Inteligencia Artificial 2026

Lab 05

18.marzo.2026

En este laboratorio vamos a simular y comparar las diferencias entre varios métodos de búsqueda: (1) DFS, (2) BFS, (3) UCS, (4) Greedy Search, (5) A^*.

Recursos de Experimentación:

- Stanford - BFS/DFS: `cs.stanford.edu/people/abisee/tutorial/bfsdfs.html`.
- Stanford - A^*: `cs.stanford.edu/people/abisee/tutorial/astar.html`.
- Stanford - Custom & Dijkstra: `cs.stanford.edu/people/abisee/tutorial/customize.html`.
- PathFinding.js: `qiao.github.io/PathFinding.js/visual/`.
- Pathfinding Visualizer: `https://clementmihailescu.github.io/Pathfinding-Visualizer/`.

Ejercicios:

1. **El Laberinto de DBF y BFS.** Recurso: Pathfinding Visualizer o Stanford BFS/DFS.
   Genera un laberinto usando "Simple Random Maze". Ejecuta BFS y luego DFS.

   Instrucciones: Toma un screenshot de ambos resultados finales donde se vea el área total explorada (nodos visitados).

   Responder a las siguientes preguntas de discusión:

   - ¿Por qué BFS garantiza el camino más corto en este grid y DFS no?
   - Mencione una desventaja crítica de memoria para BFS y una de optimicidad para DFS.
   - ¿Cuál es la diferencia visual en la "frontera" de expansión de ambos?

   ¿Qué ocurre si analizamos las soluciones de BFS y DFS en un laberinto generado por "Recursive Division"?

2. **La Trampa de la Distancia.** Recurso: Pathfinding Visualizer o Stanford BFS/DFS.
   Dibuja muros y obstáculos y contrastar el desempeño de los algoritmos.
   Ejecuta Greedy Search y luego BFS.

   Instrucciones: Diseñar un muro tal que Greedy genere un camino mucho más largo que BFS. Adjunta el screenshot de tu "escenario trampa" y la solución de Greedy, así como la solución de BFS.

   Responder a las siguientes preguntas:

   - ¿Puedes dibujar las paredes de tal forma que Greedy Search resulte en un camino de mayor longitud que BFS?
   - Muestra un escenario donde Greedy toma una ruta ineficiente por seguir ciegamente la línea recta.

   Responder a las siguientes preguntas de discusión:

- ¿Es el algoritmo Greedy óptimo? ¿Por qué?
- ¿Cuál algoritmo es generalmente más rápido en ejecución y cuál explora mayor área (nodos visitados)?

3. **Heurísticas y Pesos.** Recurso: PathFinding.js o Stanford A^*.
   Crear un escenario con algunos obstáculos dispersos.
   Ejecutar A^* luego Dijkstra (UCS).

   Instrucciones: Mostrar screenshots comparando la cantidad de nodos explorados con ambas algoritmos, elabora una tabla comparativa donde se indique: el número de nodos explorados, el número de nodos en la frontera al final del algoritmo, el tiempo y número de pasos para hallar la solución, la distancia (en número de aristas) de la solución obtenida.

   Responder a las siguientes preguntas de discusión:

   - ¿Se puede alterar el mapa para que A^* termine significativamente más rápido que Dijkstra?
   - ¿Bajo qué condiciones Dijkstra y A^* (o weighted A^*) encuentran caminos distintos?

4. **El impacto de la métrica.** Recurso: PathFinding.js o Stanford A^*.
   Crear un escenario con algunos obstáculos dispersos.
   Ejecutar A^* usando Distancia Manhattan y luego Distancia Euclideana.

   Instrucciones: Mostrar screenshots comparando la cantidad de nodos explorados con ambas métricas, elabora una tabla comparativa donde se indique: el número de nodos explorados, el número de nodos en la frontera al final del algoritmo, el tiempo y número de pasos para hallar la solución, la distancia (en número de aristas) de la solución obtenida.

   Repetir varios de estos experimentos y Responder a las siguientes preguntas de discusión:

   - ¿Cuál métrica produce los caminos más cortos, Manhattan o Euclideana?

5. **El "Peor Escenario" (Stress-Test).** Recurso: Pathfinding Visualizer o PathFinding.js.
   Pensemos ahora en el problema, no de buscar el camino más rápido, sino entender cuándo un algoritmo falla o se vuelve ineficiente.

   Instrucciones: Crea un "laberinto falso" donde el camino que parece más directo (según la línea recta al objetivo) esté bloqueado al final por un muro largo, obligando al agente a retroceder casi hasta el inicio.
   Compara el número de nodos explorados por Greedy (que se lanzará al callejón sin salida) versus A^* (que debería balancear mejor la exploración).
   Toma un screenshot donde se vea la "mancha" de nodos visitados de ambos.

   Responder las siguientes preguntas de discusión:

   - ¿En qué casos específicos Greedy termina siendo "más rápido" pero con una solución desastrosa en términos de distancia?
   - ¿Existe algún escenario donde A^* visite más nodos que BFS? (Ayuda: Piensa en heurísticas optimistas vs. pesimistas o mapas muy abiertos).
   - En PathFinding.js, utiliza la herramienta de "Weight" para dibujar un río o pantano de alto costo que atraviese todo el mapa. Ahora coloca el inicio y el fin de modo que la línea recta cruce el pantano, pero exista un camino limpio (costo 1) dando un rodeo largo.
     Compara cómo Dijkstra y A^* manejan este peso. Muestra en un screenshot si A^* intentó "forzar" la entrada al pantano antes de rendirse.

- ¿Cómo afecta el "peso" de las celdas a la función $f(n) = g(n) + h(n)$?
- Si aumentamos el peso del pantano al infinito, ¿en qué algoritmo se convierte Dijkstra?