ISSD - Desarrollo de Sistemas de Inteligencia Artificial

Profesor: Sachi, Julio Mariano
Alumno: Garcia Alves, Andrés

GitHub: https://github.com/andres-garcia-alves/issd-dsia/tree/main/Proyecto-Final
Notebook: https://colab.research.google.com/drive/1pd5cmNriSxK3FQSHNF0YwUbl6c4cecE0
Deploy (Streamlit): https://issd-dsia-andres-garcia-alves.streamlit.app/

💬 1. Descripción resumida

Busco implementar un mini sistema de GPS. 🧭

Para ello, un agente inteligente buscará en un tablero generado de forma aleatoria la ruta óptima.

El sistema se compone de 2 bloques principales:

- Distintos algoritmos de búsqueda (informada y no-informada)
- Una red neuronal que elige el algoritmo de búsqueda más eficiente para un tablero dado.

Conecto de esta manera los 2 temas principales de la materia, a saber: algoritmos de búsqueda y machine learning.

🧩 2. Descripción general

Un agente debe desplazarse desde una celda inicial hasta una celda objetivo en el tablero (una grilla) que se genera aleatoriamente, y de dimensiones variables (ej. 4×4, 5x5, 6×6, 8×8, etc).

Cada celda tiene un costo distinto de movimiento (1, 2 o 3). Un costo de 1 viene a ser el costo 'normal' de movimiento, luego los costos mayores a 1 simulan ya sean caminos rurales o embotellamientos de tránsito. El agente debe encontrar la ruta de menor costo total.

Se implementaran distintos algoritmos de búsqueda (BFS, UCS, A*, Greedy), y luego una red neuronal aprenderá a predecir cuál de ellos es el más eficiente (por tiempo y/o costo total) según las características del tablero recibido.

⚙️ 3. Descripción técnica

Se dispone de un entorno tipo grilla/laberinto (por ejemplo 4x4, 6x6, etc) donde un agente busca la salida minimizando el costo total de movimiento, con casillas que pueden tener distintos pesos (costos).

Se implementan distintos algoritmos de búsqueda informada/no-informada:

- BFS (búsqueda a lo ancho)
- UCS (Uniform Cost Search)
- Greedy Best-First Search
- A* (con heurística de distancia Manhattan)

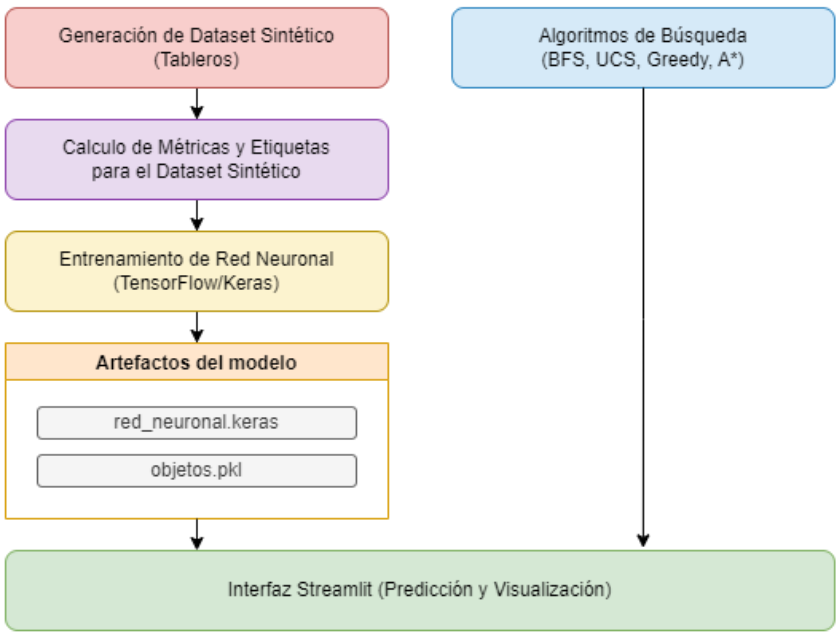
Luego, se agrega una red neuronal (en TensorFlow/Keras) para predecir cuál algoritmo de búsqueda conviene aplicar, dadas ciertas características del tablero (dimensión, cantidad y distribución de obstáculos, sus costos, etc).

Para entrenar la red neuronal se generá un dataset sintético con tableros de múltiples tamaños y disposiciones aleatorias.
Para ello, se reutilizan los algoritmos de búsqueda disponibles en el sistema, bajo el siguiente esquema:

- Generar un tablero de tamaño y disposición aleatoria
- Pasar el tablero generado por cada uno de los algoritmos de búsqueda disponibles
- Evaluar y guardar métricas de desempeño de cada algoritmo en el tablero actual
- Repetir N veces los pasos previos en distintos tableros, hasta generar suficientes datos de entrenamiento.

La red neuronal estará resolviendo un problema de clasificación: de los datos guardados, las columnas con las métricas serán los features para el entrenamiento, y la columna que identifica al algoritmo que generó esas métricas el target.

🏗️ 4. Diagrama de arquitectura



🏗️ 5. Estructura general del proyecto

📘 Proyecto_Final.ipynb
│
├── 1. Generación de tablero
├── 2. Implementación de algoritmos de búsqueda
├── 3. Creación del dataset sintético (features + target)
├── 4. Análisis exploratorio (EDA)
├── 5. Preprocesamiento de datos
├── 6. Red Neuronal (TensorFlow/Keras)
├── 7. Evaluación y métricas
└── 8. Conclusiones

🔍 6. Implementación de los algoritmos de búsqueda

Se implementan versiones simples de:

- BFS (Breadth-First Search) → ignora pesos (ideal para comparar).
- Uniform Cost Search (Dijkstra)
- Greedy Best-First Search → con heurística de distancia Manhattan.
- A* → heurística + costo acumulado.

Cada algoritmo retorna las siguiente métricas:

- costo_total
- nodos_expandidos
- tiempo_ejecucion

Con estas métricas se evalúan y comparan luego sus rendimientos.

🎞️ 7. Creación del dataset

Por cada tablero:

- Probarlo en los 4 algoritmos.
- Calcular y guardar en una fila:
    - tamaño
    - costo_promedio
    - varianza_costos
    - densidad_costo_alto
    - mejor_algoritmo

📈 8. Análisis Exploratorio (EDA)

Se utiliza Matplotlib y Seaborn para visualizar:

- Distribución de tamaños de tablero.
- Relación entre costo promedio y algoritmo óptimo.
- Frecuencia de cada algoritmo como "óptimo".

📈 9. Preprocesamiento de datos

- Chequeo de duplicados
- Chequeo de nulos
- Convertir etiquetas (nombre del algoritmo) a numéricas (LabelEncoder).
- Normalizar variables numéricas (StandardScaler)

🧠 10. Red neuronal en TensorFlow/Keras

- Un modelo simple de clasificación multiclase, una por cada algoritmo de búsqueda.

📝 11. Evaluación y métricas

Se incluirán:

- Exactitud (accuracy)
- Matriz de confusión
- Reporte de clasificación

🛠️ 12. Bibliotecas utilizadas

Propósito	Librería(s)
Generación y manipulación de datos	`numpy`, `pandas`, `scipy`
Visualización	`matplotlib`, `seaborn`, `PIL`
Implementación de búsquedas	`heapq` (para UCS y A*)
Red neuronal	`tensorflow.keras`, `pickle`
Preprocesamiento	`sklearn`
Miscelaneas	`time`, `random`, `tqdm`, `urllib`

💬 13. Análisis conceptual

El entrenamiento del modelo de red neuronal se puede entender como un proceso de búsqueda en el espacio de soluciones (pesos), donde el algoritmo de optimización (Adam) actúa como un agente que minimiza la función de pérdida (loss).

Este proceso es análogo a los algoritmos de búsqueda informa (como A*) que exploran un espacio de soluciones guiados por una heurística (la función de pérdida, en este caso).

Así, el modelo aprende a seleccionar el algoritmo de búsqueda más eficiente según las características del tablero, optimizando su desempeño en base a los datos con la experiencia previa (el dataset sintético con las simulaciones).

💎 14. Algunas posibles mejoras

Dataset sintético:
- Aumentar el tamaño de los tableros, por ej. 50x50 o más.
- Soporte de tableros rectangulares.
- Posiciones de inicio y meta aleatorias.

Algoritmos:
- Incorporar más algoritmos, por ej. Deep-First Search (DFS).

Red Neuronal:
- Explorar más arquitecturas (nro de capas, funciones de activación, etc) que mejoren la precisión del clasificador.

Deploy (Streamlit):
- Permitir al usuario que 'dibuje' (construya) su propio tablero.

📌 15. Observaciones

Se combinaron técnicas clásicas de búsqueda (BFS, A*, UCS, Greedy) con un modelo de red neuronal entrenado sobre tableros sintéticos.

La IA aprendió a predecir cuál algoritmo tendrá mejor desempeño según las características del entorno, funcionando como un meta-agente adaptativo.

El sistema no solo sugiere el algoritmo más eficiente según las condiciones del entorno, sino que también muestra visualmente la solución encontrada, permitiendo observar cómo varía la trayectoria óptima según el tipo de heurística y los costos del entorno.

Adicionalmente, de la sección de Evaluación del Desempeño, demuestro que:

- Los algoritmos clásicos funcionaron correctamente.
- La red neuronal aprendió a decidir inteligentemente cuál usar según las condiciones del entorno.
- Se integran las dos ramas de la materia: algoritmos de búsqueda y aprendizaje automático (TensorFlow/Keras).