uc3m | Universidad Carlos III de Madrid

Ingeniería informática

2023-2024

Heurística y optimización
Satisfacción de Restricciones y Búsqueda Heurística

"Práctica 2"

Fernando Consiglieri Alcántara

100472111@alumnos.uc3m.es

David Andrés Yáñez Martínez

100451958@alumnos.uc3m.es

15 de diciembre de 2023

Índice general

1. INTRODUCCIÓN

Este documento se centra en la práctica de resolver problemas complejos mediante dos enfoques fundamentales: la Satisfacción de Restricciones (CSP) y la Búsqueda Heurística. El documento se divide en varias secciones que abordan diferentes aspectos de la práctica. Dichas secciones serán:

- Modelización y argumentación de las decisiones tomadas en la parte 1 con su respectivo análisis de los resultados.
- Modelización y argumentación de las decisiones tomadas en la parte 2 con su respectivo análisis de los resultados.
- Conclusiones acerca de la práctica.

2. PARTE 1

2.1. Modelización

Para abordar este desafío de asignación de plazas de manera óptima, se emplea un enfoque de Problema de Restricciones (CSP). Este modelo permitirá incorporar las particularidades del estacionamiento, tales como la exclusividad de plazas, restricciones específicas para vehículos con congelador, y consideraciones relacionadas con la maniobrabilidad dentro del aparcamiento.

A través de la formulación de variables, dominios y restricciones, se buscará encontrar una solución que cumpla con las necesidades particulares de cada tipo de transporte y garantice la eficiencia operativa dentro del entorno hospitalario.

La notación que vamos a utilizar es la siguiente:

$$Problema = \{X, D, C\}$$

El problema vendrá definido por el conjunto X de variables, el dominio D de estas y el conjunto C de restricciones.

Variables:

Las variables son cada ambulancia diferente puesta en el problema, cada una tiene un identificador para diferenciarlas entre ellas.

$$X = \sum_{i=1}^{n} (X_i)$$

Siendo “n” la última ambulancia para poder abarcar todas las del problema.

Dominio:

El dominio del problema está dividido en dos diferentes según el tipo de ambulancia:

$$D = \{Dominio\ completo, Dominio\ PE\}$$

Existe un dominio general definido por el tamaño del mapa (x, y).

$$Dominio\ completo\ = \{(i,j) \mid 1 \leq i \leq x, 1 \leq j \leq y\}$$

No obstante, no todas las ambulancias comparten este dominio. Aquellas sin congelador pertenecen al dominio completo, ya que pueden ubicarse en cualquier parte del mapa. En cambio, las ambulancias con congelador tienen un dominio más limitado, determinado por las plazas especiales que disponen de conexión a la red eléctrica.

$$Dominio\ PE = \{(i,j) \mid 1 \leq i \leq x, 1 \leq j \leq y\} \cap \{(i,j) | (i,j)\ es\ eléctrica\}$$

La ubicación de estas plazas se define según los datos del aparcamiento, que se proporcionan como parámetros.

Restricciones:

1. **Dos ambulancias no pueden tener la misma plaza:** Cada variable deberá tener un valor único

   $$\forall i,j\ \ (i \neq j) \Rightarrow (x_i \neq x_j)$$

   Siendo i, j dos ambulancias y $x_i\ x_j$ la posición de estas ambulancias

2. **Los vehículos con congelador solo pueden ocupar plazas eléctricas:** Esta restricción la definimos a la hora de establecer el dominio de cada ambulancia en el punto anterior.

3. **Un vehículo TSU no puede tener aparcado por delante un vehículo TNU:** Para cumplir esta restricción que solo nos permite tener aparcado por delante de un vehículo de tipo TSU otro vehículo del mismo tipo, podemos utilizar una función indicadora “I” y una desigualdad. La función indicadora “I” toma el valor de 1 cuando se cumple una condición y 0 en caso contrario.

$$\forall i,j\ \in\ vehículos, i \neq j: I(tipo_i = "TSU") * I(tipo_j \neq "TSU") * I(f_i = f_j) \Rightarrow pos_i\ \geq pos_j$$

Donde:

“i” y *“j”* son índices que representan diferentes vehículos.

$tipo_i$ y $tipo_j$ son los tipos de los vehículos *“i”* y *“j”*, respectivamente.

f_i y f_j son las filas en las que están aparcados los vehículos *"i"* y *"j"*, respectivamente.

pos_i y pos_j son las posiciones de los vehículos *"i"* y *"j"* en la fila.

Esta expresión matemática establece que si un vehículo *"i"* es de tipo TSU y está en la misma fila que otro vehículo *"j"* que no es de tipo TSU, entonces la posición de *"i"* debe ser mayor o igual que la de *"j"*, lo que significa que *"i"* no puede estar aparcado por delante de *"j".*

4. **Maniobrabilidad:** Todo vehículo debe tener una plaza libre a izquierda o derecha.

$$\forall i,j\ (x_{ij} = 1)\ \Rightarrow ((x_{i-1,j} = 0) \vee (x_{i+1,j} = 0))$$

Siendo x_{ij} una plaza del aparcamiento situada en las coordenadas i, j. La ecuación formaliza que, si un vehículo se encuentra aparcada en una plaza en concreto, la plaza inmediatamente superior o inferior tiene que estar libre (izquierda o derecha si miramos en dirección a la salida del aparcamiento).

Debido al enunciado del problema entendemos que las ambulancias que se encuentran en la primera y última fila no pueden tener ninguna otra ambulancia aparcada en las columnas inmediatamente inferior y superior respectivamente. Esto ha provocado que tengamos que hacer esta restricción más compleja:

Para las ambulancias en primera fila añadimos esta ecuación:

$$\forall i,j\ (x_{ij} = 1) \wedge (\mathrm{i} = 1) \Rightarrow \neg x_{i\text{-}1,j} = 0$$

Esto indica que si una ambulancia está ubicada en la primera fila (i = 1), entonces la plaza inmediatamente inferior debe estar libre.

Por otra parte, para las ambulancias en la última fila añadimos:

$$\forall i,j\ (x_{ij} = 1) \wedge (\mathrm{i} = \mathrm{n}) \Rightarrow \neg x_{i+1,j} = 0$$

Análogamente, si una ambulancia está en la última fila (i = n), entonces la plaza inmediatamente superior debe estar libre.

2.2. Análisis de los resultados

Nuestro modelo ha sido probado en distintos casos de prueba, que buscan probar diferentes escenarios y comprobar la correcta solución del ejercicio.

parking01.dat: Este caso, propuesto en el enunciado, representa un desafío considerable debido al tamaño del aparcamiento (5x6) y la complejidad de las

restricciones. Se generaron un gran número de soluciones (2175288), lo que demuestra la capacidad del programa para abordar situaciones complejas.

parking02.dat: Un caso más simple con un mapa de 2x2, que ofrece tres soluciones. Este escenario pone a prueba la capacidad del programa para manejar mapas más pequeños y destaca cualquier posible conflicto con las restricciones si no están bien adaptadas.

parking03.dat: Un caso de prueba sencillo diseñado para aplicar todas las restricciones sin introducir una complejidad significativa. Con 38 soluciones, este caso ayuda a verificar la correcta funcionalidad del programa en situaciones más simples.

parking04.dat: Un caso más específico con una sola fila en el mapa y una distribución variada de ambulancias. Este escenario permite evaluar la adaptabilidad del programa a casos de prueba más concretos.

parking05.dat: Un caso sin plazas eléctricas, pero sin vehículos con congelador. Este escenario valida la capacidad del programa para manejar situaciones sin restricciones específicas relacionadas con las plazas eléctricas.

parking07.dat: Un caso no válido con más ambulancias con congelador que plazas eléctricas. Este escenario pone a prueba la capacidad del programa para manejar casos de entrada incorrectos y generar soluciones coherentes.

parking08.dat: Un caso no válido que desafía la restricción de que una ambulancia TSU no puede tener un vehículo de otro tipo delante de ella. Este escenario destaca cómo el programa responde a restricciones específicas y cómo gestiona situaciones donde cumplir todas las restricciones es imposible. Esto se debe a que es un mapa con una sola fila y el único parking eléctrico está situado en la primera columna, el único vehículo con congelador es un TNU y el otro vehículo que es TSU no se puede colocar en ninguna de las otras posiciones.

En resumen, la evaluación en varios casos de prueba muestra la capacidad del programa para adaptarse a diferentes escenarios y cumplir con las restricciones del problema, identificando y manejando adecuadamente casos no válidos.

3. PARTE 2

3.1. Modelización

En el contexto de optimizar el servicio de traslado de pacientes, enfrentamos un desafío que requiere un enfoque estructurado. Este problema implica el transporte eficiente de pacientes desde sus domicilios hasta sus respectivos centros de tratamiento utilizando un vehículo colectivo eléctrico. A través de un modelaje de búsqueda heurística, buscamos minimizar el coste energético, considerando restricciones específicas como la capacidad del vehículo, la presencia de pacientes contagiosos y la necesidad de recargar energía en el aparcamiento.

1. Estados:
   Lo primero de todo es plantear el formato de los estados (o nodos) sobre los que vamos a actuar.

   En nuestro caso planteamos cada estado con el siguiente formato:

   $$E = (ubicación, energia, pacientes\ a\ bordo, pacientes\ restantes)$$

   Donde:

   - Ubicación: Representa la posición en el mapa.

     $$Ubicación = \{(x, y) \mid 1 \leq x \leq fila, 1 \leq y \leq columna\}$$

   - Energía: Representa la cantidad de energía que le queda a la ambulancia.

     $$Energía = \{x \mid 0 \leq x \leq 50\}$$

   - Pacientes a bordo: Lista de los pacientes que se encuentran actualmente en la ambulancia. En nuestro programa se representa de la siguiente manera:

     $$Pacientes\_a\_bordo = ['id', (x, y), \dots]$$

     - id: Representa el tipo de paciente, C o N
     - (x, y): Representa la ubicación del domicilio del paciente

   - Pacientes restantes: Lista de pacientes que todavía no han sido recogidos. En nuestro programa se representa de la siguiente manera:

     $$Pacientes\ restantes = \{'id': [(\mathrm{x_i}, \mathrm{y_i}), \dots], \dots\}$$

     - id: Representa el tipo de paciente, C o N
     - (x, y): Representa la ubicación del domicilio del paciente

1.1. *Estado inicial:*

El estado inicial se define por: La ubicación de la celda marcada como 'P'. La energía se inicializa en 50 unidades, la lista de pacientes a bordo se establece como vacía, y la lista de pacientes restantes incluye a todos los pacientes presentes en el mapa al momento de su lectura.

$$E_{inicial} = (ubicación\_parking, 50, [], todos\_los\_pacientes)$$

1.2. *Estado final:*

El estado final se caracteriza por: La ubicación en la celda 'P'. La cantidad de energía se mantiene en 50 unidades, indicando que la ambulancia ha regresado al parking. La lista de pacientes a bordo se vacía, ya que no quedan pacientes en la ambulancia, y la lista de pacientes restantes también se establece como vacía, indicando que todos los pacientes han sido trasladados a sus destinos respectivos.

$$E_{final} = (ubicación\_parking, 50, [], [])$$

2. Operadores:

a) Moverse a una celda adyacente: El vehículo puede moverse a una celda adyacente (arriba, abajo, izquierda o derecha), siempre que no sea una celda no transitable (X).

- **Precondiciones**:

  i. La celda destino debe ser adyacente y no debe ser una celda no transitable (X).

  ii. El vehículo debe tener suficiente energía para moverse a la celda destino.

- **Efecto**: El vehículo se mueve a la celda destino y se reduce su energía en la cantidad indicada por la celda destino.

b) Recoger un paciente: Si el vehículo pasa por el domicilio de un paciente (celda N o C) y hay espacio en el vehículo, el paciente se recoge.

- **Precondiciones**:

  i. La celda actual es el domicilio de un paciente (N o C)

  ii. Hay espacio en el vehículo

  iii. Las condiciones permiten recoger al paciente (por ejemplo, si el paciente es contagioso, debe haber espacio en las plazas para pacientes contagiosos).

- **Efecto**: El paciente se añade a la lista de pacientes a bordo y se elimina de la lista paciente restantes.

c) Dejar un paciente en su centro de atención: Si el vehículo pasa por un centro de atención (celda CN o CC) y hay pacientes a bordo que necesitan ser dejados allí, se dejan.

- **Precondiciones**:
  i. La celda actual es un centro de atención (CN o CC)
  ii. Hay pacientes a bordo que necesitan ser dejados allí.
- **Efecto**: Los pacientes correspondientes son eliminados de la lista pacientes a bordo.

d) Recargar energía: Si el vehículo pasa por el parking (celda P), puede recargar su energía.

- **Precondiciones**:
  i. La celda actual es el parking (P).
- **Efecto**: La energía del vehículo se recarga al máximo.

En la implementación en Python, todos estos operadores están integrados de manera orgánica en una única función que genera los sucesores de un estado dado. Esta función se encarga de acciones como 'moverse a una celda adyacente', y automáticamente verifica la naturaleza de la celda en la que se encuentra. Por ejemplo, si la celda es de tipo 'C', la función aplicará automáticamente el operador correspondiente de 'recoger paciente'. Este enfoque simplifica la lógica del programa, permitiendo que las acciones se ejecuten de manera cohesiva y eficiente en respuesta al contexto específico del estado actual.

3. Función de costo:

La función de costo en el contexto del algoritmo A* está representada por la variable f(s), que es el costo total estimado para llegar desde el estado inicial hasta el estado *'s'*. La función de costo se define como la suma de dos componentes:

$$f(s) = g(s) + h(s)$$

Donde:

- g(s) es el costo acumulado real de llegar desde el estado inicial hasta el estado *'s'*. Se calcula sumando los costos reales de las transiciones desde el estado inicial hasta el estado *'s'*. En este caso, se utiliza un costo de transición

predeterminado de 1, o el valor numérico presente en la celda correspondiente del mapa si es un número.

- h(s) es el costo heurístico estimado desde el estado *'s'* hasta el estado objetivo. Este componente es proporcionado por una de las cinco funciones heurísticas implementadas en la práctica. La elección entre estas heurísticas depende del número proporcionado como argumento en la terminal <num-h>.

La función de costo f(s) se utiliza para priorizar la expansión de los estados en la lista abierta durante la ejecución del algoritmo A*. Al seleccionar el próximo estado a expandir, el algoritmo elige aquel con el menor valor de f(s), lo que impulsa la búsqueda hacia soluciones más eficientes en términos de costo total estimado. A continuación, explicaremos las heurísticas diseñadas.

4. Heurísticas:

a) **Heurística 1:** Esta estrategia heurística busca la distancia mínima hacia todos los pacientes, derivada de la relajación de las condiciones sobre la cantidad y la identificación específica de los pacientes a bordo. Además, considera que una vez que se han recogido todos los pacientes, en dos simples movimientos (uno para dejar a todos y otro para regresar al estacionamiento), se alcanzará el estado final. Esta heurística también es admisible e informada. Para obtener una comprensión más detallada, se puede consultar el código de manera explícita.

b) **Heurística 2:** Esta heurística guía su comportamiento hacia la identificación del paciente más cercano; al llegar a él, presupone la recolección de todos los pacientes. Luego, los transporta al centro de atención más cercano y se dirige hacia el estacionamiento para concluir la tarea. De esta manera, relaja las restricciones sobre la cantidad específica de pacientes a bordo mencionadas anteriormente, considerando siempre que un único domicilio y centro de atención son suficientes para recoger y dejar a todos los pacientes. Esta heurística también es admisible e informada. Para una comprensión más detallada, se puede consultar el código de manera explícita.

c) **Heurística 3:** Esta estrategia heurística se deriva de relajar múltiples precondiciones y restricciones del problema. Se plantea un escenario en el que la teletransportación es posible, estimando el costo de trasladarse unitariamente a todos los domicilios de los pacientes y dejarlos instantáneamente en su centro (con un costo unitario), para luego regresar al estacionamiento (con un costo unitario). Además, se relajan las precondiciones de pacientes a bordo, es decir, número ilimitado de pacientes y sin importar su condición. Es crucial destacar que esta heurística es admisible e informada, ya que no sobreestima el costo de alcanzar la meta final en ningún estado. Para una comprensión más detallada, se puede consultar el código de manera explícita.

d) **Heurística 4:** En esencia, esta heurística es idéntica a la anterior, pero multiplicada por un número lo suficientemente grande como para dejar de ser admisible, ya que tiende a sobreestimar el costo. Su implementación se ha realizado con el propósito de encontrar soluciones subóptimas en mapas de mayores dimensiones, con una variabilidad según el número por el cual se multiplica, y en un rango de probabilidad.

5. Implementación en Python:

En este apartado vamos a aclarar algunas decisiones de diseños de aspectos claves de nuestro programa:

- Clase estado: Como hemos revisado previamente, la clase Estado está conformada por cuatro atributos: ubicación, energía, pacientes a bordo y pacientes restantes. Que no volveremos a explicar. Además de estos atributos, destacan tres métodos esenciales:
    - Método str: Este método se encarga de la representación en cadena de caracteres de todos los estados que constituyen la solución.
    - Método eq: Su función es comparar dos estados para determinar si son iguales. Es importante notar que la energía no se incluye en esta comparación, ya que, aunque es un atributo esencial del estado, no tiene la suficiente relevancia como para ser un factor distintivo entre estados.
    - Método lt: Este método se utiliza al ordenar estados con la misma heurística. Aunque el criterio comparativo no es de gran relevancia, ya que, con la misma heurística, el orden es indiferente, se priorizan estados con mayor energía en caso de empate.
- Algoritmo A*:
    - Optimización de Variables: ABIERTA (lista) y CERRADA (conjunto)se emplean para almacenar nodos aún no explorados y aquellos ya explorados, respectivamente, evitando así la exploración redundante de nodos. Ademas se implementan diccionarios (g, predecesores), y variables de tipos de datos incorporados como enteros y booleanos (f, N, EXITO, g_s, h_s, f_s, costo_transicion, numero).
    - Gestión de Memoria: La ocupación de memoria de nuestro algoritmo está directamente relacionada con la cantidad de nodos generados durante la búsqueda. En el peor escenario, si todos los nodos del espacio de estados son visitados, la memoria utilizada será proporcional al tamaño total del espacio de estados.
    - Complejidad: La complejidad temporal del algoritmo A* es O(b^d), donde b es el factor de ramificación (número promedio de sucesores por estado) y

d es la profundidad de la solución más corta. La complejidad espacial también es O(b^d) en el peor de los casos, ya que todos los nodos deben almacenarse en la memoria. Sin embargo, todo esto se ve mejorado con el uso de la heurística.

- Decisiones de diseño: Es necesario aclarar algunas decisiones de diseño que impactan en nuestro programa. El parámetro <num-h> tiene la responsabilidad de determinar la heurística que se utilizará en un momento dado. Sin embargo, es importante señalar que, si se ingresa un número mayor a 4 como heurística, se aplicará una por defecto. Esta heurística por defecto no representa una heurística en sí misma, sino que es la máxima de las tres heurísticas implementadas para cada estado. Esta elección nos permite diseñar la mejor heurística posible, ya que en cada instancia se selecciona la más adecuada para nuestro programa. Si pudiéramos representar gráficamente nuestras heurísticas, la expresión max(h1(s), h2(s), h3(s)) sería la heurística que estamos considerando.

  Cabe destacar que todas las heurísticas que incluyen distancias están implementadas siguiendo el modelo de distancia de Manhattan.

  También es importante comentar sobre los parámetros de salida del mapa, ya que puede que nuestro criterio no quede completamente claro.

  Por ejemplo, en los resultados obtenidos:

  - Coste total: 22
  - Longitud del plan: 23

  Para este caso específico, notamos que no es coherente que la longitud del plan sea mayor que el coste total. Sin embargo, esta discrepancia se debe a que en la longitud del plan se considera el estado inicial, el cual no tiene coste en el problema. Esta aclaración es simplemente informativa.

3.2. Análisis de los resultados

En el presente análisis, exploraremos los resultados obtenidos mediante la implementación del algoritmo A*, diseñado para encontrar la ruta óptima desde un estado inicial hasta un estado final.

En este estudio, analizaremos la calidad de las soluciones propuestas, la eficiencia computacional, y cómo las decisiones heurísticas influyen en el rendimiento del algoritmo en distintos escenarios. A través de la evaluación detallada de los resultados obtenidos en diversas instancias de prueba, buscaremos comprender el desempeño del algoritmo A* en términos de optimización de ruta, tiempos de ejecución y la capacidad de adaptación a diferentes condiciones de mapa. Este análisis proporcionará una visión integral de la eficacia y versatilidad del algoritmo A* en la resolución de problemas de navegación de mapas.

Comencemos examinando casos de prueba en los cuales encontrar una solución no es viable. Dado que no existe una solución, no nos enfocaremos en una heurística específica. Disponemos de tres archivos: mapa_NS1.csv, mapa_NS2.csv y mapa_NS3.csv.

En el primer mapa, solo se expande un nodo debido a que el área de estacionamiento está rodeada por celdas 'X', lo que impide encontrar una solución.

- Tiempo total: 0.0006506443023681641
- Coste total: 0
- Longitud del plan: 0
- Nodos expandidos: 1

En el segundo caso, la solución no es posible porque el centro de atención es inaccesible, lo que resulta en la expansión de todos los nodos hasta concluir que no hay solución viable.

- Tiempo total: 0.017597675323486328
- Coste total: 0
- Longitud del plan: 0
- Nodos expandidos: 736

Por último, en el tercer mapa, hay solo un paciente, pero este se encuentra a una distancia mayor a 50 unidades de energía, y la ambulancia no puede llegar hasta él.

- Tiempo total: 0.4665553569793701
- Coste total: 0
- Longitud del plan: 0
- Nodos expandidos: 4349

Ahora nos sumergiremos en el análisis de nuestras heurísticas. Para simplificar el estudio, nos centraremos en una distribución de mapa específica, ya que considero que los demás mapas comparten similitudes fundamentales entre sí. En este caso, nos referiremos al archivo mapa_small3.csv. A continuación, presentaré las estadísticas de salida correspondientes a cada una de las heurísticas.

Heurística 1:
Tiempo total: 2.416480541229248
Coste total: 36
Longitud del plan: 33
Nodos expandidos: 7875

Heurística 2:
Tiempo total: 7.079196453094482
Coste total: 36
Longitud del plan: 33
Nodos expandidos: 13194

Heurística 3:
Tiempo total: 16.612823486328125
Coste total: 36
Longitud del plan: 33
Nodos expandidos: 22895

Heurística X:
Tiempo total: 1.8873867988586426
Coste total: 36
Longitud del plan: 33
Nodos expandidos: 6186

Finalmente, la heurística X se establece como la heurística predeterminada, la cual, como mencionamos anteriormente, se activa al ingresar un número diferente a 1, 2, 3 o 4.

Como se puede apreciar, todas las heurísticas están debidamente informadas y son admisibles, ya que expanden la misma cantidad de nodos solución, lo que implica que alcanzan la solución óptima. Además, se observa que una heurística es más informada que otra, siguiendo la relación h3 < h2(s) < h1(s) < hx(n). La razón es evidente: la heurística más informada siempre será aquella que integre las mejores características de cada una, y en este caso, esa heurística es la heurística X. Por otro lado, la h3 se posiciona como la menos informada, dado que se diseñó relajando un gran número de restricciones del problema.

Además, cabe mencionar la heurística4, la cual, como ya discutimos anteriormente, no es admisible, pero tiene la capacidad de resolver mapas extensos en un corto periodo. Aunque su solución puede no ser óptima, destaca por su eficiencia. Esto se evidencia en los mapas: mapa_big.csv, mapa_big2.csv y mapa_extra_big, de dimensiones 10x10, 15x15 y 20x20 respectivamente, que cuentan con un elevado número de pacientes por recoger. A continuación, se presentan las estadísticas correspondientes a cada uno de estos mapas:

Mapa big 1:	Mapa big 2:	Mapa extra big:
Tiempo total: *0.29473400115966797*	Tiempo total: *2.9048314094543457*	Tiempo total: *77.4974000453949*
Coste total: *116*	Coste total: *184*	Coste total: *396*
Longitud del plan: *111*	Longitud del plan: *183*	Longitud del plan: *387*
Nodos expandidos: *2345*	Nodos expandidos: *10560*	Nodos expandidos: *82028*

4. CONCLUSIONES

Este proyecto ha sido un desafío considerable, se pueden destacar varias complejidades que hemos enfrentado. En primer lugar, la incertidumbre sobre la veracidad de nuestras soluciones, especialmente en ejercicios con mayor tamaño. Aunque la verificación en casos más pequeños es manejable, la duda sobre si todas las restricciones se están considerando plenamente ha sido un aspecto a tener en cuenta. Sin embargo, reconocemos que esta ambigüedad también ha sido una valiosa experiencia, preparándonos para futuras situaciones en las que la solución dependa de nosotros y no exista una respuesta única.

Otro desafío significativo se presentó en la segunda parte del proyecto, donde nos enfrentamos a un ejercicio bastante diferente a los vistos en clase. La falta de una meta clara y la dependencia de objetivos a corto plazo en cada estado generaron dificultades iniciales en la creación de una heurística adecuada.

En cuanto a puntos positivos, destacamos la habilidad adquirida para modelar y resolver problemas mediante la programación de restricciones (CSP) y la implementación de búsqueda heurística. Este enfoque ha fortalecido nuestra comprensión de cómo representar problemas de manera formal y cómo diseñar estrategias para su resolución.

En resumen, a pesar de los desafíos, hemos aprendido bastante de esta práctica ya que nos ha proporcionado habilidades para abordar problemas complejos y diversas metodologías de resolución.