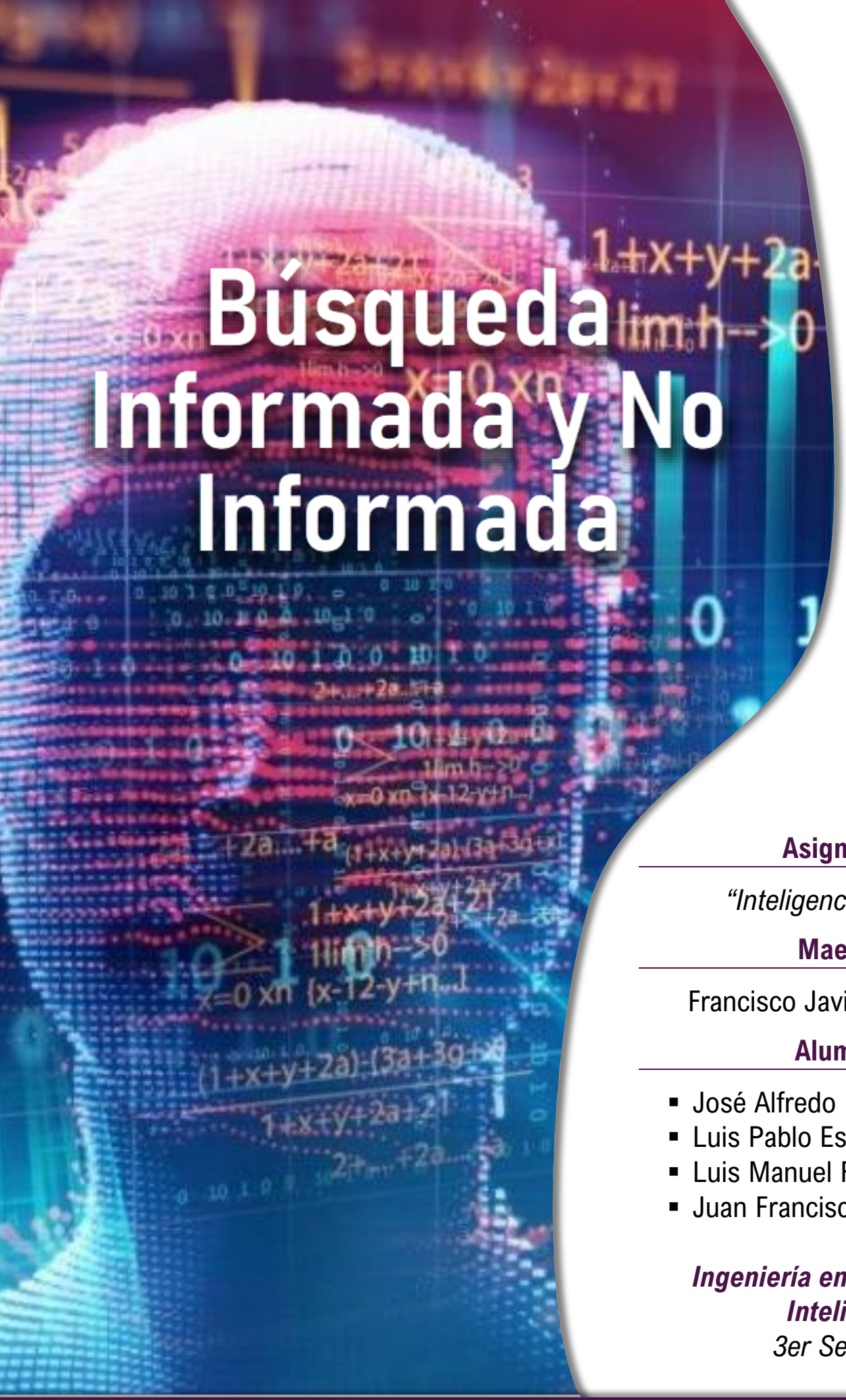

Búsqueda Informada y No Informada

Centro de Ciencias Básicas

Asignatura:

"Inteligencia Artificial"

Maestro:

Francisco Javier Luna Rosas

Alumnos:

- José Alfredo Díaz Robledo
- Luis Pablo Esparza Terrones
- Luis Manuel Flores Jiménez
- Juan Francisco Gallo Ramírez

Ingeniería en Computación Inteligente

3er Semestre

Introducción

La búsqueda desempeña un papel fundamental en la resolución de problemas en el campo de la inteligencia artificial. Los algoritmos de búsqueda, tanto informados como no informados, son herramientas esenciales para encontrar soluciones en una amplia variedad de dominios, desde la planificación de rutas en mapas hasta la toma de decisiones en juegos y la optimización de recursos en la industria.

Esta actividad se enfoca en explorar y analizar dos categorías principales de algoritmos de búsqueda: Búsqueda No Informada y Búsqueda Informada. Los algoritmos de Búsqueda No Informada, como la Búsqueda en Anchura y la Búsqueda en Profundidad, se basan en un enfoque más simple de exploración, mientras que los algoritmos de Búsqueda Informada, como la Búsqueda Avara (Best-first), Hill Climbing, A*, Recocido Simulado y Búsqueda Tabú, incorporan información adicional para guiar la búsqueda hacia soluciones óptimas de manera más eficiente.

En este documento, se proporcionará una implementación detallada de cada uno de estos algoritmos, seguida de un análisis exhaustivo de sus características, ventajas y desventajas. Además, se llevará a cabo una evaluación de su rendimiento en diferentes escenarios y se presentarán conclusiones que resuman los hallazgos clave.

La presente actividad se desarrollará tanto de forma individual como colaborativa en equipos, y se espera que los participantes presenten una comprensión sólida de los algoritmos de búsqueda y su capacidad para aplicarlos en diversos contextos. También se hace hincapié en la importancia de entregar no solo el documento PDF, sino también los archivos de código fuente para permitir la revisión y replicación de los resultados.

Análisis

Breadth First *(Búsqueda Informada)*

1. **Importación de bibliotecas:**

Se importan las bibliotecas networkx y matplotlib.pyplot. networkx se utiliza para trabajar con grafos, y matplotlib.pyplot para visualizar el grafo.

2. **Creación del grafo:**

Se crea un grafo vacío utilizando la función nx.Graph() de NetworkX. Este grafo se utilizará para representar ciudades en los Estados Unidos.

3. **Adición de nodos:**

Se agregan nodos al grafo, representando diferentes ciudades de los Estados Unidos. Cada ciudad se agrega utilizando G.add_node("NombreDeLaCiudad").

4. **Adición de aristas:**

Se agregan aristas entre los nodos para representar las conexiones entre las ciudades. Cada arista se agrega con G.add_edge("CiudadOrigen", "CiudadDestino", weight=PesoDeLaArista). Los pesos de las aristas representan la distancia o costo entre las ciudades.

5. **Visualización del grafo:**

El grafo se visualiza utilizando la función nx.draw(), lo que genera un gráfico que muestra las ciudades y las conexiones entre ellas.

6. **Función de búsqueda en anchura:**

Se define la función bfs(grafo, nodo_inicial) para realizar la búsqueda en anchura en el grafo. El algoritmo utiliza una cola para llevar un registro de los nodos que se deben visitar. Comienza desde el nodo inicial y explora todos los nodos

vecinos antes de avanzar al siguiente nivel. La función devuelve una lista de nodos visitados en orden de exploración.

7. **Búsqueda en anchura:**

Se llama a la función bfs para realizar una búsqueda en anchura desde un nodo inicial. En este caso, se inicia la búsqueda desde "Rochester", pero se puede elegir cualquier otra ciudad como punto de partida.

8. **Impresión del resultado:**

Se imprime el recorrido resultante de la búsqueda en anchura.

Depth First *(Búsqueda Informada)*

1. **Importación de bibliotecas:**

Se importan las bibliotecas networkx y matplotlib.pyplot. networkx se utiliza para trabajar con grafos, y matplotlib.pyplot para visualizar el grafo.

2. **Creación del grafo:**

Se crea un grafo vacío utilizando la función nx.Graph() de NetworkX. Al igual que en el ejemplo anterior, este grafo se utilizará para representar ciudades en los Estados Unidos.

3. **Adición de nodos:**

Se agregan nodos al grafo, representando diferentes ciudades de los Estados Unidos. Cada ciudad se agrega utilizando G.add_node("NombreDeLaCiudad").

4. **Adición de aristas:**

Se agregan aristas entre los nodos para representar las conexiones entre las ciudades. Cada arista se agrega con G.add_edge("CiudadOrigen",

"CiudadDestino", weight=PesoDeLaArista). Los pesos de las aristas representan la distancia o costo entre las ciudades.

5. **Visualización del grafo:**

El grafo se visualiza utilizando la función nx.draw(), lo que genera un gráfico que muestra las ciudades y las conexiones entre ellas, al igual que en el ejemplo anterior.

6. **Función de búsqueda en profundidad:**

Se define la función dfs(grafo, nodo_inicial) para realizar la búsqueda en profundidad en el grafo. El algoritmo utiliza una pila para llevar un registro de los nodos que se deben visitar. Comienza desde el nodo inicial y explora lo más profundo posible antes de retroceder. La función devuelve una lista de nodos visitados en orden de exploración.

7. **Búsqueda en profundidad:**

Se llama a la función dfs para realizar una búsqueda en profundidad desde un nodo inicial. En este caso, se inicia la búsqueda desde "Rochester", pero puedes elegir cualquier otra ciudad como punto de partida.

8. **Impresión del resultado:**

Se imprime el recorrido resultante de la búsqueda en profundidad.

Hill Climbing *(Búsqueda No Informada)*

1. **Importación de bibliotecas:**

Se importan las bibliotecas networkx y matplotlib.pyplot para trabajar con grafos y visualizarlos, al igual que en los ejemplos anteriores.

2. **Creación del grafo:**

Se crea un grafo vacío utilizando la función nx.Graph() de NetworkX, que se utilizará para representar las ciudades y las conexiones entre ellas.

3. **Adición de nodos y aristas:**

Se agregan nodos al grafo para representar diferentes ciudades de los Estados Unidos y se añaden aristas para representar las conexiones entre ellas, al igual que en los ejemplos anteriores.

4. **Visualización del grafo:**

El grafo se visualiza con la función nx.draw().

5. **Función calculate_cost:**

Se define la función calculate_cost(route, graph) para calcular el costo total de una ruta dada en el grafo. Esta función suma los pesos de las aristas en la ruta.

6. **Algoritmo Hill Climbing:**

Se define la función hill_climbing(graph, initial_node, destination_node) que implementa el algoritmo Hill Climbing. El algoritmo comienza en el nodo inicial y busca el nodo de destino de manera iterativa. En cada iteración, considera los vecinos del nodo actual y selecciona el vecino que reduce el costo de la ruta. El algoritmo termina cuando llega al nodo de destino o no puede encontrar una mejora en el costo.

7. **Recorrido del grafo:**

Se llama a la función hill_climbing para encontrar la mejor ruta desde "IFalls" hasta "Chicago". El resultado se almacena en best_route y best_cost.

8. **Impresión de resultados:**

Se imprime la mejor ruta encontrada y su costo.

Best First *(Búsqueda No Informada)*

1. **Importación de bibliotecas:**

Se importan las bibliotecas networkx y matplotlib.pyplot para trabajar con grafos y visualizarlos, además de la biblioteca heapq para administrar una cola de prioridad.

2. **Creación del grafo:**

Se crea un grafo vacío utilizando la función nx.Graph() de NetworkX, que se utiliza para representar las ciudades y las conexiones entre ellas.

3. **Adición de nodos y aristas:**

Se agregan nodos al grafo para representar diferentes ciudades de los Estados Unidos y se añaden aristas para representar las conexiones entre ellas, de manera similar a los ejemplos anteriores.

4. **Visualización del grafo:**

El grafo se visualiza utilizando la función nx.draw().

5. **Función calculate_cost:**

Se define la función calculate_cost(route, graph) para calcular el costo total de una ruta dada en el grafo. Esta función suma los pesos de las aristas en la ruta.

6. **Algoritmo Best First Search:**

Se define la función best_first_search(graph, initial_node, destination_node) que implementa el algoritmo Best First Search. El algoritmo utiliza una cola de prioridad para explorar las rutas más prometedoras primero. Comienza en el nodo inicial y busca el nodo de destino de manera iterativa.

7. **Recorrido del grafo:**

Se llama a la función best_first_search para encontrar la mejor ruta desde "IFalls" hasta "Chicago". El resultado se almacena en best_route y best_cost.

8. **Impresión de resultados:**

Se imprime la mejor ruta encontrada y su costo. Si no se encuentra una ruta, se imprime un mensaje de que no se encontró una ruta válida.

A* (A Star) *(Búsqueda No Informada)*

1. **Importación de librerías:**

El código comienza importando las siguientes librerías: networkx, utilizada para trabajar con grafos; matplotlib.pyplot, empleada para visualizar el grafo; y heapq, necesaria para mantener una cola de prioridad en el algoritmo A*.

2. **Creación del grafo:**

Se crea un objeto de grafo llamado G utilizando NetworkX. Este grafo representa ciudades y sus conexiones en Estados Unidos.

3. **Añadiendo nodos al grafo:**

A continuación, se agregan nodos al grafo para representar ciudades en Estados Unidos, como "IFalls," "GForks," "Bemindji," etc.

4. **Añadiendo aristas al grafo:**

Luego, se agregan aristas ponderadas al grafo para representar las conexiones entre las ciudades. Cada arista tiene un atributo "weight" que indica la distancia entre las ciudades.

5. **Visualización del grafo:**

Se utiliza nx.draw para visualizar el grafo con etiquetas en los nodos y colores específicos.

6. **Función calculate_cost:**

Esta función calcula el costo total de una ruta tomando en cuenta las aristas ponderadas en el grafo.

7. **Algoritmo A:***

La función a_star_search implementa el algoritmo de búsqueda A*. Realiza una búsqueda desde un nodo inicial a un nodo de destino en el grafo. Utiliza una cola de prioridad para explorar las rutas con menor costo primero, tomando en cuenta tanto el costo acumulado desde el nodo inicial como un costo heurístico estimado hasta el destino. El algoritmo realiza la exploración de las rutas en el grafo y se detiene una vez que encuentra una ruta desde el nodo inicial al nodo de destino o cuando se exploran todas las posibilidades. Devuelve la mejor ruta encontrada y su costo total.

8. **Búsqueda del camino:**

El código realiza una búsqueda A* desde el nodo "Rochester" hasta el nodo "Chicago" utilizando la función a_star_search.

9. **Resultados:**

Los resultados se imprimen en la pantalla, mostrando la mejor ruta encontrada desde "Rochester" hasta "Chicago" y su costo total. Si no se encuentra una ruta, se muestra un mensaje indicando que no se encontró una ruta válida.

Implementación y Evaluación

Las Implementaciones se realizaron mediante el uso de Jupyter Notebooks, usando lenguaje Python, además en el archivo se muestran las evaluaciones obtenidas.

Breadth First

BreadthFirst.html

Depth First

DepthFirst.html

Hill Climbing

HillClimbing.html

Best First

BestFirst.html

A* (A Star)

AStar.html

Conclusiones

En esta actividad, los estudiantes tienen la oportunidad de explorar y comprender en profundidad diversos algoritmos de búsqueda utilizados en inteligencia artificial. A través de la implementación de algoritmos de búsqueda no informada, como "Primero en Anchura" y "Primero en Profundidad", y algoritmos de búsqueda informada, como "Búsqueda Avara (Bestfirst)", "Hill Climbing", "A*", "Recocido Simulado", "Búsqueda Tabu", entre otros, los alumnos pueden desarrollar una comprensión práctica de cómo funcionan estos algoritmos y cuándo son apropiados para resolver problemas específicos.

El proceso de implementación y evaluación de estos algoritmos les brinda a los estudiantes la oportunidad de aplicar sus conocimientos teóricos en un contexto práctico. Pueden analizar la eficiencia y efectividad de cada algoritmo en función de la complejidad del problema y los resultados obtenidos.

La actividad fomenta la colaboración entre compañeros de equipo, lo que puede enriquecer el proceso de aprendizaje al permitir el intercambio de ideas y enfoques. Además, la entrega de archivos fuente y la presentación de resultados en un documento PDF aseguran que los estudiantes documenten y compartan su trabajo de manera adecuada.

En resumen, esta actividad es una oportunidad valiosa para que los estudiantes apliquen sus conocimientos teóricos en la implementación y evaluación de algoritmos de búsqueda, lo que les permite adquirir una comprensión más profunda de la inteligencia artificial y sus aplicaciones en la resolución de problemas.

Referencias

No se consultaron fuentes.