Heurística y Optimización

Grado de Ingeniería en Informática. Curso 2023-2024

Práctica 2 - Satisfacción de Restricciones y Búsqueda Heurística

Link al repositorio de GitHub de la práctica:
Práctica 2 - 100451112- 100451258 - Repositorio de GitHub

Práctica realizada por los alumnos:
Darío Caballero Polo: 100451112@alumnos.uc3m.es
Miguel Domínguez Gómez: 100451258@alumnos.uc3m.es

1. Tabla de contenidos

2. Introducción

Esta memoria se ha estructurado de la siguiente manera:

Primero, se hará una descripción de los modelos:

- En la parte 1 se hablará sobre las *variables, dominios y restricciones* que se han planteado a partir del enunciado para el modelado del CSP.
- En la parte 2, se hará hincapié en la definición del *espacio de estados* y *operadores*, así como las distintas heurísticas implementadas para el *algoritmo A**.

Tras los modelos, tendrá lugar el análisis de los resultados obtenidos, donde se explicarán los resultados de las pruebas, su significado, si son los esperados o no y una *comparación de las heurísticas* en el caso de la parte 2, para poder sacar unas *conclusiones consistentes* de las mejoras e inconvenientes entre ellas.

Por último, se expondrán las conclusiones acerca de esta práctica donde se comentarán nuestras impresiones después del desarrollo de toda la práctica.

3. Parte 1: Validación con Python constraint

3.1. Descripción del modelo

Se ha de modelar el problema como un CSP, para lo cual se definen las variables, sus respectivos dominios, y las restricciones a las cuales se ven sometidas: $< X, D, R >$

Variables:
Se tomarán como variables los diferentes vehículos, debido a que esto permite definir una de las restricciones planteadas mediante el dominio, lo que simplifica ampliamente su modelización, frente a considerar las posiciones del parking como variables. Así, las variables se definen de la siguiente manera, pudiendo pertenecer a los conjuntos TNU y TSU, mutuamente excluyentes, y C y X, también mutuamente excluyentes. Una misma variable puede pertenecer a la vez a uno de los dos primeros conjuntos y a uno de los dos últimos.

$$TNU_{C_i}\ ,\ TNU_{X_i} \in TNU \equiv Vehículos\ no\ urgentes$$
$$TSU_{C_i}\ ,\ TSU_{X_i} \in TSU \equiv Vehículos\ urgentes$$

$$TNU_{C_i}\ ,\ TSU_{C_i} \in C \equiv Vehículos\ con\ congelador$$
$$TNU_{X_i}\ ,\ TSU_{X_i} \in X \equiv Vehículos\ sin\ congelador$$

Dominios:
El dominio de los vehículos con congelador abarca tan solo las plazas especiales, definidas en esta formalización por la función $PlazasEspeciales()$:

$$D_{TNU_{C_i}} = D_{TSU_{C_i}} = \{P_{xy} \mid x \in N,\ y \in M,\ PlazasEspeciales(P_{xy})\}$$

En cambio, el dominio de los vehículos sin congelador se compone de todas las plazas del parking:

$$D_{TNU_{X_i}} = D_{TSU_{X_i}} = \{P_{xy} \mid x \in N,\ y \in M\}$$

Se ha de destacar que N hace referencia al conjunto de índices de las filas del parking y M al conjunto de índices de las columnas. Ambos índices comienzan en 1.

Restricciones:

1. *Todo vehículo tiene que tener asignada una plaza y solo una.*
   Esta restricción se cumple inherentemente, pues una variable -vehículo- no puede tener asignados dos valores -plazas- al mismo tiempo. Además, para que el problema de CSP sea satisfacible, todas las variables han de tomar algún valor, por lo que todo vehículo tendrá asignada una plaza.
2. *Dos vehículos distintos no pueden ocupar la misma plaza.*
   Esta restricción, en cambio, no se cumple mediante las propiedades inherentes al CSP, pues impide que dos variables tengan asignadas el mismo valor, lo que podría ocurrir. Para solucionarlo, se añade la siguiente restricción:

$$\forall\, v_i = P_{xy} \mid v_i \in TSU$$
$$\nexists\, v_j = P_{x'y'} \mid v_j \in TNU,\ (x' = x) \wedge (y' > y),\ i \neq j$$

3. *Los vehículos con congelador solo pueden ir en las plazas especiales.*
   Esta restricción se ha implementado mediante la definición de un dominio específico para los vehículos con congelador:

$$D_{TNU_{C_i}} = D_{TSU_{C_i}} = \{P_{xy} \mid x \in N,\ y \in M,\ PlazasEspeciales(P_{xy})\}$$

4. *Un vehículo TSU no puede tener aparcado ningún vehículo TNU en las plazas que tenga delante, en su misma fila.*
   Esta restricción se aplicó forzando que para cada vehículo TSU, no pudiese existir ningún vehículo TNU que a su vez estuviese en su misma fila y en sus columnas sucesivas:

$$\forall\, v_i = P_{xy} \mid v_i \in TSU$$
$$\nexists\, v_j = P_{x'y'} \mid v_j \in TNU,\ (x' = x) \wedge (y' > y),\ i \neq j$$

5. *Todo vehículo debe tener libre o bien la plaza de su izquierda o bien la de su derecha.*
   Por último, esta restricción se dividió en tres casos debido a las consideraciones a tener en cuenta en los límites del parking:
   a. *El vehículo no se encuentra en la primera ni en la última fila del parking:*
      En este caso, para cada vehículo, no puede existir un par de vehículos que estén a su izquierda y derecha.

$$\forall\, v_i = P_{xy} \mid 1 < x < N$$
$$\nexists\, v_j = P_{x'y'},\ v_k = P_{x''y''} \mid (y' = y'' = y) \wedge (x' = x - 1) \wedge (x' = x + 1),\ i \neq j,\ i \neq k,\ j \neq k$$

b. *El vehículo se encuentra en la primera fila del parking:*
   En este caso, para cada vehículo, no puede existir un vehículo en la plaza de su derecha.

$$\forall\, v_i = P_{xy} \mid x = 1$$
$$\nexists\, v_j = P_{x'y'} \mid (y' = y) \wedge (x' = x + 1),\ i \neq j$$

c. *El vehículo se encuentra en la última fila del parking:*
   En este caso, para cada vehículo, no puede existir un vehículo en la plaza de su izquierda.

$$\forall\, v_i = P_{xy} \mid x = N$$
$$\nexists\, v_j = P_{x'y'} \mid (y' = y) \wedge (x' = x + 1),\ i \neq j$$

Se ha de saber que en todas las restricciones ha especificarse que $x \in N,\ y \in M$. Sin embargo, esto se omitió para simplificar la comprensión de las mismas.

3.2. Análisis de los resultados

Para poder comprobar la ejecución de nuestro modelo, se han desarrollado 10 tests entre los que se incluyen casos de ejecución correctos, que devolverán *soluciones factibles* y casos de ejecución imposibles, que *no deberán devolver ninguna solución*. El objetivo de estos tests es comprobar que las restricciones se cumplen, ya sea teniendo *casos límites*, *casos que violan las restricciones* o *casos* en los que sabemos de antemano que *el modelo debería actuar de una manera determinada* para que todo funcione correctamente. Para reducir el tiempo de ejecución, ya que debe computar todas las soluciones, se ha decidido reducir el parking a 4x4 en la mayoría de los tests.

Casos posibles:

1. **Ejemplo del enunciado:**
   Se ha creado este caso para comprobar que nuestro modelo consigue resolver el ejemplo de enunciado, que ya se sabía de antemano que iba a tener solución. Contamos con una matriz de 5x6 de parking, 6 plazas especiales [(1,1), (1,2), (2,1), (4,1), (5,1),(5,2)] y 8 vehículos [(1-TSU-C), (2-TNU-X), (3-TNU-X), (4-TNU-C), (5-TSU-X), (6-TNU-X), (7-TNU-C), (8-TSU-C)]. Este test ha producido alrededor de 2.2 millones de soluciones.

2. **Todo son plazas especiales:**
   El objetivo de este test era comprobar que las plazas especiales también pueden actuar como plazas estándar, sin crear ningún conflicto.
   Contamos con una matriz de 4x4 de parking, 16 plazas especiales (todas las posiciones de la matriz) y 6 vehículos [(1-TNU-C),(2-TSU-X),(3-TSU-C),(4-TNU-C),(5-TSU-C),(6-TNU-X)]

Este test ha calculado alrededor de 500000 soluciones. Ya que tenemos más huecos que vehículos, los vehículos pueden maniobrar y, como todo son plazas especiales, no hay problema con los vehículos con congelador.

3. **Hay más vehículos TNU que columnas:**
   El objetivo de este test es comprobar que, cuando se tienen más TNUs que columnas, debe ser obligatorio dejar este tipo de vehículos al fondo del parking para poder dejar pasar a los vehículos TSU.
   Contamos con una matriz de 4x4 de parking, 4 plazas especiales [(1,1),(2,1),(3,1),(4,1)] y 8 vehículos [(1-TNU-C), (2-TSU-X), (3-TNU-X), (4-TNU-X), (5-TSU-X), (6-TNU-C), (7-TNU-X), (8-TNU-X)]. Este test ha obtenido alrededor de 160000 soluciones. Como se explicó en el objetivo, al observar cómo son las soluciones, se puede ver como los vehículos TNU se colocan al final del garaje para dejar pasar a los vehículos TSU. Además, hay el mismo número de huecos que vehículos, por lo que los vehículos pueden maniobrar y por esa razón hay menos soluciones (hay menos permutaciones al tener más vehículos), y tampoco habría problema con las plazas especiales ya que hay menos vehículos con congelador que plazas especiales (y hueco entre ellas para poder maniobrar) .

En los 3 siguientes ejemplos, se pretende observar qué ocurre cuando se tienen más unidades de un tipo de vehículo frente a otro. En los 3 tests tenemos una matriz de 4x4 como parking, 4 plazas especiales [(1,1),(2,1),(3,1),(4,1)] y 6 vehículos:

4. En el primer test, tenemos más TNUs que TSUs [(1-TNU-X), (2-TNU-X), (3-TNU-X), (4-TNU-X), (5-TSU-X), (6-TSU-X)]. Tenemos un total de 546048 soluciones.
5. En el segundo test, tenemos el mismo número de TNUs que TSUs [(1-TNU-X), (2-TNU-X), (3-TNU-X), (4-TSU-X), (5-TSU-X), (6-TSU-X)]. Tenemos un total de 496512 soluciones.
6. En el tercer test, tenemos menos TNUs que TSUs [(1-TNU-X), (2-TNU-X), (3-TSU-X), (4-TSU-X), (5-TSU-X), (6-TSU-X)]. Tenemos un total de 546048 soluciones.

Casos imposibles:

7. **No hay suficientes plazas con congelador:**
   El objetivo de este test es comprobar que, cuando tenemos más vehículos con congelador que plazas especiales, no hay solución.
   Contamos con una matriz de 4x4 de parking, 1 plaza especial (1,1) y 2 vehículos con congelador [(1-TNU-C),(2-TNU-C)]. Como se ha comentado en el objetivo, nuestro modelo no devuelve ninguna solución y devuelve un error:

   ```
   ValueError: Sample larger than population or is negative
   ```

   Esto implica que no ha podido generar ninguna solución y, si se consulta el .csv generado, también se puede observar que el número de soluciones es 0.

8. **TSU con TNU bloqueando la salida:**
   El objetivo de este test es forzar esta situación para que nos devuelva un error o una salida con cero soluciones, ya que, si un TNU bloquea a un TSU, no puede ser solución.
   Contamos con una matriz más pequeña, de 1x4 de parking, para simular un callejón, 1 plaza especial (1,1) y 2 vehículos [(1-TSU-C),(2-TNU-C)]. Efectivamente, no obtenemos ningún resultado ya que no se puede dar una instanciación global en este problema.

9. **Hay más vehículos que plazas:**
   El objetivo de este test es ilustrar que si se da un problema con esta situación, definitivamente no va a tener solución.
   Para este test contamos con una matriz de 4x4 de parking, 4 plazas especiales [(1,1),(2,1),(3,1),(4,1)] y 9 vehículos (todos son vehículos TNU sin congelador). Al ejecutar este test, el modelo no va a encontrar solución posible, por lo que, si hay más vehículos que huecos, los vehículos no pueden maniobrar ya que mínimo van a haber 2 juntos, violando así una restricción del problema.

10. **Con plazas especiales juntas, 2 vehículos con congelador van a tener que estar juntos:**
    El objetivo de este test es demostrar que si tenemos el mismo número de vehículos con congelador que plazas especiales, y esas plazas están juntas, inevitablemente no va haber solución.
    Contamos con una matriz de 3x3 de parking, 2 plazas especiales [(1,1), (2,1)] y 2 vehículos (2 TNUs con congelador). Como se ha expuesto anteriormente, este test no tiene solución.

4. Parte 2: Planificación con Búsqueda Heurística

4.1. Descripción del modelo

Para modelar el problema de traslado de pacientes como un problema de búsqueda heurística, se han de definir el espacio de estados, los operadores, el estado inicial y el test de meta, así como el algoritmo y la heurística que se utilizarán.

Espacio de estados:
Cada estado está compuesto de los siguientes campos:

- **x**: fila del mapa en la que se encuentra la ambulancia.
- **y**: columna del mapa en la que se encuentra la ambulancia.
- **valor**: valor de la casilla justo antes de que la ambulancia se mueva a ella, para que cuando se pase por primera vez por la casilla de un paciente, el valor del estado sea dicho paciente, pero cuando se vuelva a pasar por esa casilla, el valor sea 1.
- **carga**: energía que le queda a la ambulancia.
- **ubi_n**: lista con las coordenadas de los pacientes no contagiosos que quedan por recoger.
- **ubi_c**: lista con las coordenadas de los pacientes contagiosos que quedan por recoger.
- **plazas_n**: lista que almacena los pacientes no contagiosos que ya han sido recogidos.
- **plazas_c**: lista que puede almacenar tanto pacientes no contagiosos como contagiosos.

Adicionalmente, en la implementación práctica de los estados, se añadieron los siguientes atributos para permitir el funcionamiento del algoritmo:

- **padre**: nodo padre del nodo actual. Permite recorrer el camino a la inversa tras encontrar una meta.
- **coste_g**: coste de la función g
- **coste_h**: coste de la función h
- **coste_f**: coste de la función $f = g + h$

Y los siguientes para realizar los cálculos de las heurísticas:

- **ubi_p**: coordenadas del parking.
- **ubi_cn**: lista con las coordenadas de los centros de atención de pacientes no contagiosos.
- **ubi_cc**: lista con las coordenadas de los centros de atención de pacientes contagiosos.

Operadores:

1. $MoverIzquierda(Estado)$
   - **Precondiciones:**
     $Estado_y \geq 1$
     $ValorIzquierda(Estado) \neq 'X'$
   - **Efectos:**
     $Estado'_x = Estado_x$
     $Estado'_y = Estado_y - 1$
     $Estado'_{resto\_de\_atributos} = AccionesCasilla(ValorIzquierda(Estado), Estado)$

2. $MoverDerecha(Estado)$
   - **Precondiciones:**
     $Estado_y < M - 1$
     $ValorDerecha(Estado) \neq 'X'$
   - **Efectos:**
     $Estado'_x = Estado_x$
     $Estado'_y = Estado_y + 1$
     $Estado'_{resto\_de\_atributos} = AccionesCasilla(ValorDerecha(Estado), Estado)$

3. $MoverArriba(Estado)$
   - **Precondiciones:**
     $Estado_x \geq 1$
     $ValorArriba(Estado) \neq 'X'$
   - **Efectos:**
     $Estado'_x = Estado_x - 1$
     $Estado'_y = Estado_y$
     $Estado'_{resto\_de\_atributos} = AccionesCasilla(ValorArriba(Estado), Estado)$

4. $MoverAbajo(Estado)$
   - **Precondiciones:**
     $Estado_x < N - 1$
     $ValorAbajo(Estado) \neq 'X'$
   - **Efectos:**
     $Estado'_x = Estado_x + 1$
     $Estado'_y = Estado_y$
     $Estado'_{resto\_de\_atributos} = AccionesCasilla(ValorAbajo(Estado), Estado)$

Funciones utilizadas en la definición de operadores:

1. $ValorIzquierda(Estado)$
   Devuelve el valor en el mapa de la casilla $x = Estado_x$; $y = Estado_y - 1$
2. $ValorDerecha(Estado)$
   Devuelve el valor en el mapa de la casilla $x = Estado_x$; $y = Estado_y + 1$
3. $ValorArriba(Estado)$
   Devuelve el valor en el mapa de la casilla $x = Estado_x - 1$; $y = Estado_y$
4. $ValorAbajo(Estado)$
   Devuelve el valor en el mapa de la casilla $x = Estado_x + 1$; $y = Estado_y$
5. $AccionesCasilla(Valor, Estado)$
   Ejecuta las acciones que se han de llevar a cabo al pasar por una casilla que contiene un determinado valor.
   - Si $Valor = 'P'$
     Se devuelve una $carga = Estado_{carga} - 1$
     Se devuelve un $valor = 'P'$
     Se devuelven el resto de atributos inalterados.

   - Si $Valor = 'N'$
     Se devuelve una $carga = Estado_{carga} - 1$
     Se comprueba si el paciente ya ha sido recogido. Si no, se comprueba si se puede recoger. Un paciente no contagioso se puede recoger si hay plazas no contagiosas libres o si hay plazas contagiosas libres y no hay ningún paciente contagioso en ellas. En caso de poderlo recoger, se lo recoge registrándolo en $plazas\_n$ o $plazas\_c$. Si hay plazas no contagiosas libres, se registra en $plazas\_n$. En caso contrario, en $plazas\_c$.
     Si el paciente ya había sido recogido, se devuelve un $valor = 1$. En caso contrario, se devuelve un $valor = 'N'$.
     Se devuelven el resto de atributos inalterados.

   - Si $Valor = 'C'$
     Se devuelve una $carga = Estado_{carga} - 1$
     Se comprueba si el paciente ya ha sido recogido. Si no, se comprueba si se puede recoger. Un paciente contagioso se puede recoger si hay plazas contagiosas libres y no hay ningún paciente no contagioso en ellas. A su vez, no debe haber ningún paciente no contagioso restante por recoger. Si esto se cumple, se registra en $plazas\_c$.
     Si el paciente ya había sido recogido, se devuelve un $valor = 1$. En caso contrario, se devuelve un $valor = 'C'$.
     Se devuelven el resto de atributos inalterados.

   - Si $Valor = 'CC'$
     Se devuelve una $carga = Estado_{carga} - 1$
     Se devuelve un $valor = 'CC'$
     Se comprueba si en $plazas\_c$ hay pacientes contagiosos. En tal caso, se descargan devolviendo $plazas\_c = [\,]$.
     Se devuelven el resto de atributos inalterados.

   - Si $Valor = 'CN'$
     Se devuelve una $carga = Estado_{carga} - 1$
     Se devuelve un $valor = 'CN'$

Se comprueba si hay pacientes no contagiosos en $plazas\_n$ o $plazas\_c$. Si esto es así, y además no hay ningún paciente contagioso en $plazas\_c$, se pueden descargar pacientes no contagiosos en el centro de atención. Si hay pacientes en $plazas\_n$, esta se devuelve como $plazas\_n = [\,]$ para descargarlos. Si hay en $plazas\_c$, esta se devuelve como $plazas\_c = [\,]$.
Se devuelven el resto de atributos inalterados.

- Si $Valor \in \mathbb{Z}$
  Se devuelve una $carga = Estado_{carga} - Valor$
  Se devuelve un $valor = Valor$
  Se devuelven el resto de atributos inalterados.

El coste de cada operador es de 1, excepto cuando $Valor \in \mathbb{Z}$, en cuyo caso el coste es $Valor$.

Estado inicial:

- **x** = ubi_inicial[0]
- **y** = ubi_inicial[1]
- **valor** = 'P'
- **carga** = 50
- **ubi_n** = ubi_n
- **ubi_c**: ubi_c
- **plazas_n** = [ ]
- **plazas_c**: = [ ]

Test de meta:

Un estado es meta si $Estado_{valor} = 'P'$, $Estado_{ubi\_n} = [\,]$, $Estado_{ubi\_c} = [\,]$, $Estado_{plazas\_n} = [\,]$ y $Estado_{plazas\_c} = [\,]$.

Esto es, si es un parking, no quedan pacientes no contagiosos por recoger, tampoco quedan pacientes contagiosos por recoger, las plazas de pacientes no contagiosos están vacías y las plazas de pacientes contagiosos también lo están.

Algoritmo de búsqueda:

Se implementó la versión del algoritmo A* que realiza la comprobación de los estados ya expandidos justo antes de expandir un nuevo nodo, pues es más eficiente.

Heurística:

Se implementaron 5 heurísticas distintas, ordenadas de peor a mejor:

1. Heurística 1:
   No es una heurística en sí misma, sino que implementa una búsqueda no informada (Dijkstra) devolviendo siempre un valor de 0.

2. Heurística 2:
   Devuelve 0 cuando no quedan pacientes por recoger, y 1 cuando sí quedan. Es mejor que la búsqueda no informada, pues aporta algo de información en el caso de que se recoja al último paciente, pero sigue siendo muy mala.

3. Heurística 3:
   Devuelve 0 cuando no quedan pacientes por recoger, y el número de pacientes que quedan por recoger en caso contrario. Es mejor que la heurística anterior, pues no solo proporciona información cuando se recoge al último paciente, sino cada vez que se recoge a un paciente.

4. Heurística 4:
   Esta heurística presenta un salto cualitativo frente a la anterior, pues tiene en consideración las distancias.

   - Si todavía quedan pacientes no contagiosos por recoger:
     - Se calcula la distancia de Manhattan entre la ambulancia y el paciente no contagioso más cercano.
     - A esta distancia, se le suma la distancia de Manhattan entre dicho paciente, y el paciente no contagioso más cercano a él.
     - Este proceso se repite con todos los pacientes no contagiosos que quedan por recoger.
     - A esta suma de distancias, se le suma la distancia de Manhattan entre el último paciente no contagioso y el paciente contagioso más cercano a él.
     - Se continúa este proceso hasta que no queden pacientes no contagiosos por recoger
     - Cuando no queden pacientes por recoger, se calcula la distancia de Manhattan entre el último y el parking y se suma.
     - Se devuelve el resultado de esa suma.
   - Si no quedan pacientes no contagiosos por recoger:
     - Se calcula la distancia de Manhattan entre la ambulancia y el paciente contagioso más cercano.
     - A esta distancia, se le suma la distancia de Manhattan entre dicho paciente, y el paciente contagioso más cercano a él.
     - Este proceso se repite con todos los pacientes contagiosos que quedan por recoger.
     - A esta suma de distancias, se le suma la distancia de Manhattan entre el último paciente contagioso y el parking.
     - Se devuelve el resultado de esa suma.

     Esta heurística es admisible porque los pacientes contagiosos no se pueden recoger a menos que ya se hayan recogido todos los pacientes no contagiosos. Si esta recogida no se hiciera en ese estricto orden, se rompería la admisibilidad. Se están relajando tanto las restricciones de movimiento por obstáculos y costes elevados, como de carga y espacio en las plazas de la ambulancia.

5. Heurística 5:
   Esta heurística es igual que la anterior, pero cuando no quedan pacientes por recoger, se tiene en cuenta la distancia entre la ambulancia y los centros de atención si sigue habiendo pacientes en la ambulancia. Cuando todavía quedan pacientes por recoger, se comporta igual que la heurística anterior.

- Si no quedan pacientes por recoger:
    - Si quedan pacientes contagiosos en la ambulancia:
        - Se calcula la distancia de Manhattan entre la ambulancia y el centro de atención de pacientes contagiosos más cercano.
        - Si quedan pacientes no contagiosos en la ambulancia:
            - Se calcula la distancia de Manhattan entre el centro de atención de pacientes contagiosos más cercano y el centro de atención de pacientes no contagiosos más cercano a este y se suma.
            - Se calcula la distancia de Manhattan entre el centro de atención de pacientes no contagiosos más cercano y el parking y se suma.
            - Se devuelve el resultado de esa suma.
        - Si no quedan pacientes no contagiosos en la ambulancia:
            - Se calcula la distancia de Manhattan entre el centro de atención de pacientes contagiosos más cercano y el parking y se suma.
            - Se devuelve el resultado de esa suma.
    - Si no quedan pacientes contagiosos en la ambulancia:
        - Si quedan pacientes no contagiosos en la ambulancia:
            - Se calcula la distancia de Manhattan entre la ambulancia y el centro de atención de pacientes no contagiosos más cercano.
            - Se calcula la distancia de Manhattan entre el centro de atención de pacientes no contagiosos más cercano y el parking y se suma.
            - Se devuelve el resultado de esa suma.
        - Si no quedan pacientes no contagiosos en la ambulancia:
            - Se devuelve la distancia de Manhattan entre la ambulancia y el parking.

Esta heurística es admisible porque los pacientes contagiosos se deben descargar siempre antes que los no contagiosos si ambos se transportan a la vez. Es mejor que la anterior heurística porque también aporta información cuando no quedan pacientes por recoger.

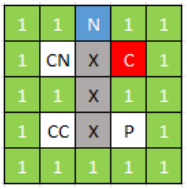
4.2. Análisis de los resultados

Para poder comprobar la ejecución de nuestro modelo, se han desarrollado 11 tests entre los que, al igual que en la anterior parte, se incluyen casos de ejecución correctos, que devolverán *soluciones factibles* y casos de ejecución imposibles, que *no deberán devolver ninguna solución*. El objetivo de estos tests es comprobar que el algoritmo de búsqueda resuelve correctamente el problema, respetando las restricciones en el orden de recogida y dejada de pacientes, el límite de carga, etc.

Para reducir el tiempo de ejecución, puesto que devolver la ruta óptima tiene un alto coste computacional, se ha decidido utilizar mapas de dimensiones 5x5 como máximo para los test.

Casos posibles:

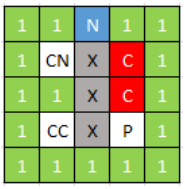
1. **Problema normal 5x5**
   Este ha sido el problema principal que hemos usado para comprobar que el algoritmo estaba correctamente implementado durante el proceso inicial de creación del modelo.



Con este problema ha sido posible encontrar una solución óptima en **alrededor de 0,17 s**, con un coste de **20** y **1011** nodos expandidos con la heurística menos informada y en **alrededor de 0,04 s**, con un coste de **20** y **453** nodos expandidos con la heurística más informada. En ambos casos, la solución posee una **longitud de 21**.

2. Un paciente no contagioso y 2 pacientes contagiosos:

Con este test se pretende comprobar que primero se coge al no contagioso y, Posteriormente, se coge a los dos contagiosos y se dejan en el centro de contagiosos, siguiendo las indicaciones del enunciado.



Con este problema ha sido posible encontrar una solución óptima en **alrededor de 0.47 s**, con un coste de **20** y **1656** nodos expandidos con la heurística menos informada y en **alrededor de 0,09 s**, con un coste de **20** y **611** nodos expandidos con la heurística más informada. En ambos casos, la solución posee una **longitud de 21**. Si se observa el camino, se puede ver que, aunque pase por casillas C de antemano, no es hasta después recoger al N cuando se cogen los C**.**

3. Más contagiosos que sitios en el vehículo:

Con este test se quiere comprobar que el vehículo hace varias descargas antes de poder recoger a todos los contagiosos. Este test lo creamos para observar, además, la parada intermedia en el parking para recargar.

Con este problema ha sido posible encontrar una solución óptima en **alrededor de 18.59 s**, con un coste de **58** y **11122** nodos expandidos con la heurística menos informada y en **alrededor de 11.51 s**, con un coste de **58** y **8370** nodos expandidos con la heurística más informada. En ambos casos, la solución posee una **longitud de 23**. Si se observa la ruta que toma, se puede ver que va a repostar en P y que hace varios viajes a CC.

4. Nueve no contagiosos en un vehículo:

En este test se comprueba que un no contagioso puede tomar los sitios para contagiosos cuando no hay contagiosos.

Con este problema ha sido posible encontrar una solución óptima en **alrededor de 5.40 s**, con un coste de **18** y **5581** nodos expandidos con la heurística menos informada y en **alrededor de 1.59 s**, con un coste de **18** y **3029** nodos expandidos con la heurística más informada. En ambos casos, la solución posee una **longitud de 19**. Si se observa la ruta que toma, se puede observar que coge en el mismo viaje a los 8 no contagiosos y los deja en el centro de no contagiosos.

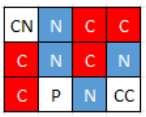
5. **Dos parkings:**

Este test se ha creado para comprobar que, incluso habiendo 2 parkings, se puede encontrar una solución.

Con este problema ha sido posible encontrar una solución óptima en con un coste de **3** y **9** nodos expandidos con la heurística menos informada y con un coste de **3** y **7** nodos expandidos con la heurística más informada. En ambos casos, la solución posee una **longitud de 4**. Si se observa la ruta que toma, se puede ver que sale de uno de los parkings y acaba en el otro. La resolución en ambos casos es tan rápida que no se registra el tiempo de ejecución.

6. **Todo pacientes y centros:**

Este test se ha creado para observar más detalladamente el comportamiento de nuestro modelo cuando no hay casillas numéricas (todas tienen un comportamiento especial).



Con este problema ha sido posible encontrar una solución óptima en **alrededor de 364.49 s**, con un coste de **24** y **30470** nodos expandidos con la heurística menos informada y en **alrededor de 24.05 s**, con un coste de **24** y **9143** nodos expandidos con la heurística más informada. En ambos casos, la solución posee una **longitud de 25**. Si se observa la ruta que toma, se puede ver que primero trabaja con los no contagiosos y luego recoge y lleva al centro a los contagiosos.

7. **Hay que regresar a recargar energía:**

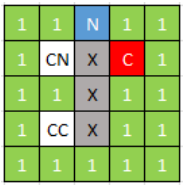
Este test se ha creado para comprobar que la ambulancia regresa a recargar energía si es necesario para completar su ruta.

Con este problema ha sido posible encontrar una solución óptima en **alrededor de 2.24 s**, con un coste de **136** y **3734** nodos expandidos con la heurística menos informada y en **alrededor de 2.05 s**, con un coste de **136** y **3583** nodos expandidos con la heurística más informada. En ambos casos, la solución posee una **longitud de 53**. A diferencia de lo ocurrido en el tercer test, en este el desvío para recargar se ha de realizar intencionadamente, pues no se halla en el camino.

<u>Casos imposibles:</u>

8. **Mapa sin parking:**

En este test se ha querido comprobar que, a falta de un estado inicial, el parking, no se puede ejecutar A* y, por lo tanto, no puede devolver una solución.

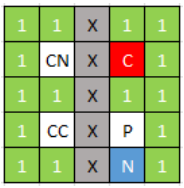


Como se ha anticipado correctamente, al ejecutar con cualquier heurística este test, nos devuelve este mensaje y no crea ningún archivo de salida:

```
ValueError: No se ha encontrado el estado inicial
```

9. Camino bloqueado:

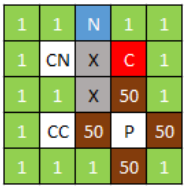
En este test se pretende colocar una columna de casillas X y, en cada lado, los pacientes, por uno, y los centros por otro, para ver que el algoritmo no se salta las casillas X y no devuelve solución.



Como se ha explicado previamente, independientemente de la heurística que se use, vamos a tener un archivo de salida que va a tener como contenido el texto "No hay solución". Además, todas las heurísticas expanden **618** nodos y dan un tiempo de **0.06 s**.

10. El vehículo se queda sin energía:

En este caso, se quiere comprobar que, si el vehículo se queda sin energía y no puede volver al parking desde las 4 direcciones, no encuentra solución.



Lo que se ha hecho para replicar esta situación es colocar alrededor del parking casillas de valor 50 para que se quede sin energía. El resultado que devuelve este caso cuando se ejecuta es el esperado, no encuentra solución. Expande **5** nodos: el inicial y las 4 casillas adyacentes. El tiempo de ejecución es tan breve que no se registra.

11. El vehículo no puede recoger a todos los pacientes por falta de energía:

Este caso es muy parecido al anterior solo que con la variación de que el vehículo se queda sin energía a medio camino de dejar a los pacientes en su centro.

La idea de este test era ver que se ejecutaba el algoritmo, lo cual el anterior test no enseñaba ya que solo se centraba en comprobar que se podía quedar sin energía el vehículo. Como se suponía, no devuelve una solución tampoco. Expande **1716** nodos y se ejecuta en **0.43 s**.

5. Conclusión

Esta práctica nos ha hecho comprender en profundidad los detalles del modelado de un problema en CSP desde una perspectiva más práctica, con su implementación con Python Constraint. Adicionalmente, el redactar la formalización matemática de las restricciones nos ha hecho enfocar el problema desde una perspectiva que los ejercicios convencionales no abarcan.

En cuanto a la parte de búsqueda heurística, fue todo un reto idear una forma de modelar el espacio de estados de cara a la implementación. Sin embargo, una vez se logró, la definición de los operadores para la expansión de los nodos, así como las heurísticas, se plantearon de forma orgánica. Es realmente satisfactorio desde el punto de vista académico e incluso personal ver cómo la ambulancia va siguiendo la ruta respetando todas las restricciones que se se exigían, demostrando la correcta implementación del problema.