Examen Parcial de IA

(19 de abril de 2021)
Duración: 90 min

1. (6 puntos) El mago Rincewind debe componer un hechizo a partir de conocimiento arcano que se encuentra en diversas torres esparcidas por el mundo, cada una con una parte única y indispensable. Hay un total de T torres, y desde cada torre se puede viajar a cualquier otra torre, pero la distancia puede variar y a Rincewind no le gusta andar sin sentido, así que quiere hacer el recorrido con menor distancia posible.

   Sin embargo, el problema no acaba aquí. **Una proporción bastante alta** de estas torres se encuentra cerrada bajo sellos arcanos que requieren de llaves (en forma de hechizos). Estas llaves se encuentran en algunas de las otras torres que Rincewind tiene que visitar. Por ejemplo, para visitar(y acceder a) la torre A Rincewind requiere de haber visitado la torre B o la torre C (ambas contienen el hechizo para abrir la torre A). Generalmente, el hechizo de cada torre se puede encontrar en una o dos torres, no en muchas más. En consecuencia, el recorrido de Rincewind debe tener en cuenta un conjunto de preordenes entre las torres. El sistema está montado de forma que todas las torres son accesibles. Es decir, no puede darse que haya un ciclo entre los preordenes como que el hechizo que abre la torre A está solo en la torre B y el que abre la torre B, solo en A (este problema no tendría solución).

   Rincewind dispone de una serie de propuestas para organizar la ruta, que puede programar para que Hex (el superordenador de la universidad) saque como salida el órden de visita a las torres, minimizando la distancia.

   a) Usar el algoritmo de A*. Definimos el estado como la asignación de torres a la ruta. El estado inicial es la ruta vacía. Tenemos un operador `visitar_torre` que asigna la siguiente torre a la ruta, el coste es la distancia entre la torre actual y la siguiente escogida. Como función heurística usamos $h(n) = M * P + k * L$, donde M es la distancia media entre dos torres, P es el número de torres a las que todavía no hemos ido (*pending*), k es un factor de ponderación y L es el número de torres todavía selladas (*locked*, es decir, el número de llaves por recoger).

   > El algoritmo A^* está pensado para encontrar la solución óptima. Si entendemos que el problema nos pide encontrar la configuración que tarde *el mínimo* tiempo posible, es el algoritmo adecuado para ello. Si entendiéramos que nos pide minimizar en lo posible el tiempo sin encontrar el óptimo, el A^* no justificaria su coste elevado para la resolución del problema, ya que introducimos una complejidad algorítmica innecesaria. Debido a los problemas detallados más adelante (en concreto, la dificultad de hallar un heurístico admisible para este problema), A^* es de entrada una mala propuesta.
   >
   > A priori el espacio de búsqueda está bien definido. Se plantea el problema como un espacio de estados en el que partimos de un estado inicial vacío (correcto), y cuyo operador de 'visitar torre' parece correcto inicialmente. Su factor de ramificación es $O(T)$, y permitiría explorar todo el espacio de estados. El operador no tiene en cuenta de ninguna forma el requisito de que la torre esté 'abierta', es decir, que pueda ser visitada. Esto se puede arreglar añadiendo una condición de aplicabilidad. *En ningún caso* esto se puede 'solventar' añadiendo penalización al heurístico, ya que no es como funciona el método. Además, si añadieramos una precondición para no revisitar torres, el algoritmo generaría menos sucesores (y sería más eficiente). El estado final debería ser aquel que tenga 'todas las torres visitadas'.
   >
   > El problema principal de esta propuesta es que es prácticamente imposible encontrar un heurístico admisible e informativo para este problema. El heurístico propuesto considera 'la distancia media entre dos torres'. Sea esta distancia media 'entre cualquier par de torres del problema' o entre las torres restantes, con alta probabilidad, $M(n) * P(n) > h^*(n)$ (no admisible). Para ver esto, basta solo con considerar el caso con tres torres en linea (a distancia X) y Rincewind en una de las torres extremo. La distancia media sería $(X + X + 2X)/3 = Y > X$, dando $h(n) = 2 * Y + ...$ pero parece obvio que el coste de una asignación podría ser perfectamente $2 * X$. Este es solo un caso de ejemplo, y contra más torres se den, mayor es la probabilidad de que este heurístico sea altamente superior a h^*. Es más, el otro sumando (en caso de $k > 0$, que es el único caso con sentido) solo empeoraría la situación. (*NOTA: La descripción detallada en esta solución no es representativa de la explicación que esperábamos, una pequeña argumentación de que el heurístico es no admisible bastaba, pero creemos informativo incluir un análisis más profundo en la solución que os proponemos*).
   >
   > En definitiva, esta solución sería bastante impráctica, ya que con un heurístico no admisible no aseguraríamos optimalidad. Entorno a eficiencia, A* está en el espectro de alta complejidad, así que es poco eficiente: aunque arreglásemos las condiciones de aplicabilidad del operador, y estaríamos en un caso de 'búsqueda metódica' parecida a un BFS.

   b) Usar el algoritmo de Hill Climbing. La solución inicial se contruye añadiendo todas las torres en orden alfabético. Como operador de búsqueda usamos `swap(torre1, torre2)`, que comprueba que el intercambio de torres en la ruta cumpla los pre-ordenes impuestos por los sellos. La función heurística es la suma de las distancias entre torres consecutivas en la solución.

   > Inicialmente Hill Climbing parece un método correcto para solucionar el problema, ya que es un problema complejo en el cual queremos una solución que minimiza un criterio (distancia de la ruta), pero no requerimos que sea el óptimo global. Sin embargo, la solución propuesta tiene una serie de errores no triviales.

Empezamos por la heurística, que de entrada es correcta ya que claramente busca minimizar el criterio de calidad descrito en el enunciado (un recorrido con la menor distancia posible).

La solución inicial planteada puede ser no solución (con alta probabilidad), ya que no tiene por qué respetar los preordenes. Esto se puede solucionar fácilmente cambiando el esquema de generación de alfabético a un toposort (para que se respeten los preordenes), o incluso un algoritmo propio que escoja la siguiente torre de forma aleatoria de entre las abiertas, y actualice el conjunto de torres abiertas con los preordenes de dicha torre (coste $O(T)$ si se programa inteligentemente). Con un algoritmo estocástico, podríamos también generar 'diferentes soluciones iniciales', permitiendo re-ejecuciones del algoritmo para llegar a mínimos diferentes. Alternativamente (aunque de forma menos deseable) se podría penalizar los preordenes no cumplidos en la heurística para permitir no soluciones. Discutimos esta posibilidad más adelante.

El operador propuesto parece a priori correcto, ya que su condición de aplicabilidad nos mantiene en el espacio de soluciones. Su factor de ramificación es $O(T^2)$ (concretamente: $T(T-1)/2$). Sin embargo, el problema principal de este operador es que tiene una altísima probabilidad de no explorar todo el espacio de soluciones dado el heurístico y las palabras clave del enunciado 'una proporción bastante alta de las torres están selladas'.

Imaginemos un caso en que existe una ruta 'XABY', donde X, A, B e Y son 'secuencias de torres que cumplen preordenes'. Supongamos que A y B son secuencias con preordenes estrictos $A_1 -> A_2 -> A_3 ...$ (cada una antes que la siguiente). Un operador de swap nos dificultaría la tarea de mover el bloque B antes que el bloque A (ya que la única forma sería intercambiar el primero de B con el último de A, con un número de pasos total del orden de $|A| * |B|$). Además, aunque 'XBAY' fuera muchísimo mejor que 'XABY', para llegar a esta solución, deberíamos conseguir que cada uno de esos pasos mejorara la solución anterior (altamente improbable). Con el comentario del enunciado sobre preordenes, situaciones como la descrita en este párrafo serían abundantes e impedirían explorar la mayor parte del espacio de soluciones, dejándonos soluciones poco deseables (situación de picos de sierra).

Un ejemplo más sencillo, sería no poder swappear una torre A por una C dado un recorrido XABCY ya que B debe ser anterior a C, aunque XBCAX es mejor. Con suficientes preordenes, el número de mínimos locales escala muy rápido con mucha probabilidad.

Un operador de *shift torre* (mover una torre X pasos hacia adelante o hacia atrás) sería mucho mejor y tendría mejor sinergia con cualquier heurístico, manteniendo el factor de ramificación. En este caso nos encontraríamos en una aplicación del algoritmo de búsqueda local más correcta, y llegaríamos a óptimos más aceptables.

Como posibilidad alternativa, podríamos permitir movernos dentro del espacio de no soluciones añadiendo un factor de penalización lo suficientemente grande al heurístico. Sin embargo, deberíamos tener en cuenta que un factor lo suficientemente grande nos dejaría en la misma situación que en el caso de tener condiciones de aplicabilidad, mientras que un factor demasiado pequeño nos llevaría a salirnos del espacio de soluciones y devolver no soluciones.

c) Usar un algoritmo de satisfacción de restricciones. El grafo de restricciones tendría como variables las torres, los dominios son un número natural (de 1 a T) indicando el orden de cada torre en la ruta. Como restricciones se propone una restricción n-aria tal que se conserven los preordenes establecidos por los sellos arcanos, y otra restricción n-aria para comprobar que la distancia de la ruta sea menor que una distancia D (que iremos disminuyendo y re-ejecutando a medida que encontramos soluciones hasta encontrar la óptima).

A priori, usar un algoritmo de CSP no parece tener sentido, ya que estamos en un problema de optimización y el CSP está pensado para dar una asignación psoble que cumpla unas restricciones. Sin embargo, cuando añadimos la restricción sobre la distancia máxima, conseguimos añadir la funcionalidad de optimización intentando asegurar una asignación con una calidad mínima, lo cual nos permitiría resolver el problema. De hecho, como obtendremos soluciones con distancias cada vez menores, podemos detener el algoritmo en cualquier momento. Si cogemos la última solución propuesta antes de llegar a un conjunto de restricciones no satisfactible, significa que no existe ruta con menor recorrido y habremos llegado al óptimo global. De todas las propuestas de este enunciado, esta es la única que lo conseguiría, a costa de una (posible) menor eficiencia temporal.

A nivel de correctitud, la asignación de valores a restricciones nos dará una ruta, que es lo que buscamos. Sobre las restricciones, falta una para asegurar que toda torre se visita en una posición diferente (eg. no hay dos torres que se visiten a la vez). Para ello, nos basta con un conjunto de restricciones binarias de desigualdad entre todas las variables.

A nivel de eficiencia, dado el modelado, disponemos de 3 tipos de restricción: la que acabamos de introducir ($T(T-1)/2$ restricciones binarias), la restricción sobre preórdenes, y la restricción de distancias (1 restricción T-aria).

La restricción sobre preórdenes es muy ineficiente tal como está planteada en el enunciado y dado el algoritmo de arco-consistencia. Sin embargo, esta restricción se puede simplificar muchísimo si se descompone. Cada 'preorden' del enunciado se puede re-enunciar como una restricción única entre 2 o 3 variables (como A toma valor mayor que B o que C si el preorden es B o C antes que A). Este modelado simplifica muchísimo el problema, y hace que a pesar que a priori parezca un mal algoritmo para el problema, se trate de un modelado muy rápido y que funcionaría mejor que las alternativas propuestas. (*NOTA: Algunos han planteado que una*

restricción de preorden entre tres torres se puede expresar como varias variables binarias. Esto no es correcto, ya que requiere una disyunción (A<B o C<B), pero se ha aceptado ya que no requeríamos demasiado detalle).

El problema requiere de correr el algoritmo muchas veces, pero no debería ser excesivamente costoso ya que no es necesario empezar de 0 cada vez (se puede cambiar la restricción en ejecución), ni es necesario esperar a la solución óptima (la calidad de la solución es proporcional a la paciencia que tengamos dejando el algoritmo correr).

Comenta cada una de las soluciones que se proponen, analizando si la técnica escogida es adecuada para este problema, si cada uno de los elementos de la solución son correctos o no (cada uno por separado y en conjunción los unos con los otros). Incluye un análisis de los costes algorítmicos y/o factores de ramificación allá donde sea necesario. Justifica tu respuesta.

2. (4 puntos) Como una de sus medidas de reactivación económica para salir de la crisis provocada por la COVID-19, y aprovechando la gran proporción de población jubilada que ya está vacunada, el Ministerio de Sanidad, Consumo y Bienestar Social quiere proporcionar viajes gratuitos a los jubilados a través del IMSERSO. Para ello, el Ministerio intenta repartir a los J jubilados que se apuntan a la iniciativa anualmente entre los H hoteles que se han presentado al plan de ayudas del sector por la COVID-19 (no hay restricciones sobre J y H, pero claramente $J \gg H$). Cada jubilado j puede tener o no una discapacidad de tipo K(cognitiva), M(medicalización continua) o R(movilidad reducida), y puede presentar al IMSERSO un máximo de tres lugares a los que prefiere viajar. Por otra parte, cada hotel h está asociado a un lugar, dispone de un número de plazas PL_h a un precio único (cada hotel tiene un precio por plaza PR_h) y puede estar habilitado para discapacidades de tipo K, M y/o R, o no estarlo para ninguna.

   El reparto debe:

   - maximizar el número de jubilados con plaza asignada,
   - minimizar el precio total que se deberá gastar el Ministerio,
   - priorizar la asignación de jubilados que han viajado menos a través, del IMSERSO.
   - intentar respetar al máximo las preferencias de los jubilados,
   - respetar las discapacidades de los jubilados.

   En los siguientes apartados se proponen diferentes alternativas para algunos de los elementos necesarios para plantear la búsqueda (solución inicial, operadores, función heurística, ...). El objetivo es comentar la solución que se propone, analizando si la técnica escogida es adecuada para este problema, si cada uno de los elementos de la solución son correctos o no (cada uno por separado y en conjunción los unos con los otros). Incluye un análisis de los costes algorítmicos y/o factores de ramificación allá donde sea necesario. Justifica tu respuesta.

   Sea:
   A = ♯jubilados asignados a plazas,
   B = ♯jubilados con preferencias satisfechas,
   C = suma de frecuencias de viajes de los jubilados asignados,
   D = precio total de la asignación,
   E = suma de los precios de todas las plazas de todos los hoteles,
   F = número total de plazas de los H hoteles.

   a) Queremos usar Hill-climbing, tomando como solución inicial la asignación de F jubilados con menos frecuencia de viajes, intentando llenar primero los hoteles más baratos. Como operador de búsqueda usamos `intercambiar(plaza, jubilado1, jubilado2)` donde *jubilado1* está asignado a la plaza en la solución actual, *jubilado2* no está asignado a la solución actual, y la *plaza* cumple las posibles discapacidades del *jubilado2*. La función heurística será:

$$h'(n) = A + B + C + D$$

   Plantear como solución un Hill Climbing para este problema es adecuado a priori, ya que nos piden encontrar una solución maximizando o minimizando una serie de criterios (en este caso A, D, C y B) sin necesidad de obtener el óptimo.

   La solución inicial puede ser no-solución, ya que está teniendo en cuenta los criterios C (frecuencia de viajes) y D (coste de las plazas asignadas) pero no hay garantía de que, por ejemplo, una o varias de las plazas asignadas sean incompatibles con las discapacidades de los jubilados. La calidad de la solución inicial (en caso de ser solución) es media, ya que esta solución inicial se genera de forma equivalente a un greedy que intenta minimizar C y D y maximizar A (intenta asignar el mayor número de jubilados a plazas, que es F), pero no tiene en cuenta el criterio B (número de jubilados con preferencias satisfechas), por lo que tiene margen de mejora. El coste de generación de la solución inicial, asumiendo que ordenamos primero los jubilados por su frecuencia de viaje y los hoteles por su precio por plaza, es $O(J \times logJ) + O(H \times logH) + F$, ya que una vez ordenados los jubilados y los hoteles solo hemos de recorrer la lista de plazas de cada hotel e irles asignando un jubilado. Aunque se ha de decir que, como la solución inicial está asignando todas las plazas de todos los hoteles, no tiene mucho sentido ordenar las plazas por su precio, ya que al final las asignaremos todas (baratas y caras).

   El operador propuesto comprueba que la *plaza* cumpla las posibles compatibilidades del *jubilado*2 (por lo que sirve para ir reduciendo el número de asignaciones incompatibles en la solución parcial), pero no comprueba sus preferencias o la frecuencia de sus viajes (eso no es un problema si se valora/penaliza adecuadamente en la función de evaluación). El factor de ramificación del operador en el peor caso es $O(A \times (J - A))$. El operador mantiene en todo momento el número de plazas asignadas, y eso solo nos permetirá resolver el problema en los casos en los que exista una asignación de F jubilados a las F plazas que cumpla todas las compatibilidades (en esos casos partimos de una asignación de todas las F plazas con algunas incompatibilidades, y con los intercambios se va incrementando el número de asignaciones compatibles con las discapacidades). Pero en el caso de que no haya asignación posible que nos permita asignar todas las plazas (por ejemplo, porque hay muchos jubilados con discapacidades y hay muchas plazas que no estan adaptadas a esas capacidades), como se parte de una solución inicial donde todas las plazas estan asignadas el operador no puede generar ninguna

de las soluciones en las que se asignan solo un subconjunto de plazas. Es por todo ello que podemos decir que el operador no cubre todo el espacio de soluciones que nos interesan en algunos casos, y sería necesario añadir al menos un operador para `eliminar` una asignación de jubilado a plaza.

La función heurística propuesta suma todos los criterios, pero el problema es que algunos nos piden maximizarlos (A y B) y otros nos piden minimizarlos (C y D), por lo tanto, aunque Hill Climbing es una técnica que puede maximizar o minimizar una función de evaluación, en este caso ni minimizando ni maximizando la función propuesta podemos obtener el tipo de soluciones que queremos. Dado que cada uno de estos criterios está en diferentes unidades, deberiamos convertirlo en una suma y resta ponderada. Además este heurístico debería incluir una penalización de las no soluciones que permita distinguir entre una no-solución mala (con varias asignaciones incompatibles) y una mejor (con menos asignaciones incompatibles), de forma que pueda guiar la búsqueda por el espacio de no-soluciones dirigiéndola hacia el espacio de soluciones. Por ejemplo, sea $G = \sharp$ jubilados asignados a plazas incompatibles con sus discapacidades, podriamos redefinir el heurístico (a maximizar) como $h'(n) = p * (A + B) - q * C - r * D - s * G$, siendo p, q y r ponderaciones que reescalen la influencia de los criterios A, B, C y D segun los rangos de posibles valores, y siendo s una ponderación que tendrá un valor muy superior a p, q o r, de forma que el número de asignaciones incompatibles siempre gane al resto de criterios. Aunque cabe destacar que, con la solución inicial y el operador propuesto, el valor de A es siempre constante ($A = F$) y el valor de D también es constante ($D = E$).

b) Se plantea utilizar algoritmos genéticos. Asignamos a cada plaza de hotel un número de 0 a F, y representamos una solución como una secuencia de $J \cdot log_2(F + 1)$ bits, representando para cada jubilado j la plaza de hotel asignada, o el valor 0 si no tiene plaza asignada. Para generar la población inicial obtenemos una solución asignando aleatoriamente un jubilado a cada plaza (solo F jubilados tienen un número diferente de 0). Como operadores genéticos usamos los operadores habituales de cruce y mutación. La función heurística es:

$$h'(n) = \frac{A + B}{\frac{C}{10 * J} + \frac{D}{E}}$$

Plantear como solución un algoritmo genético para este problema es adecuado a priori, ya que nos piden encontrar una solución maximizando o minimizando una serie de criterios sin necesidad de obtener el óptimo.

La codificación de las soluciones es una representación a priori correcta del problema: un vector que representa el identificador de plaza que se asigna a cada jubilado, con un número extra para representar la asignación nula (*NOTA: hay un pequeño error en el enunciado: a cada plaza se le asigna un número de* 1 *a* F*, dejando el valor* 0 *para la asignación nula.*). Tal y como dice el enunciado esta representación requiere $J \cdot log_2(F + 1)$ bits, pero si $J \gg H$ es de esperar que $J > F$ (el numero de plazas totales dependerá del número de plazas que aporta cada hotel, suponiendo que m es la media de plazas por hotel, $F = m * H$), y eso significa que probablemente tenemos muchos jubilados con asignación nula. La representación escogida permite representar cualquier solución del problema pero también muchas no-soluciones: identificadores de plaza que no existen (si $F + 1$ no es potencia de 2, los $log_2(F + 1)$ permiten representar numeros mayores que F) , plazas asignadas a más de un jubilado, plazas asignadas a jubilados incompatibles con ellas por su discapacidad. Una alternativa mejor seria tener una secuencia de $F \cdot log_2(J + 1)$ bits, representando para cada plaza f el identificador del jubilado asignado, o el valor 0 si no tiene jubilado asignado (asignación nula). Esta representación requiere menos bits y tendrá muchas menos asignaciones nulas, pero también será capaz de representar no soluciones (identificadores de jubilado que no existen, jubilados asignados a más de una plaza, plazas asignadas a jubilados incompatibles con ellas por su discapacidad). Por lo tanto en ambos casos tenemos el potencial de entrar en el espacio de no-soluciones por los mismos motivos.

La estrategia para generar la población obtiene diferentes soluciones iniciales, esto es correcto. Pero las soluciones iniciales pueden no cumplir la restricción de que todas las asignaciones de plazas sean compatibles con las discapacidades de los jubilados, por lo que puede generar no-soluciones. El coste de generación es $O(P \times J)$ en el caso peor de no tener los vectores pre-inicializados a 0 (si están pre-inicalizados lo podemos reducir a $O(P \times F)$), siendo P el tamaño escogido de la población de soluciones iniciales.

Los operadores habituales pueden generar soluciones pero también todos los tipos de no-solución que permite la representación: identificadores de plaza que no existen, plazas asignadas a más de un jubilado, plazas asignadas a jubilados incompatibles con ellas por su discapacidad. En este caso el operador de mutación genera soluciones muy próximas a las originales (solo cambia el identificador de plaza asignada a un jubilado en un bit, si ya estan todas las plazas asignadas lo más probable es que asigne el jubilado a una plaza ya asignada -compatible o incompatible- o a una plaza inexistente) y es el operador de cruce el que genera sucesores algo más diferentes (según sea la distribución de las asignaciones a izquerda y derecha del punto de cruce en las soluciones padres, las soluciones generadas después del cruce tendrán más o menos jubilados con asignaciones no nulas, probablemente habrá una o varias plazas asignadas a más de un jubilado, y si el punto de cruce se hace en medio de la codificación de un identificador de plaza, puede generar también plazas inválidas).

Los algoritmos genéticos interpretan la función de evaluación como una funcion de $fitness$ de la solución al problema, por lo que normalmente suelen maximizar dicha función (buscar soluciones de mayor fit al problema). Si maximizamos la función heurística planteada valorará mejor las soluciones que queremos, ya

> que los criterios a maximizar estan en el numerador y los criterios a minimizar están en el denominador. El problema es que en el numerador tenemos cantidades de personas que se dividen por una suma ponderada de frecuencias de viajes y coste total, y este cociente nos puede dar valores similares con un gran número de jubilados asignados que hayan viajado mucho en el pasado, y con un numero muy pequeño de jubilados asignados que hayan viajado muy poco en el pasado, por lo que no nos distingue bien entre soluciones diferentes, ya que en realidad está intentando maximizar un ratio que no tiene mucho sentido. Sería mucho mejor (y más fácil de controlar) la suma-resta ponderada que hemos planteado en la respuesta del apartado anterior. Y al igual que antes, este heurístico debería incluir una penalización de las no soluciones, ya que podemos empezar en el espacio de no-soluciones, y los operadores también permiten generar no-soluciones.

Las notas se publicarán el día **10 de mayo**.