10. Para hacer presión ante RENFE, los sindicatos de conductores de Cercanías en Barcelona quieren encontrar la forma de cumplir los requisitos mínimos que se les imponen sobre número de trenes, pero afectando al mayor número de usuarios, para que sus reclamaciones les den a ellos mayor poder en las negociaciones. Para ello han construido una tabla que, para cada hora y línea de cercanías, les dice el número de usuarios que viajan a esa hora por esa línea (es decir, la demanda), y el número mínimo de trenes que según RENFE han de pasar durante esa hora por esa línea para cumplir los servicios mínimos. Sabemos también que cada tren puede llevar como máximo P pasajeros. Hay además otras reglas que han de cumplir, y es que RENFE impone un número mínimo L_i total de trenes que han de circular por la linea i cada día, y un número mínimo H_h total de trenes que han de circular en una hora h, siendo estos números algo mayores que la suma de los trenes por lineas o por horas de los servicios mínimos antes mencionados.

    Se nos plantean las siguientes alternativas:

    a) Queremos utilizar satisfacción de restricciones donde tenemos una variable por cada línea de cercanías y cada hora, y los valores son el número de trenes asignados a cada línea y cada hora. Las restricciones son el número mínimo de trenes para cada línea y hora, el número mínimo de trenes que han de circular para cada línea durante el día, el número mínimo de trenes que han de circular para cada hora y que el número total de usuarios que se queden sin tren sea mayor que un cierto valor U.

    A priori, usar un algoritmo de satisfacción de restricciones para este problema parece no ser buena idea, ya que el algoritmo tiene una complejidad temporal muy elevada y el problema no parece requerir de un óptimo global. Para intentar optimizar, el enunciado plantea añadir la restricción sobre un mínimo impacto en el número total de usuarios que se queden sin tren ($> U$). Esto sería mejor ejecutarlo en un optimizador de restricciones: en lugar de codificar esta restricción explícita, crearemos una variable modelando el número de usuarios afectados, y dejando al algoritmo incluir propagadores incrementando el valor mínimo de esta variable hasta que se deje de obtener solución. Empezando con valores bajos de U podemos asegurar el obtener soluciones iniciales. Incrementando paulatinamente el valor de U, si cogemos la última solución propuesta antes de llegar a un conjunto de restricciones no satisfactible, significa que hemos encontrado la solución que afecta al máximo de usuarios posibles entre las exploradas. Este planteamiento nos daría soluciones subóptimas (pero incrementalmente mejores), que podría argumentarse como una solución al problema inicial de complejidad.
    La elección de variables y sus dominios es correcta. El problema pide asignar un conjunto de trenes a unas líneas en unas horas, y lo que se han escogido como variables son los pares línea-hora. Cada variable tiene como dominio el número de trenes asignados a esa línea en esa hora. Por lo tanto, si el algoritmo nos devuelve una asignación de un valor por variable cumpliendo las restricciones, estaría representando una asignación de trenes a lineas y horas correcta, que es lo que buscamos. En la representación no se ha incluido el número de usuarios que viajan cada hora, pero no hace falta: como lo que queremos es poner menos trenes de los necesarios para cubrir la demanda de pasajeros de cada hora, se puede asumir que los pocos trenes que pongamos irán siempre llenos, llevando a P personas cada uno.
    Asumiendo que las restricciones indicadas son aplicables, la salida del algoritmo (para cada línea-hora, número de trenes) nos permite calcular el número mínimo de pasajeros afectados multiplicando el número total de trenes por P.
    Sin embargo, el modelado sobre el número de usuarios afectados requeriría de una variable adicional (como se ha mencionado anteriormente) que actualmente no está representada: el número de usuarios afectados. Una primera aproximación sería hacer una restricción n-aria sobre todas las variables, para asignar como valor a esta variable el número de usuarios afectados, mirando el número de trenes que pasan por cada lugar. Sin embargo, esto es difícil de calcular, y también difícil de propagar: requeriríamos una asignación completa para saber si hemos superado el valor U. Una solución mejor sería incluir las variables mencionadas con anterioridad (número de usuarios afectados por linea-hora), que es fácilmente calculable a partir de cada variable trenes por linea-hora (L*H restricción binaria afectados linea-hora con trenes linea-hora), y utilizando estas variables para definir el total. El propagador de restricciones será capaz de, mediante mirar los valores asignados y el valor máximo de las variables todavía por asignar, saber si la solución

parcial hasta ahora nos permite llegar a un valor superior a U en su suma, rápidamente podando el espacio de búsqueda.
Las tres primeras restricciones que se indican salen directamente del enunciado: R1) número mínimo de trenes para cada línea y hora, R2) número mínimo de trenes que han de circular para cada línea durante el día y R3) número mínimo de trenes que han de circular para cada hora.
R1 no es ni una restricción binaria ni n-aria, se aplicaría directamente sobre la propia variable, eliminando posibles valores en la primera pasada del propagador.
R2 es una restricción n-aria abarcando todas las variables, pero lo que tendria sentido es convertirla en un conjunto de L restricciones H-arias (tendríamos una restricción por cada conjunto de variables que corresponden a diferentes horas de la misma linea). Como cada una se esas restricciones es un sumatorio (de trenes asignados a una línea durante el día) seria imposible convertirlas en un conjunto de restricciones binarias.
El caso de R3 es similar: se propone una restricción n-aria abarcando todas las variables, pero lo que tendria sentido es convertirla en un conjunto de H restricciones L-arias (tendriamos una restricción por cada conjunto de variables que corresponden a diferentes lineas en la misma hora). Pero de nuevo, al ser cada una de esas restricciones un sumatorio (de trenes asignados en una hora determinada en todas las lineas), es imposible convertirlas en un conjunto de restricciones binarias.

b) Queremos utilizar búsqueda local, donde se genera una solución inicial colocando suficientes trenes para cubrir los servicios mínimos de cada línea y hora, y asignando aleatoriamente los trenes restantes entre todas las líneas. Los operadores de modificación de la solución consisten en mover un tren de una hora a otra en la misma línea, y mover un tren de una línea a otra. Queremos maximizar el número de usuarios que se verán afectados por la falta de trenes.

A priori un algoritmo de búsqueda local (como Hill Climbing) es adecuado para este problema, en el que se nos pide maximizar y minimizar criterios (maimizar usuario afectados, minimizar trenes).
El planteamiento es correcto ya que las soluciones nos permiten calcular el número de usuarios afectados (si asumimos que los trenes transportan P pasajeros) y el heurístico planteado coincide con el objetivo del problema: queremos maximizar el número de personas afectadas.
La solución inicial asegura que se cumplen los trenes mínimos necesarios en una linea y hora en concreto, pero al asignar el resto de trenes al azar es posible que no se cumplan los minimos para una hora en todas las lineas o para una linea en todas las horas, y por lo tanto puede ser no-solución. Y en el caso de ser solución, al incluir todos los trenes disponibles su calidad es baja, ya es posible que no se necesiten todos los trenes disponibles para cubrir los servicios mínimos. Por ello se ha de modificar el planteamiento de la solución en una de las dos formas siguientes:

- modificar la generación de la solución inicial para que coloque suficientes trenes para cubrir los servicios mínimos de cada línea y hora, y para aquellas líneas en las que aun no se cumple el número mínimo de trenes por día o para aquellas horas en las que faltan trenes para llegar al mínimo de trenes/hora, asignar aleatoriamente los trenes que faltan entre las horas y lineas.
- no modificar la generación de la solución inicial, pero entonces añadir un nuevo operador que elimina un tren de una hora y día, para permitir al algoritmo eliminar los trenes sobrantes de la solución. En principio no haria falta otro operador para añadir trenes, ya que en este caso se empieza siempre con un exceso de trenes.

Los dos operadores planteados son correctos pero no comprueban ninguna de las restricciones, y podrian llevarnos a estados que no son solución.
La función de evaluación está ligada directamente al objetivo del problema (maximizar usuarios afectados) pero falta que penalice las no-soluciones que pueden aparecer tanto en la generación de la solución inicial como en la aplicación de los operadores.

Comenta cada una de las posibilidades indicando si resuelven o no el problema y qué ventajas e inconvenientes tiene cada una de ellas. Justifica la respuesta.

19. Tras el proceso de reestructuración bancaria el potente Banco Nacional de Crédito (BNC) ha comprado la Caja de Ahorros Popular (CAP) con todas sus oficinas. El problema es que, tras la fusión de ambas entidades, hay un exceso de oficinas bancarias en una de las ciudades donde ambas entidades operaban antes de la fusión. Nos piden un sistema inteligente que les ayude a decidir que oficinas cerrar. Tenemos un mapa de la ciudad con las O oficinas disponibles tras la fusión. Para cada oficina o_i podemos obtener el coste anual de su alquiler ($alquiler(o_i)$). Tenemos también un listado de todos los C clientes que tenemos tras la fusión del BNC y la CAP. Para cada cliente c_j tenemos su dirección (geo-localizada en el mapa) y podemos obtener el importe total de los depósitos que tiene en la entidad ($depositos(c_j)$).

    El banco le da mucha importancia a tener oficinas cerca de sus clientes. Por sus estudios de mercado han visto que sus mejores clientes suelen estar en un radio de 600 metros de sus oficinas. Nos dan una función `Oficina oficina_cercana(Cliente c)` que dado un cliente `c` usa el mapa de la ciudad para devolvernos la oficina `o` más cercana, o devuelve `null` si no hay ninguna oficina a menos de 600 metros del cliente. También tienen una función `float beneficio_deposito(Cliente c)` que dado un cliente `c` hace una previsión de los beneficios que obtendrá el banco durante un año operando en los mercados financieros con el dinero de los depósitos de ese cliente. A partir de estas funciones definen dos más: la primera es `float estim_benef_depositos(Oficina o)` que calcula la suma de beneficios obtenidos por los depósitos de todos los clientes cuya oficina más cercana es `o`; la segunda es `LClientes clientes_sin_oficina()` que nos devuelve la lista de clientes que no tienen ninguna oficina a menos de 600 metros de donde viven. Todas estas funciones se pueden invocar varias veces durante la ejecución y nos darán diferentes resultados dependiendo de las oficinas que se quieran cerrar y de cual quede más cerca de cada cliente en cada momento.

    El objetivo es reducir el número de oficinas que se mantienen abiertas tras la fusión de ambas entidades, minimizando el coste total anual en alquileres de las oficinas que se mantienen abiertas y maximizando el benefico total obtenido a partir de los depósitos de los clientes que tienen una oficina a menos de 600 metros de su vivienda. La idea es cerrar oficinas que no salen a cuenta mantener porque no tienen suficientes clientes para cubrir gastos con los beneficios obtenidos, o porque hay otra oficina en la cercania que puede atender a esos clientes. Aunque no gusta perder clientes, es posible dejar algunos clientes sin oficina cercana si mantener dicha oficina genera más costes que beneficios.

    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 respecto a si es correcta, es eficiente, o es mejor o peor respecto a otras alternativas posibles. Justifica tu respuesta.

    a) Se plantea solucionarlo mediante Hill-Climbing. Como solución inicial se parte del conjunto completo de todas las O oficinas disponibles y de los C clientes. Como operador único se plantea el eliminar una oficina de la solución. Como función heurística se usa

    $$h_1 = \sum_{\forall o_i \in OS} estim\_benef\_depositos(o_i) - alquiler(o_i)$$

    siendo OS el conjunto de oficinas supervivientes (no eliminadas).

    A priori Hill Climbing es un algoritmo adecuado para este problema, ya que buscamos una solución que maximice los beneficios y minimice los gastos.

    Tal y como se plantea el problema cualquier selección de un subconjunto de oficinas supervivientes es solución, por lo que estaremos trabajando siempre dentro del espacio de soluciones. Esto implica que no hace falta comprobar si nos salimos del espacio de soluciones ni en los operadores ni en la función de evaluación.

    La solución inicial es válida (aunque su calidad es mala). Su mayor ventaja es que el coste de generación es mínimo, y deja margen del maniobra al algoritmo de búsqueda.

    El heurístico es correcto, si optamos por maximizarlo conseguiremos ambos objetivos (maximizar el benefico total obtenido y minimizar el coste total en alquileres de las oficinas abiertas). No hace falta añadir el número de oficinas abiertas al heurístico, ya que al reducir el coste de los alquileres estamos reduciendo el número de oficinas abiertas. Dado que tenemos en este heurístico dos criterios con las mismas unidades (Euros) y que en el enunciado no se establece ninguna prioridad sobre ellos, restarlos sin ponderación es una buena opción.

Como se parte de una solución inicial con todas las oficinas, el operador que elimina oficinas es correcto, tiene un factor de ramificación lineal respecto al número de oficinas y nos permitiría en teoria explorar todo el espacio de búsqueda. Con el heurístico planteado, a cada paso del Hill Climbing se generan OS soluciones sucesoras, en cada una de ellas se ha eliminado una oficina. Se evalua el heurístico y se escoge la mejor solución, que será en la que se elimina una oficina que genera pérdidas o donde la mayoria de sus clientes tienen otra oficina a menos de 600 metros y quedarán cubiertos, por lo que el banco mantiene sus beneficios y se ahorra el alquiler de la oficina eliminada.

Aunque el heurístico calcula exactamente lo que necesitamos, no se puede asegurar el óptimo debido a la naturaleza compleja del problema: cada vez que eliminamos una oficina se redistribuyen los clientes entre las oficinas supervivientes (y, a veces, perdemos algunos). Esto hace que en un momento puntual una oficina puede ser poco interesente por no tener suficientes clientes que aporten beneficios, pero pueda pasar a serlo si eliminamos sus oficinas vecinas. Es posible que si añadieramos un operador intercambiar donde se substituya una oficina en la solución por otra previamente eliminada se puedan hacer pequeñas mejoras en la solución obtenida, pero no está claro que el pequeño incremento en la calidad de las soluciones obtenidas compense el factor de ramificación extra de este segundo operador (cuadrático respecto al número de oficinas). Tampoco podemos mejorar los resultados del Hill Climbing ejecutándolo varias veces, ya que siempre empieza en el mismo estado inicial y nos daría la misma solución en cada ejecución.

b) Se plantea solucionarlo mediante A*. Para ello se parte del estado en el que no hay ninguna oficina en la ciudad. Tenemos dos operadores: colocar oficina, que añade una oficina $o_i \in O$, su coste es $estim\_benef\_depositos(o_i) - alquiler(o_i)$; eliminar oficina, que elimina una oficina o_i que hubiera sido colocada con el operador anterior, su coste es $-(estim\_benef\_depositos(o_i) - alquiler(o_i))$. Como heurístico h se usa:

$$h_2 = \sum_{\forall c_j \in CN} beneficio\_deposito(c_j)$$

siendo CN el conjunto de clientes que quedan por servir (los que no tienen ninguna oficina cercana), que se obtiene con la función `clientes_sin_oficina()`.

El planteamiento de usar un A* a priori puede parecer correcto para este problema. En este caso se parte de un estado inicial con solución vacia y se va construyendo una solución añadiendo oficinas de forma que se optimice un criterio (maximizar beneficios y minimizar costes). Pero en este problema no nos están pidiendo que obtengamos el óptimo de todo el espacio de estados, y por lo tanto no es adecuado usar A*, ya que estamos añadiendo una complejidad innecesaria en la resolución de este problema.

El operador para añadir oficinas es correcto (si tenemos el heurístico adecuado), ya que nos permite ir construyendo una solución. Su factor de ramificación es líneal respecto al número de oficinas por colocar en la solución. Pero su función de coste no es correcta, tal y como veremos luego. El operador para eliminar oficinas es incorrecto. En A* no necesitamos operadores para deshacer el efecto de otros operadores, ya que el algoritmo es capaz de 'deshacer' dicho efecto escogiendo otra rama en el arbol de búsqueda.

Pero los problemas más graves están en las funciones que han de guiar al algoritmo.

En primer lugar, el coste del operador añadir no es una función de coste correcta debido a que es una resta que no garantiza resultado positivo (cuando el alquiler de una oficina supera los beneficios obtenidos gracias a los clientes). A* no es un algoritmo que pueda trabajar con costes negativos (no es posible garantizar optimalidad). Una posible alternativa, que consistiría en calcular el coste de la oficina restándole los beneficios obtenidos en ella ($alquiler(o_i) - estim\_benef\_depositos(o_i)$), seguiría teniendo el mismo problema de resultar en costes negativos ya que no podemos asumir por la información que tenemos que los beneficios no superen al alquiler en oficinas concretas.

Otro problema de la función de coste con el operador añadir tal como está formulado es que, al colocar nuevas oficinas, puede haber clientes que estaban servidos por oficinas en la solución que

pasen a tener la nueva oficina más cerca de sus casas, y para no contarlos dos veces deberíamos de recalcular la función $estim\_benef\_depositos(o_i)$ para las oficinas previamente colocadas y así recalcular la función G ya calculada para varios nodos. Que el coste de atravesar un arco vaya cambiando durante la ejecución del algoritmo es algo para lo cual el algoritmo A* no está preparado.

En lo que respecta al heurístico h_2, este sólo puede llegar a ser cero cuando todos los clientes están servidos por una oficina. El enunciado remarca que es perfectamente posible dejar clientes sin oficina si eso implica aumentar el beneficio neto total. Por lo tanto, la solución óptima puede pasar por dejar clientes sin servir. A* impone que el heurístico en estados finales sea igual a 0. Al no cumplir h_2 con esta condición, es posible pasar por estados que según la definición del problema deberían ser solución pero cuyo valor heurístico sea estrictamente positivo. En estos casos, h_2 nos dice que aún no hemos llegado a la solución, lo cual hace que este heurístico no sea admisible y por lo tanto no garantizaría la optimalidad.

Por otro lado, A* tiene como objetivo minimizar la función $F = G + H$. Pero G y H están calculando beneficios obtenidos por las oficinas, que es un criterio a maximizar. Además G (la suma de costes del operador añadir) acumula la suma de los beneficios obtenidos gracias a los clientes servidos por las oficinas colocadas, mientras que H (en este caso h_2) acumula la suma de los beneficios que se pueden obtener gracias a los clientes aun no servidos por ninguna oficina. Al sumar beneficios de clientes servidos y no servidos se obtiene siempre la misma cifra constante (los beneficios que se obtienen gracias a los depósitos de todos los clientes), por lo que en realidad A* está intentando minimizar lo que queda, $-alquiler(o_i)$. El algoritmo estaría siendo guiado por un criterio diferente del exigido (maximizar beneficios y minimizar alquileres).

Es difícil encontrar una variación del modelo de este problema que haga que el algoritmo A* funcione correctamente, con funciones G donde el coste de los arcos no cambie dinámicamente durante la ejecución y heurísticos H admisibles, por lo que en realidad no es una buena opción.

c) Se plantea resolverlo mediante algoritmos genéticos. Para representar el problema utilizamos una tira de O bits, donde un bit a 1 indica que esa oficina se va a mantener abierta y un bit a 0 indica que esa oficina se va a cerrar. Como población inicial generamos aleatoriamente n individuos donde en cada uno hay $O/2$ bits a 1. Como operadores usamos un operador de cruce en un punto y un operador de mutación que cambia el valor de un bit de la cadena (de 0 a 1 o de 1 a 0).La función heurística es:

$$h_3 = \sum_{\forall o_i \in OS} \frac{estim\_benef\_depositos(o_i)}{alquiler(o_i)}$$

siendo OS el conjunto de oficinas supervivientes (no eliminadas), es decir, las que tienen el bit a 1.

Los algoritmos genéticos son una elección adecuada para este problema, ya que nos permiten escoger soluciones que maximicen o minimicen criterios.

La representación es correcta y permite representar todas las soluciones: cada posición representa la inclusión o no de una oficina en la solución. Además en este caso, como cualquier subconjunto de oficinas escogidas es solución, esta representación nunca representa no-soluciones, por lo que no tendremos que controlar el salirnos del espacio de soluciones ni durante la generación de las soluciones iniciales ni en los operadores genéticos.

La generación de la población inicial es correcta, ya que al tener una cierta aletoriedad puede generar soluciones iniciales diferentes. El optar por generar siempre soluciones iniciales con exactamente la mitad de las oficinas abiertas es algo arbitraria, pero no podemos decir a priori cual será la calidad de las soluciones iniciales ya que dependerá de la distribución de los clientes en el mapa y de cuantos quedan cubiertos al colocar solo la mitad de las oficinas. El coste de generación es de $O(n \times O/2)$ (colocar n veces $O/2$ bits a 1).

La función que se usa para evaluar la calidad de las opciones divide beneficios estimados de cada oficina abierta entre el coste del alquiler de dicha oficina. En este caso nos interesa maximizar esta

función. Pero usar en este caso una división en vez de una resta no es una buena idea, ya que la fracción puede dar valores similares con diferentes números en el numerador y en el denominador. Podría darse el caso en el que el valor máximo de la función se obtenga con unas pocas oficinas con un ratio Beneficio/Coste muy alto (pero beneficios medios o bajos). Tampoco penaliza en exceso las oficinas con pérdidas (aportan un valor entre 0 y 1 al sumatorio) respecto a las que aportan ganancias (aportan un valor mayor que 1).

El operador de mutación apenas modifica el valor de uno de los bits de la solución, lo que es equivalente a añadir o eliminar una oficina en dicha solución. Sirve para explorar soluciones cercanas a las que ya se tienen. En cambio el operador de cruce en un punto genera dos soluciones futuras que pueden divergir bastante de sus padres (por ejemplo, si una de las soluciones padre tiene la mayoria de bits a 1 a la izquierda del punto de corte y la otra solución padre tiene la mayoría de bits a 1 a la derecha de ese punto, el resultado del cruce es una solución hija mucho más cargada de bits que la otra). Esta propiedad no es mala de por sí, pero si la probabilidad de cruce es alta puede ser que el algoritmo deshaga soluciones buenas en otras peores y no consiga converger hacia soluciones buenas.