Universidad de los Andes Facultad de Ingeniería	**Ingeniería de Sistemas y Computación** **ISIS-1611: Inteligencia artificial** Semestre: 2026-10 Créditos: 3	Accredited ABET Engineering Accreditation Commission

Taller 1: Búsqueda y Rescate

Bogotá ha experimentado un fuerte terremoto que ha dejado múltiples edificios colapsados y estructuralmente comprometidos. La Unidad de Gestión del Riesgo y Desastres ha desplegado una unidad de robot terrestre de búsqueda y rescate (SAR) de última generación para explorar las ruinas de varios edificios residenciales en el sector de Chapinero.

Su[1] misión es crítica: localizar y rescatar sobrevivientes atrapados entre los escombros de manera segura, eficiente y óptima. El robot debe navegar por múltiples edificios parcialmente destruidos, enfrentándose a terrenos peligrosos como agua estancada (proveniente de tuberías rotas), fuego y humo en áreas incendiadas, escombros que dificultan el movimiento, y muros colapsados que son completamente impasables.

Cada decisión de ruta importa: la batería del robot es limitada, cada segundo cuenta, y cada movimiento ineficiente reduce las probabilidades de rescate exitoso. El costo energético de transitar por diferentes tipos de terreno varía significativamente. Caminar sobre piso normal requiere poco esfuerzo, pero moverse entre escombros es lento y agotador, y cruzar agua estancada es especialmente peligroso y consume mucha batería.

Su responsabilidad es diseñar e implementar la lógica de navegación que permita al robot encontrar a los sobrevivientes de manera óptima, comparando diferentes estrategias de búsqueda, entendiendo cuándo cada una es más eficiente, y desarrollando heurísticas sofisticadas que permitan rescates simultáneos de varias personas.

Objetivos de aprendizaje

Al completar este taller, el estudiante será capaz de:

1. Modelar problemas de búsqueda como agentes que exploran estados y caminos posibles dentro de un entorno definido.
2. Implementar algoritmos clásicos de búsqueda desinformada (DFS, BFS, UCS) e informada (A*).
3. Diseñar heurísticas admisibles y consistentes aplicadas a escenarios reales de toma de decisiones.
4. Analizar el trade-off entre optimalidad, costo computacional y tiempo de respuesta, comparando diferentes enfoques de búsqueda.
5. Desarrollar pensamiento crítico y estructurado al traducir un contexto real de rescate a un modelo computacional.

[1] A lo largo del documento, las menciones a usted o su se refieren al grupo que presenta el taller

Escenario del Problema

El robot de rescate cuenta con las siguientes características operacionales:

- Puede desplazarse únicamente en 4 direcciones: arriba, abajo, izquierda y derecha.
- No puede atravesar muros ni salirse de los límites del mapa.
- Puede salvar una cantidad indefinida de sobrevivientes, pero debe llegar hasta cada uno.
- Cuenta con una cantidad limitada de energía. Con el fin de poder salvar a más personas, debe minimizar el uso de energía teniendo en cuenta los costos de cada tipo de terreno:

Símbolo	Tipo de Terreno	Costo por Paso	Descripción
.	Piso normal	1	Piso estructuralmente estable, sin obstáculos. Ideal para navegación rápida.
~	Agua estancada	2	Proveniente de tuberías rotas y sistemas de riego. Peligrosa para equipamiento electrónico, requiere cautela. Costo de batería elevado.
^	Escombros	3	Restos de estructuras colapsadas, vigas y cemento. Navegación lenta y arriesgada. Alto consumo de energía.
*	Fuego / Humo	5	Área incendiada actualmente. Extremadamente peligrosa. Costo de batería máximo. Debe evitarse si es posible.
%	Muro (impasable)	∞	Estructura colapsada, completamente impasable. El robot no puede ocupar esta celda bajo ninguna circunstancia.
S	Sobreviviente	N/A	Posición objetivo de rescate. Puede haber múltiples. No afecta movimiento (el robot puede ocupar la celda).
R	Posición inicial robot	N/A	Punto de inicio de la búsqueda. La exploración comienza aquí.

Estructura del Proyecto

Usted recibe un esqueleto base funcional que incluye toda la infraestructura de simulación y visualización. Su responsabilidad es completar los algoritmos de búsqueda y las heurísticas específicas en dos archivos designados:

- `algorithms/search.py`: Contiene las implementaciones de los algoritmos genéricos de búsqueda: DFS, BFS, UCS y A*. Aquí escribirá las funciones principales que resuelven los problemas de navegación del robot.
- `algorithms/heuristics.py`: Define las funciones heurísticas (como manhattanHeuristic) para búsqueda informada. Aquí adapta los algoritmos genéricos al contexto específico de búsqueda y rescate.

Para facilitar la implementación, usted cuenta con el archivo `algorithms/utils.py` que proporciona estructuras de datos eficientes como Stack, Queue y PriorityQueue. Aunque no debe modificar los demás archivos, puede consultarlos para entender detalles de implementación:

- Módulo `algorithms/`:
  - `agents.py`: Implementa la definición de un agente de búsqueda.
  - `problems.py`: Define la estructura de un problema de búsqueda y especifica los problemas de `SimpleSurvivorProblem` y `MultiSurvivorProblem`. Para cada uno define el estado inicial, los estados objetivo, los sucesores y los costos.

- Módulo `world/`:
    - `game.py`: Implementa las definiciones de agente, acciones, grilla y estado. Coordina el movimiento del robot, la detección de sobrevivientes y el término de la misión.
    - `rescue_layout.py`: Carga los archivos con formato `lay` (mapas de texto) a representaciones internas que el simulador puede procesar. Define cómo se interpreta cada símbolo del mapa.
    - `rescue_mission.py`: Maneja el flujo de ejecución de la simulación.
    - `rescue_rules.py`: Implementa las reglas del entorno incluyendo qué movimientos son válidos y el efecto de tomar una acción particular.
    - `rescue_state.py`: Mantiene la representación interna del estado del mundo (posición del robot, sobrevivientes rescatados, terreno).
- Módulo `view/`: Contiene módulos de visualización (`graphics_display.py` para interfaz gráfica, `text_display.py` para modo texto). Estos se ejecutan automáticamente según las opciones que usted especifique.
- Carpeta `layouts/`: Contiene archivos definidos en formato lay con mapas de prueba en formato texto. Cada mapa tiene una configuración diferente para probar distintos escenarios.

Ejecución

El programa se ejecuta con el siguiente formato general:

```
python main.py -p PROBLEM -f FUNCTION -l LAYOUT_FILE [OPCIONES]
```

Las opciones disponibles se describen a continuación:

Opción	Descripción	Default	Ejemplo
`-p`	Especifica el problema que debe resolver el agente. Puede consultar sus definiciones en `algorithms/problems.py`	Ninguno	`-p SimpleSurvivorProblem`
`-f`	Especifica la función de búsqueda a utilizar para resolver el problema. Por ejemplo: BFS, DFS o A*.	Ninguno	`-f tinyHouseSearch`
`-h`	Especifica la heurística que utilizará la búsqueda A*. Este campo es obligatorio si se utiliza esta función de búsqueda.	Ninguno	`-h manhattanHeuristic`
`-l`	Especifica el archivo de mapa a cargar (sin extensión .lay). Define el entorno donde se ejecuta la misión de búsqueda y rescate.	Ninguno	`-l tinyHouse`
`-t`	Activa el modo de visualización por texto en lugar de interfaz gráfica. Útil si tiene problemas con tkinter o no dispone de interfaz visual.	Desactivado	`-t`

`-q`	Genera salida mínima en consola, sin gráficos de ningún tipo. Ideal para debugging o cuando solo le interesa el resultado final.	Desactivado	`-q`
`-z`	Ajusta el tamaño de la ventana gráfica (si aplica). Un valor de 2.0 duplica el tamaño, 0.5 lo reduce a la mitad.	1.0	`-z 2.0`
`-x`	Especifica el tiempo de espera en segundos entre cada fotograma de la visualización. Un valor negativo requiere interactuar para avanzar paso a paso.	0.1	`-x 0.5`
`-r`	Guarda el historial de acciones del robot en un archivo. Útil para analizar posterior la ejecución.	Desactivado	`-r`
`-c`	Activa el manejo de excepciones durante la ejecución. Si ocurre un error, el programa intenta continuar en lugar de terminar.	Desactivado	`-c`
`--help`	Muestra la ayuda con todas las opciones disponibles.	—	`--help`

Puede probar que el proyecto se ejecuta correctamente utilizando el ejemplo `tinyHouse`:

```
python main.py -p SimpleSurvivorProblem -f tinyHouseSearch
               -l tinyHouse -x 0.5 -z 2.0
```

Tip: Analice el código de la función `tinyHouseSearch` en `algorithms/search.py` y el layout de `tinyHouse`. Note que sus algoritmos de búsqueda deben devolver una secuencia de acciones.

Nota: El proyecto utiliza la librería tkinter para mostrar una interfaz gráfica que visualiza el mapa, el robot y el progreso de la búsqueda. Si en su computadora tkinter no está instalado o genera errores, use el modo texto (`-t`) o el modo silencioso (`-q`).

Punto 1: Respuesta a Baliza de Emergencia (15%)

Un sobreviviente ha activado una baliza de emergencia desde una ubicación específica de uno de los edificios. El robot debe llegar a esa posición, pero no se encuentra programado para navegar por el terreno. Usted debe encontrar una primera solución rápidamente.

a) Implemente el algoritmo `depthFirstSearch`.
b) Pruebe su algoritmo sobre los layouts que encuentra en `layouts/simple`. Para esta misión debe modelar el problema como un `SimpleSurvivorProblem`.

Punto 2: Camino Más Corto a un Sobreviviente (15%)

En los mismos edificios se detecta a otro sobreviviente en una ubicación diferente. Esta vez, el equipo de rescate necesita minimizar el número de pasos para llegar: cada paso es peligroso, podría causar derrumbes adicionales

a) Implemente el algoritmo `breadthFirstSearch`.
b) Pruebe su algoritmo sobre los layouts que encuentra en `layouts/simple`. Para esta misión debe modelar el problema como un `SimpleSurvivorProblem`.

Punto 3: Navegación en Terreno Peligroso (20%)

Hasta ahora hemos encontrado caminos que atraviesan escombros, estanques de agua y áreas incendiadas. Sin embargo, la búsqueda se alarga y tenemos que minimizar el uso de batería del robot para poder rescatar a los sobrevivientes de todos los edificios. Ahora el objetivo no es minimizar pasos, sino minimizar el costo total.

a) Implemente el algoritmo `uniformCostSearch`.
b) Pruebe su algoritmo sobre los layouts que encuentra en `layouts/simple`. Para esta misión debe modelar el problema como un `SimpleSurvivorProblem`.

Punto 4: Búsqueda Informada Eficiente (20%)

El robot dispone de sensores que le permiten estimar la distancia aproximada al sobreviviente (aunque el camino no sea directo). Puede usar esta información para guiar su búsqueda, priorizando direcciones que lo acerquen al objetivo. Combinando información de costo real gastado + estimación de distancia restante, puede encontrar soluciones óptimas de manera mucho más rápida.

a) Implemente las heurísticas de distancia Manhattan y distancia Euclidiana.
b) Implemente el algoritmo `aStarSearch`.
c) Pruebe su algoritmo sobre los layouts que encuentra en `layouts/simple`. Para esta misión debe modelar el problema como un `SimpleSurvivorProblem`.

Punto 5: Rescate de Múltiples Sobrevivientes (20%)

Se han recibido reportes de múltiples sobrevivientes dispersos en edificios en la zona. El robot debe rescatarlos a todos en algún orden. El problema ahora es: ¿en qué orden debe visitarlos para minimizar el costo total?

a) Diseñe una heurística admisible e impleméntela en `survivorHeuristic`.
b) Pruebe su algoritmo sobre los layouts que encuentra en `layouts/multiple`. Para esta misión debe modelar el problema como un `MultiSurvivorProblem`. [2]

Punto 6: Reflexión sobre el trabajo realizado (10%)

La Unidad de Gestión del Riesgo y Desastres desea mejorar el funcionamiento de su robot SAR para incrementar las probabilidades de éxito de futuras misiones. Con este fin, le ha solicitado un análisis a partir de su experiencia diseñando algoritmos de búsqueda. Conteste las siguientes preguntas en un documento PDF:

a) (6%) Para cada uno de los puntos anteriores indique claramente en el contexto del problema:
    - Complejidad en espacio (Notación O).
    - Complejidad en tiempo (Notación O).
    - ¿El algoritmo es completo?
    - ¿El algoritmo encuentra la solución óptima siempre o en cuáles condiciones?
    - Para los puntos 4 y 5 indique si las heurísticas son consistentes.

[2] Algunas instancias del problema de rescate múltiple pueden ser particularmente complejas. Se recomienda que inspeccione los archivos de *layout* para empezar sus pruebas con aquellos que contienen un pequeño espacio de búsqueda. Si su heurística no le permite resolver alguna instancia, explique por qué.

Se espera que en su análisis compare cualitativamente el desempeño de los distintos algoritmos en los layouts que encuentre de interés, así como que compare cuantitativamente métricas como el costo de la solución, la cantidad de nodos expandidos y el tiempo.

b) (4%) Escriba una reflexión concisa sobre la co-construcción de la solución final con apoyo de la IA (vea política de uso de la IA más adelante), indicando, por ejemplo, qué aprendizajes obtuvo al ver las correcciones hechas por la IA, si éstas fueron básicas o solo mejoras secundarias; la facilidad o dificultad para crear el prompt adecuado, si la solución propuesta fué fácil de entender (¿usted la podría explicar en una sustentación?). Si no hizo ninguna iteración con IA, indique por qué no le pareció necesario ni útil. Aquí no hay respuestas correctas ni incorrectas. Este punto busca que usted se tome el tiempo de reflexionar sobre el uso de la IA en tareas de programación.

Política de Uso de IA Generativa y presentación en el trabajo a entregar

Para este taller se espera que usted desarrolle una primera versión de la solución de forma completamente autónoma, usando su propio criterio, conocimiento y creatividad antes de recurrir a herramientas de IA. La IA puede emplearse después para mejoras puntuales, refactorización, comentarios de calidad o apoyo en la corrección de errores, pero nunca como sustituto del esfuerzo personal ni como generador principal del código. Use estas herramientas con criterio profesional, asumiendo responsabilidad sobre su proceso de aprendizaje y entendiendo que lo que más valor tiene aquí es el camino que usted recorre para llegar a la solución, no solo el resultado final.

Todos los prompts y las versiones de código generadas con apoyo de IA deben quedar registrados dentro de los archivos del proyecto. Si utiliza IA para refactorizar u optimizar su solución, incluya en el archivo: (1) la versión inicial del código, (2) los prompts que utilizó para refinarla y (3) la versión final del código. Al finalizar, deje como código activo únicamente la versión final y conserve la versión inicial y todos los prompts en forma de comentarios dentro del mismo archivo.

Entrega del trabajo

- Guarde todo su trabajo (incluyendo el documento PDF) en una carpeta comprimida (.zip)
- Suba el trabajo al espacio correspondiente en Bloque Neón a más tardar la fecha y hora indicadas (un solo miembro del grupo debe subir el trabajo).
- Recuerde incluir los nombres y códigos de todos los integrantes de su grupo.

Nota: Al enviar su solución, usted declara que el código entregado en la primera versión de cada punto es de su autoría y que las versiones que utilizan IA fueron la respuesta a los prompts incluidos, que comprende el funcionamiento en todas las versiones y que acepta que este material pueda ser revisado, ejecutado y evaluado por el equipo docente para efectos académicos y de verificación de originalidad.