Examen Parcial de IA

(26 de octubre de 2022)

Duración: 90 min

1. (6 puntos) El aumento del precio del cable de fíbra óptica en el mercado negro ha disparado los casos de robo de cableado de fibra óptica a gran escala, tanto de los cables que proporcionan conexión de internet a poblaciones como los cables que se usan en las vias del AVE para transmitir comunicaciones y señales de control. Esto ha disparado la demanda de cable de fibra óptica de diferentes longitudes, según lo grande del tramo robado. Una empresa se quiere especializar en servir de forma rápida y eficiente estos pedidos en el sur de Europa. Para ello, en vez de fabricar cables de fíbra óptica a medida, planea tener pre-fabricados R cables de una única longitud fija (l) que se empacan y almacenan en R rulos, y luego se sirven los pedidos cortando el cable del rulo en tramos de cable de la longitud solicitada (ningún tramo de cable solicitado será de mayor que l).

   Para reducir la cantidad y longitud de los tramos cortos de cable que no le sirven a nadie, nos han pedido hacer un sistema que, dado un conjunto de T pares (id_solicitud, longitud) obtenido al juntar todos los pedidos recibidos en un día, nos diga como hemos de cortar los cables de rulo pre-fabricados en tramos de cable de manera que desperdiciemos la menor cantidad de material.

   En los siguientes apartados se proponen diferentes alternativas para algunos de los elementos necesarios para plantear la búsqueda (solución inicial, operadores, función heurística,...). Para cada a una de ellas se ha de 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.

   a) Se plantea aplicar Hill-climbing, usando como solución inicial asignar (en orden creciente de longitud) tramos de cable solicitados a cables de rulo; cuando la suma de las longitudes de los tramos asignados a un cable superan su longitud se pasa al siguiente cable de rulo. Como operadores se usan `mover_un_tramo` solicitado de un cable de rulo a otro y `quitar_un_rulo` de cable que no tiene tramos asignados. Como función heurística se usa la longitud de cable desechado, que se calcula como la diferencia entre 1) la suma de las longitudes de los tramos solicitados y 2) la longitud total de los cables de rulo que forman parte de la solución (o lo que es lo mismo, l multiplicado por el número de rulos con tramos asignados).

   El planteamiento del problema como una búsqueda en el espacio de soluciones con un Hill Climbing es correcto, buscamos minimizar el desperdicio de material buscando una buena asignación de tramos de cable a rulos.

   La solución inicial se encuentra dentro del espacio de soluciones ya que tenemos todos los pedidos asignados a algún cable de rulo y se comprueba que nunca excedemos la longitud del cable de rulo (solo tendríamos una no-solución en el caso extremo de que la asignación secuencial de los tramos más largos uno tras el otro sin tramos pequeños en medio provocara un gasto excesivo de rulos por encima de R). El coste de la asignación es lineal respecto al número de solicitudes de tramos de cable ($O(T)$), pero hay que añadir el coste de ordenar antes las longitudes de los tramos de cable, un coste asumible ($O(T \times \log(T))$). La calidad de la solución inicial no es muy buena, ya que al asignar primero los tramos pequeños y luego asignar los grandes hará que se puedan colocar muchos menos tramos en los últimos rulos, y probablemente se desperdiciará bastante material de esos últimos rulos. Esto en el caso de Hill Climbing es bueno, porque una solución inicial demasiado buena bloquearía muy rápido la búsqueda en un máximo local. Una solución inicial de calidad similar y menor coste ($O(T)$) seria asignar directamente los tramos al rulo actual sin ordenar, pasando de un rulo al siguiente cuando el tramo no cabe.

   El operador de mover tal y como se plantea en el enunciado puede crear no-soluciones, debería comprobar antes que en el rulo destino cabe el tramo que vamos a mover para mantenernos en el espacio de soluciones. El factor de ramificación será $O(T \times R)$. El operador de quitar cables de rulo es correcto ya que comprueba antes si están vacíos, y su factor de ramificación será lineal respecto al número de cables de rulo ($O(R)$). A priori, la combinación de ambos operadores permite explorar todo el espacio de soluciones, ya que partimos de una solución inicial con todas las solicitudes asignadas a algún rulo que luego las podemos recolocar con el operador de mover, y partimos del

número máximo de cables de rulo necesarios para servir las solicitudes que luego vamos reduciendo con el operador de quitar. Pero sería una buena idea fusionar el operador de quitar cables de rulo dentro de operador de mover tramos (en el momento en el que se mueve el último tramo asignado a un rulo, se elimina) ya que mantenemos la misma cobertura del espacio de soluciones con un menor factor de ramificación. Otra mejora que se podría añadir es un operador de intercambio de dos tramos para poder afinar aun más la colocación en los casos de rulos que ya están muy llenos (pero para que funcionara se debería cambiar el heurístico, como se discutirá a continuación).

Respecto al heurístico, la longitud total de los tramos de cable solicitados es una constante, por lo que se puede obviar. Esto quiere decir que todo el algoritmo está guiado únicamente por la longitud de los cables de rulo en la solución (de hecho, como l es constante, está guiado solo por el número de rulos en la solución), y este heurístico solo cambia por aplicación del operador quitar cable de rulo. Si no se añade una función de aplicabilidad al operador mover para evitar salirnos del espacio de soluciones hemos de penalizar dichos estados no-solución en este heurístico. Pero existe un problema aun más grave: esta función heurística no nos daría ningún sucesor mejor al actual aplicando el operador mover por lo que este operador no se aplicará **nunca**, y sin mover tramos no podemos vaciar de solicitudes un rulo dado, por lo que nunca podremos aplicar tampoco el operador de quitar cables de rulo, haciendo que el Hill Climbing quedará encallado desde el inicio, realizando **cero pasos**. Una posibilidad sería usar alguna medida no lineal sobre como se distribuyen las longitudes sobrantes en los cables de rulo, en este caso buscamos que estas longitudes se encuentren en los extremos, o muy cortas o muy largas. Una función de entropía (que intente forzar a los rulos a estar completamente llenos de tramos o completamente vacios) podría funcionar bien.

b) Se plantea usar algoritmos genéticos. Para la representación de una solución se decide asociar primero cada cable de rulo a un número de 1 a R. La idea es asignar a cada tramo solicitado el número que identifica el cable de rulo que tiene asignado. Cada solución se codifica como una cadena de $T \times log_2 R$ bits, teniendo para cada tramo el identificador de rulo en binario. Como mecanismo para generar la población inicial se escoge al azar, para cada tramo solicitado, un número en binario dentro de los valores $[1, R]$ codificando el cable de rulo asignado al tramo. Como operadores genéticos se usan los operadores de cruce y mutación habituales. La función heurística será el número de rulos a cortar distintos que hay en la codificación de la solución.

El planteamiento del problema como una búsqueda en el espacio de soluciones a través de un algoritmo genético es correcto, buscamos la solución que minimice el desperdicio de material. Sin embargo, dado su coste temporal mayor (con respecto a un Hill Climbing), solo sería adecuado en casos donde es difícil encontrar un buen heurístico para el Hill Climbing (no es el caso para este problema, como hemos visto en el apartado a).

La codificación permite representar todas las posibles soluciones (asignaciones de tramos solicitados a rulos de corte), pero también es capaz de representar dos tipos de no-soluciones: identificadores de rulo inválidos (si R no es exactamente una potencia de 2) y overflow de tramos asignados en un rulo (la suma de las longitudes de los tramos asignados es mayor que la longitud del cable de rulo), por lo que fácilmente nos saldremos del espacio de soluciones. girar la representación en una cadena de $R \times log_2 T$ bits para hacerla más compacta ($T > R$) no solo no resuelve ninguno de esos problemas sino que provoca otro más (cada rulo solo puede tener asociado el identificador de un único tramo).

La generación de soluciones iniciales se ha planteado correctamente al tener una aleatoriedad que es capaz de crear numerosos individuos para la población inicial de soluciones, pero al no comprobar si se produce overflow en los tramos asignados a un cierto rulo es muy probable que genere no soluciones. El coste de generación es $O(T \times NI)$, siendo T el número de peticiones y NI el número de individuos de cada generación del algoritmo. La calidad de las soluciones (cuando son solución) es probablemente mala, ya que al asignar al azar tramos a rulos es muy probable que los tramos queden distribuidos por un gran número de rulos, lo que va en contra del objetivo de minimizar el número de rulos usados. Una alternativa mejor para generar variabilidad de soluciones iniciales seria el ir rellenando rulos de uno en uno, de forma secuencial, como en el apartado a, pero escogiendo al azar el tramo a colocar. De esta forma tendríamos soluciones iniciales diferentes de calidad media-alta con un coste de generación lineal respecto al número de tramos ($O(T)$).

Los operadores de cruce y mutación estándar, al no poder comprobar ninguna restricción, muchas

veces convertirán una solución en una no solución:

- El primer problema es que ambos operadores pueden generar overflow en uno o varios rulos de corte (la suma de las longitudes de los tramos asignados es mayor que la longitud del cable de rulo);
- El segundo problema es que ambos operadores pueden generar un identificador de rulo inválido. Este segundo problema se podría evitar en el operador de cruce si restringimos su aplicación a los puntos entre un identificador de rulo y el siguiente, de forma que no se parta por enmedio la cadena de bits que codifica un identificador.

La función heurística se basa en un criterio (número de rulos de corte utilizados) que queremos minimizar (ya que existe una relación directa entre rulos de corte usados y longitud de cable desechado: partiendo de la base de que hemos de asignar todos los tramos solicitados, si somos capaces de servir la misma suma de longitudes de tramos solicitados con menos rulos, significa que estamos dejando menos longitud de esos rulos sin usar). Pero el heurístico no penaliza soluciones que están fuera del espacio de soluciones: debería penalizar el $overflow$ (asignaciones de tramos por encima de la longitud del rulo) en la asignación de tramos a rulos (a mayor overflow, peor solución), pero dando peso sobretodo a la existencia de overflows (un overflow en uno solo de los rulos ya es un problema). También se debe penalizar por cada asignación de un tramo a un identificador de rulo inválido. Otro problema importante es que en los algoritmos genéticos el heurístico se interpreta siempre como una función de fitness que se ha de maximizar, por lo que deberíamos de girar la función para que al maximizarla nos lleve a las soluciones que queremos (por ejemplo, cambiar el número de tramos usados por el número de tramos por usar, cambiar el número de identificadores de tramo inválidos por el número de identificadores válidos, etc.). De hecho en este problema sería aun mejor substituir el número de tramos por usar por la función de entropía como la descrita en el apartado a pero al revés (1 - entropía), invertir las penalizaciones descritas y maximizar la función resultante.

c) Se plantea usar un algoritmo de programación de restricciones. Las variables son los tramos de cable de los pedidos, los dominios son un número natural (de 1 a R) que identifica el rulo del que se cortará el tramo de cable del pedido. Como restricciones se propone una restricción n-aria que abarca todas las variables que comprueba que, para cada rulo, la suma de longitudes de los tramos de cable asignados al rulo no exceda l, y otra restricción n-aria para comprobar que la suma de longitudes de cable desechado (no asignado a ningún pedido) sea menor que una longitud D.

A priori, un algoritmo de programación de restricciones parece adecuado, ya que necesitamos encontrar una solución dentro de un espacio con muchas no soluciones (asignaciones que provocan overflow de tramos asignados a un rulo). Un problema de optimización con tantas asignaciones invalidas es bastante indicado para este algoritmo.

En la representación actual tenemos un total de T variables, una por tramo; con un dominio de R valores cada una (de 1 a R). Una asignación en este espacio donde reducimos el dominio de todas las variables a un solo valor (cumpliendo todas las restricciones) nos devuelve una solución válida (una asignación donde cada tramo tiene asociado un rulo y solo uno, que es lo que buscamos). No sería posible, por ejemplo, una representación de R variables (una por rulo) con un dominio de T valores (de 1 a T), porque cuando el algoritmo redujera los dominios de todas las variables a un valor forzaría que cada variable rulo tuviera un tramo (y solo uno) asignado, que no es lo que nos interesa. Para que esta representación girada funcionara deberíamos complicarla bastante más, (como veremos al final de este apartado al analizar otras representaciones).

Las restricciones descritas en la propuesta requieren de razonar sobre longitudes de tramos asignados a rulos y longitudes de cable de los rulos que quedan sin asignar. En la representación actual, la única manera de comprobar esto es sumar la longitud de los tramos asignados a cada rulo para comprobar que no hay overflow y calcular el remanente de cable de rulos utilizados en la solución que queda sin asignar. Ya que no sabemos donde se coloca cada rulo hasta su asignación, esto hace que las dos restricciones sean N-arias (en este caso $N = T$) y no se puedan reducir a R-arias o binarias. La primera restricción (sobre no superar la longitud l en cada rulo) guiará bastante bien el algoritmo ya que se pueden ir haciendo asignaciones de tramos a cada uno de los rulos desde el inicio del algoritmo y podar en el momento en el que no caben más tramos. En el caso de la segunda restricción (que el cable desechado sea menor que una longitud D) hay varios problemas. El primero

es que la longitud de cable desechado no es un parámetro estable: para un rulo r_i vale 0 cuando no tiene ningún tramo asignado, cuando se le asigna un tramo crece de repente hasta $l_r - l_t$ (siendo l_r la longitud del rulo y l_t la longitud del tramo recién asignado), y luego va decreciendo cada vez que se van asignando más tramos a ese rulo. Por ello, el sumatorio de longitudes de cable desechado es un parámetro inestable que va creciendo y decreciendo cuando se van asignado tramos. Como ya hemos visto, el cable desechado tiene una relación directa con el número de rulos usados, por lo que podemos usar directamente el número de rulos como parámetro de la restricción (es más estable y más fácil de calcular (convirtiendo a D en el número máximo de rulos a utilizar). El segundo problema es ver este valor D como un parámetro fijo: seria mejor convertirlo en una variable D que luego el algoritmo intentará optimizar (mediante un propagador de límite que va adaptando el valor de D durante la exploración, explorando valores de D menores). Las dos restricciones mencionadas en el enunciado son suficientes, no hace falta añadir una restricción más de que cada tramo se asigna siempre a algún rulo y solo a uno ya que eso queda asegurado por la elección de variables y dominios.

En resumen, el modelo propuesto para esta técnica obtiene una solución válida.

Existen al menos dos posibles variaciones de la representación que se podrian plantear, pero no son mucho más simples:

- la primera ya la mencionamos antes: girar las variables y los dominios, haciendo que las variables sean rulos y los dominios sean tramos. El problema es que para hacerla funcionar cada variable rulo debería tener asociados el conjunto de todos los subconjuntos posibles de tramos ($\{null, t_1, t_2, ..., t_T, \{t_1, t_2\}, \{t_1, t_3\}, ...\{t_1, t_2, t_3, ..., t_T\}\}$, este dominio tiene en total 2^T valores), de forma que al reducir todas las variables rulo a un único valor dentro del dominio nos permita rulos sin asignar, rulos con un solo tramo asignado, con dos, etc. En esta representación la primera restricción se aplicaría como un preprocesado del dominio de las variables rulo (eliminando todos los subconjuntos de tramos cuya suma de longitudes sea mayor que l, lo cual seguramente nos dará un número de subconjuntos muy menor que 2^T pero aun así será mayor que T (como mínimo cada tramo solo cabe en un rulo). La buena noticia es que calculado el conjunto de subconjuntos de tramos que caben en un rulo, como todos los rulos son de la misma longitud, podemos usar el mismo dominio para las R variables. La segunda restricción (ya la entendamos como una longitud máxima desechada o como número de rulos usados) seguirá siendo N-aria (en este caso $N = R$). Pero esta representación necesita de otra restricción R-aria más (que cada tramo aparezca solo en uno de los subconjuntos asignados a un solo rulo). Por lo tanto volvemos a tener 2 restricciones R-arias y con dominios más grandes que la propuesta original.
- la segunda sería muy diferente: tenemos $T \times R$ variables donde cada variable es un par $(tramo_i, rulo_j)$, y el dominio de valores se reduce a solo dos valores $\{0, l_{t_i}\}$ (siendo l_{t_i} la longitud del $tramo_i$ si ese tramo se asigna al $rulo_j$, en caso contrario el valor es 0). Supongamos que colocamos las $T \times R$ variables en una matriz $T \times R$ de forma que cada $tramo_i$ corresponde a una fila y cada $rulo_j$ corresponde a una columna. La primera restricción (asignación de tramos a cada rulo por debajo de la longitud l) se codificaria como R restricciones T-arias (una por columna, abarcando todas las variables que corresponden a un $rulo_j$) que serian fáciles de calcular (ir sumando 0 o l_{t_i} por cada tramo no asignado/asignado). La segunda restricción seria RxT-aria (abarcaría todas las variables) para comprobar que el número de rulos con longitud asignada >0 sea menor que D. Y nos haría falta una tercera restricción por cada fila de variables que compruebe que para un cierto tramo $tramo_i$ solo una de las variables tiene el valor l_{t_i} y el resto tienen 0. No esta claro que esta opción funcione mejor que la propuesta en el enunciado.

2. (4 puntos) La crisis militar en Ucrania ha provocado que miles de familias inmigrantes deban ser acogidas por los países de la Unión Europea. Como parte de esta, la Generalitat de Cataluña ya ha recibido una cantidad de familias de refugiados en sus comarcas, y deben tomarse medidas para asegurar su adaptación y bienestar. Partimos de que hay C comarcas en Cataluña, con I_c familias inmigrantes actualmente en cada comarca. Para asegurar su bienestar, el Govern de la Generalitat puede distribuir paquetes de ayuda en tamaños estandarizados de 100.000 o 500.000€, tantos como quiera por comarca. Por cada familia inmigrante, se establece un mínimo de A (ayuda) euros de presupuesto que debería obtener la comarca para asegurar su bienestar.

Para flexibilizar la situación, el Govern puede ayudar con la redistribución de las familias de inmigrantes, desplazando una cantidad fija (K) de familias de su comarca actual a otra, pero este desplazamiento deberá ir acompañado de una indemnización a cada família (con coste total de D euros), rebajando la cantidad de presupuesto en la comarca original por $K * A$ y subiendo el de la de destino por la misma cantidad. Por otro lado, la Generalitat sabe que cada envío de paquete a una comarca tiene un coste burocrático en personal y tiempo, así que busca minimizar el total de paquetes que se envían. Además, cada comarca tiene un Consell Comarcal que puede o no estar alineado políticamente con el Govern de la Generalitat, y en el reparto de dinero el partido en el poder preferiría que el dinero total de ayudas que van a comarcas no alineadas sea menor, ya que esto le dará más poder en las siguientes elecciones.

Queremos pues, encontrar el reparto de ayudas y, si es necesario, de desplazamientos a hacer, con tal de minimizar la cantidad de dinero invertido, minimizar la cantidad total de paquetes de ayudas enviados y minimizar la cantidad de presupuesto que va a comarcas bajo partidos políticos no afines al del Govern. Al Govern le gustaría que el resultado fuera el óptimo si es posible; pero si fuera imposible aceptará sencillamente que los criterios sean optimizados. Se nos proponen las siguientes soluciones:

a) Usar el algoritmo de A*. El estado inicial parte de ninguna ayuda asignada, y ningún desplazamiento de famílias hecho. Como operadores tenemos dos: el operador `mover_refugiados` de una comarca a otra, con condición de aplicabilidad que haya $\geq K$ refugiados en la comarca de origen y coste D; operador `añadir_paquete` que añade un paquete de 100K o de 500K a una comarca, con coste $coste(p)+\lambda+\epsilon*no\_afin(c)*coste(p)$, donde $coste(p)$ es el coste del paquete añadido (100K o 500K), λ y ϵ son constantes dadas que tienen que ver con el coste burocrático y político (y $no\_afin(c)$ vale 1 si la comarca no es afín y 0 si lo és); y el operador `quitar_paquete` que es el opuesto, con el mismo coste pero en negativo, y con condición de aplicabilidad que haya ≥ 1 ayuda del tipo escogido en la comarca. Como función heurística, usamos el número total de famílias todavía no cubiertas por las ayudas; en otras palabras, la suma de, para cada comarca c: $I_c - min(I_c, \lfloor \frac{suma\_ayudas(c)}{A} \rfloor)$, y donde entendemos que I_c ya tiene en cuenta los desplazamientos hechos con el operador `mover_refugiados`.

A* es a priori una buena opción para el problema, dado que se nos pide el óptimo global; siempre que haya un heurístico admisible (y no trivial) sería una buena opción. Los operadores propuestos tienen una ramificación de O(C*(C-1)), O(2*C) y O(2*C) respectivamente, y sus condiciones de aplicabilidad son correctas. Un detalle importante a notar es que el operador quitar es irrelevante para un algoritmo como A* que puede deshacer cambios, y además es un error conceptual grave tener un coste negativo en A*, así que este operador debe ser eliminado. Dado que C es un número bajo (no hay cientos de comarcas), la ramificación parece adecuada.

Una vez modificado el modelo quitando el operador quitar, debemos mirar que el coste g en una solución sea realmente lo que queremos optimizar, y en este caso lo es. Mirando ahora a la heurística h, nos damos cuenta que ya que en A*: $f = g + h$, g y h deberían tener las mismas unidades; y mientras que g está claramente en euros, h cuenta familias (lo cual es un indicio de algo malo), pero siempre que h sea admisible podríamos aceptarlo. Que h sea admisible es lo más importante para este problema. En este caso, siempre que A no sea un valor ridículamente pequeño (eg. menor que 1), parece que el coste de aplicar cualquier operador es muy superior a cualquier cambio que podamos hacer en h, así que intuitivamente sería admisible, si bien demasiado optimista. Sin embargo, podemos hacerlo mejor: si pasamos las unidades de h a contar euros (contando cuántos euros faltan en cada comarca): esto es análogo a multiplicar la heurística anterior por A. Tenemos la heurística $h = I_c * A - min(0, sum\_ayudas(c)) = max(0, I_c * A - sum\_ayudas(c))$. Vemos rápidamente que el coste de añadir un paquete en una comarca siempre es superior a la mejoría que podamos tener en la heurística, así que por ese lado es admisible. Por el otro lado, en el operador de `mover_refugiados`, la heurística puede mejorar un máximo de $K * A$ (si movemos K refugiados de una comarca que no puede mantener a ninguno a una en que mantiene a todos) con un coste de D. Para asegurar admisibilidad, deberíamos asegurar que $K * A \leq D$.

Como punto extra, y dado el análisis anterior, cabría mencionar que intuitivamente el heurístico parece ser, además de admisible, consistente; pero no pediríamos analizarlo durante un examen. La demostración es sencilla: para llegar a cualquier estado, independientemente de la ruta, hay que los mismos de operadores de añadir paquete por cada comarca y de mover (u otros mover análogos y con costes iguales). Esto hace que el coste de llegar a cualquier estado concreto siempre sea el mismo independientemente de en qué orden apliquemos operadores (o de si movemos familias de comarca A a B y C a D o si hacemos de A a D y de C a B). Esto hace que el coste g de un estado

es siempre el camino óptimo a él, condición suficiente para garantizar la desigualdad triangular de consistencia.

b) Usar el algoritmo de Hill Climbing. La representación de una solución incluye, para cada comarca, cuántas familias inmigrantes hemos desplazado, cuántas hay en cada comarca y cuántos paquetes de ayudas de cada tipo (100K, 500K) hay en cada comarca. Como solución inicial, partimos de una solución con 0 familias desplazadas, y en cada comarca hay el mínimo de ayudas de 100K necesarias para cubrir a todos los inmigrantes que hay inicialmente (es decir, $\lceil \frac{I_c * A}{100K} \rceil$). Como operadores tenemos `mover_refugiados`, con condición de aplicabilidad que haya $\geq K$ refugiados en la comarca de origen; `añadir_paquete` a una comarca, de 100K o de 500K; `quitar_paquete` de una comarca, con condición de aplicabilidad que haya ≥ 1 paquete de ese tipo en la comarca; `comprimir_paquete` que coge 5 paquetes de 100K y los sustituye por uno de 500K, condición de aplicabilidad que haya ≥ 5 paquetes de 100K en la comarca; `descomprimir_paquete` que hace lo opuesto, condición de aplicabilidad que haya 1 paquete de 500K en la comarca. Como función heurística usamos: $D * \#desplazamientos + \sum_{c \in C} \lambda * \#paquetes(c) + suma\_ayudas(c) + \epsilon * no\_afin(c) * suma\_ayudas(c)$, donde igual que en el apartado anterior, λ y ϵ son constantes dadas que tienen que ver con el coste burocrático de cada ayuda y con el coste político de dar dinero a partidos opuestos, y $no\_afin(c)$ vale 1 si la comarca es de un partido no afín al Govern y 0 si es afín.

A priori Hill Climbing parece una mala opción, dado que se nos presiona a dar la solución óptima y Hill Climbing rara vez puede garantizarla, pero si no encontrásemos nada mejor sería una opción. Partimos de un estado que es solución, ya que se garantiza la cobertura de todos los refugiados, y con un coste de creación pequeño (O(C)) Tenemos 5 operadores propuestos, con ramificaciones: mover refugiados (C^2), añadir ($2C$), quitar ($2C$), comprimir ($2C$) y descomprimir ($2C$), que no parecen ser exageradamente grandes, y el valor del heurístico es directamente la calidad a optimizar, que querríamos minimizar.

Sin embargo, ahí termina lo positivo que podemos decir de este modelo. En primer lugar, si queremos permanecer en el espacio de soluciones, tendríamos que añadir condiciones de aplicabilidad a los operadores de mover refugiados y de quitar paquetes: mover refugiados no puede mover refugiados a una comarca sin presupuesto para cubrirlos, y quitar debería mirar que no dejamos a familias sin cubrir. Además, dos de los operadores (descomprimir y añadir paquete) no se van a usar jamás, puesto que lo único que hacen es incrementar la heurística. Lo mismo se puede decir del operador de mover refugiados, ya que si empezamos con todos los refugiados cubiertos y no permitimos no-soluciones, lo único que causa mover refugiados es subir costes. Por motivos similares, podemos garantizar que el modelo no explora todo el espacio de soluciones, ya que soluciones que impliquen añadir un paquete en una comarca que no lo tuviera en la solución inicial nunca van a ser exploradas. Todos estos problemas ya hacen de este modelo uno que no tiene mucho sentido probar, ya que la asignación de paquetes inicial ya es prácticamente definitiva. Lo único que hace este modelo es una asignación greedy de paquetes (por inicialización) y una compresión de paquetes, y poco hay que pueda cambiarla.

Por otro lado, alguien podría proponer permitir explorar en el espacio de no soluciones, permitiendo que haya familias no cubiertas pero penalizando esto en el heurístico. Sin embargo, esto no ayuda prácticamente nada, ya que deberíamos hacer que el penalizador fuera lo suficientemente grande como para que, dada una solución en que hay una familia no cubierta (y no haya comarcas donde desplazar K familias no se pase del presupuesto), forzase al algoritmo a poner un paquete de 100K en esa comarca (que potencialmente es de un partido rival). Dado el tamaño del penalizador, rápidamente volveríamos a una situación donde el algoritmo no sale del espacio de soluciones: no le conviene mover a K familias a un sitio donde al menos una no está cubierta porque esto conlleva un heurístico muchísimo mayor.

En resumen, si bien el algoritmo es eficiente, no explora todo el espacio de soluciones (de hecho, no explora prácticamente nada) y hará una exploración muy cándida que se quedará en óptimos pésimos. El modelo es difícilmente mejorable sin un cambio radical de planteamiento.

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.