3. METAHEURÍSTICA GRASP

3.1. Introducción a las técnicas metaheurísticas

Las técnicas metaheurísticas son nuevas herramientas que han sido desarrolladas en los últimos años para la resolución de diferentes tipos de problemas que antes no habían podido ser abordados debido a su complejidad.

Consisten en algoritmos que proporcionan soluciones factibles (admisibles), que, aunque no alcancen el óptimo, al menos se acercan a su valor en un tiempo de cálculo razonable. Aunque en principio parezca lo contrario, en la mayoría de los casos nos encontramos con procedimientos muy simples, muchas veces basados únicamente en el sentido común.

Son varias las situaciones en las que resulta muy recomendable la utilización de este tipo de algoritmo:

- Cuando no existe un método exacto de resolución o éste requiere mucho tiempo de cálculo o memoria.
- Cuando no se necesita una solución óptima, sino una solución satisfactoria que oriente sobre el comportamiento del problema.
- Cuando los datos son poco fiables.
- Cuando las limitaciones de espacio, tiempo, etc, obliguen al empleo de métodos de respuesta rápida.
- Como paso intermedio en la aplicación de otro algoritmo.

El uso de técnicas metaheurísticas como método de resolución de problemas de optimización está, a día de hoy, muy extendido, debido entre otros, a los siguientes motivos:

- Suelen ser más fáciles de entender por parte de gente no experta, el entendimiento de las heurísticas que los complejos métodos que utilizan la mayoría de las técnicas exactas.
- Mayor flexibilidad para el manejo de las características del problema.
- No suelen ser complejas incluso para funciones no lineales.

No obstante, estas técnicas también presentan inconvenientes. Uno de los más destacados es que, en la mayoría de los casos en los que estos algoritmos son utilizados, no se conoce el valor del óptimo, y dado que son

procedimientos en los que no se asegura la optimalidad, no se puede tener una idea clara del nivel de calidad de la solución obtenida.

Son muchas las metaheurísticas creadas hasta el día de hoy. Unas, tienen su uso limitado a problemas particulares, mientras que otras pueden ser aplicadas a problemas más diversos. Entre estas últimas podríamos destacar las siguientes como las más importantes:

- Recocido simulado.
- Algoritmos genéticos.
- GRASP.
- Búsqueda Tabú.

3.2. Descripción de la metaheurística GRASP

La palabra GRASP proviene de las siglas de “Greedy Randomized Adaptative Search Procedure”, que en castellano sería algo así como: Procedimientos de Búsqueda basados en funciones “Greedy” Aleatorizadas Adaptativas. Se trata de una metaheurística basada en un procedimiento de búsqueda avaricioso, aleatorio y adaptativo el cual, aplicado a una amplia gama de problemas de optimización, garantiza una buena solución, aunque no necesariamente la óptima.

La metodología GRASP fue desarrollada al final de la década de los ochenta con el objetivo inicial de resolver problemas de cubrimientos de conjuntos. El término GRASP fue introducido por Feo y Resende en 1995 como una nueva técnica metaheurística de propósito general.

Este tipo de implementaciones son generalmente robustas, en el sentido de que es difícil encontrar ejemplos patológicos en donde el método funcione arbitrariamente mal.

GRASP es un procedimiento iterativo donde cada paso consiste en una fase de construcción y una de mejora.

En la fase de construcción se construye iterativamente una solución factible, añadiendo un elemento en cada paso. En cada iteración la elección del próximo elemento para ser añadido a la solución parcial viene determinado por una función greedy. Esta función mide el beneficio de añadir cada uno de los elementos y se elige la mejor. Es necesario resaltar el hecho de que esta medida es “miope” en el sentido que no tiene en cuenta qué ocurrirá en

iteraciones sucesivas una vez que se hace una elección, sino únicamente en la iteración actual.

Se dice además que el procedimiento es adaptativo porque en cada iteración se actualizan los beneficios obtenidos de añadir el elemento seleccionado a la solución parcial. Es decir, la evaluación que se tenga de añadir un determinado elemento a la solución en la iteración *j* no coincidirá necesariamente con la que se tenga en la iteración *j+1.*

Por otro lado, el heurístico es aleatorio porque no selecciona el mejor candidato según la función greedy adaptada sino que, con el objeto de diversificar y no repetir soluciones en dos construcciones diferentes, se construye una lista con los mejores candidatos de entre los que se selecciona uno al azar.

Dado que la fase inicial no garantiza la optimalidad local respecto a la estructura de entorno en la que se esté trabajando (notar que hay selecciones aleatorias), se aplica un procedimiento de búsqueda local como posprocedimiento para mejorar la solución obtenida.

Hemos de tener en cuenta que, en un problema de optimización combinatoria dado, cada solución del problema tiene un conjunto de soluciones asociadas al que se denomina entorno o vecindad de la solución. Así, dada una solución, podemos calcular cualquier solución perteneciente a su entorno mediante una operación denominada movimiento.

Al realizar el GRASP muchas iteraciones, nos está ofreciendo un muestreo del espacio de soluciones existente.

Atendiendo a todo lo anterior, el funcionamiento del GRASP se puede estructurar de la siguiente manera:

1. Fase Constructiva:
    - Generar una lista de candidatos mediante la utilización de la función greedy.
    - Considerar una lista restringida de los mejores candidatos.
    - Seleccionar aleatoriamente un elemento de la lista restringida.
    - Repetir el proceso hasta constituir una solución de partida.

2. Fase de Mejora:
    - Hacer un proceso de búsqueda local a partir de la solución construida hasta que no se pueda mejorar más.
3. Actualización:
    - Si la solución obtenida mejora a la mejor almacenada, actualizarla.

En la fase de construcción se va generando la solución admisible de partida de forma iterativa, es decir, eligiendo un elemento en cada iteración. En cada iteración de esta fase, la elección del siguiente elemento que formará parte de la solución se determina elaborando una lista de candidatos (LC) con todos los elementos que pueden entrar a formar parte de la solución en esa iteración. Los elementos de esta lista de candidatos se ordenan con respecto a la función greedy que mide, como ya hemos comentado, el beneficio asociado a cada uno de los elementos, creándose a partir de estos valores la llamada *lista restringida de candidatos* (RCL). Esta lista incluirá a aquellos elementos cuyos valores de la función greedy sean más beneficiosos desde el punto de vista del criterio de optimización definido. Una vez constituida esta lista, se seleccionará aleatoriamente un elemento de la misma, que automáticamente pasará a formar parte de la solución de partida.

Un pseudocódigo de la fase de construcción se puede ilustrar como sigue:

```
Procedimiento de ConstrucciónSolución (Solución)
    Solución = {}
    MIENTRAS La solución no esté completa
         Actualizar la función greedy
         Hallar RCL (RCL)
         a = SelecciónAleatoriaElemento (RCL)
         Solución = Solución + {a}
    FIN MIENTRAS
Fin
```

Al igual que ocurre en muchos métodos deterministas, las soluciones generadas en la fase de construcción de un GRASP no garantizan la optimalidad local respecto a una definición cualquiera de vecindad, de aquí el interés de realizar una búsqueda local posterior a la fase construcción.

Un procedimiento de búsqueda local parte de una solución inicial, calcula su entorno o vecindad y escoge una nueva solución perteneciente a dicho entorno. Dicho de otro modo, realiza un movimiento que, aplicado sobre la solución de partida, da como resultado una nueva solución que pertenece a la

vecindad de la de partida. En el GRASP, este proceso es utilizado reiteradamente: En cada iteración el algoritmo hace un movimiento, “visitando” una nueva solución del entorno de la solución de partida.

Un pseudocódigo genérico del GRASP podría ser el siguiente:

```
Procedimiento GRASP (NumIter, Semilla)
    MejorSolución = 0
    Para k = 1 hasta NumIter
         P^S = ConstruirSolución (Semilla)
         P^L = BúsquedaLocal (P^S)
         Si Valor (P^L) < Valor (MejorSolución) entonces
             MejorSolución = P^L
         Fin si
         Devolver (MejorSolución)
      Fin Para
Fin
```

3.3. Análisis de los parámetros del algoritmo

En este proyecto, vamos a aplicar la metaheurística GRASP sobre dos versiones del problema POC: En la primera de ellas se tratará de maximizar la suma de los pesos de los trabajos realizados, mientras que en la segunda, el objetivo a conseguir será maximizar el número de trabajos procesados suponiendo igual peso para todos ellos.

Teniendo en cuenta que los resultados obtenidos en ambos casos dependen en gran medida del valor que tomen una serie de parámetros con los que trabaja la metaheurística, debemos realizar un severo estudio que nos permita identificar cuáles de estos parámetros tienen una mayor influencia sobre el funcionamiento de la metaheurística y en qué valores concretos tenemos que fijar éstos de forma que se obtengan los mejores resultados.

Los parámetros que, por su importancia, han sido seleccionados para la realización del estudio son los detallados a continuación:

Función índice: El criterio de optimización elegido para la resolución de este tipo de problema hace que tanto el peso como la duración de los trabajos sean factores determinantes en la resolución de los mismos. Si lo que queremos conseguir es maximizar la suma de los pesos de los trabajos procesados, parece lógico que los índices de trabajo estén basados en los parámetros anteriores. Por ello, vamos a elegir índices que dependan del peso y de la

duración, pues hemos de tener en cuenta que, dado que existen solapamientos entre los trabajos y estamos supeditados a restricciones de tiempo, debemos mostrar interés por incluir en la solución aquellos trabajos que posean un peso por unidad de longitud lo más alto posible.

Tamaño de la RCL: El tamaño de la RCL tiene un papel decisivo en la fase de construcción de la solución en la metaheurística GRASP, ya que de su valor depende el número de trabajos aspirantes a entrar en la solución en cada una de las iteraciones de esta fase. Dado que la selección del candidato es un proceso totalmente aleatorio, la elección de una RCL de gran tamaño, aunque aumenta la aleatoriedad, permite que trabajos con valores del índice no excesivamente buenos puedan llegar a formar parte de la solución, mientras que una RCL de pequeño tamaño, aunque hace que sólo los mejores trabajos sean candidatos a entrar en la solución, repercute en que el método pierda aleatoriedad. Por tanto, escoger el tamaño adecuado es fundamental para obtener resultados de calidad.

En este primer caso vamos a considerar dos vías de estudio:

En la primera de ellas, consideraremos que, en la fase constructiva del método, para que un trabajo pueda entrar en la RCL y así poder aspirar a formar parte de la solución el valor de su índice deberá ser mayor que el mejor de los índices de los trabajos a procesar multiplicado por una determinada constante α. Tendremos pues que el tamaño de la RCL podrá ser distinto en cada una de las iteraciones del proceso de construcción pues el número de candidatos a entrar en la solución dependerá del valor del índice de cada uno de los trabajos.

En la segunda de las vías consideraremos que el tamaño de la RCL permanecerá constante durante todo el proceso constructivo, de manera que el número de trabajos que van a entrar en la lista estará fijado de antemano.

En ambas vías, una vez seleccionado el trabajo que pasará a formar parte de la solución, la asignación se llevará a cabo de dos maneras:

En un primer caso, la asignación del trabajo a uno de los tipos de máquinas con los que es compatible se realizará de forma aleatoria, es decir, una vez determinados los tipos de máquinas con los que el trabajo seleccionado es compatible, se seleccionará un tipo de máquina al azar y el trabajo será realizado por una de las máquinas pertenecientes a ese tipo.

En el segundo de los casos, a la hora de asignar el trabajo a uno de los tipos de máquinas, se buscará, entre todos los tipos de máquinas compatibles, aquel que hasta ese momento acumule un tiempo de procesamiento menor, es decir, se seleccionará para la asignación el tipo de máquina que esté más "libre" hasta ese momento, consiguiéndose así una asignación más uniforme.

Número de iteraciones: El número de iteraciones, que en nuestro caso funcionará además como criterio de parada, se puede definir como el número de veces que la metaheurística realiza el proceso completo de construcción y mejora de la solución. Según esto, el algoritmo generará tantas soluciones como iteraciones realice y cuanto mayor sea dicho número mayor será la probabilidad de encontrar mejores soluciones. Ahora bien, hay que tener en cuenta que un número excesivo de iteraciones repercute negativamente en el tiempo empleado por el algoritmo en la obtención de la mejor solución.

Un número adecuado de iteraciones será pues aquél que nos permita obtener una solución de calidad en un tiempo razonable.

Una vez probadas con todos los índices todas las alternativas anteriores y para un número de iteraciones constante, se seleccionará el índice que mejores resultados haya obtenido y será con éste con el que compararemos, a continuación, la metaheurística GRASP con los algoritmos Tabú y exacto.

En la segunda de las aplicaciones de la metaheurística GRASP sobre el problema POC, la morfología del problema cambia completamente: Al compartir todos los trabajos el mismo peso, éste pierde su papel relevante, de forma que la función índice únicamente se basará en la duración de las tareas a procesar.

Si exceptuamos el valor en el que se basa la función índice, podemos considerar que el comportamiento de la metaheurística GRASP en ambas aplicaciones es muy similar. Por ello, en lo que se refiere al tamaño de la RCL y al número de iteraciones, aprovecharemos los resultados obtenidos en la primera de las aplicaciones para concretar los valores a utilizar.

3.4. Ejemplo de funcionamiento de la metaheurística.

Para clarificar el procedimiento de trabajo de la metaheurística GRASP aquí implementada y teniendo en cuenta su carácter repetitivo, vamos a resolver una iteración del siguiente ejemplo:

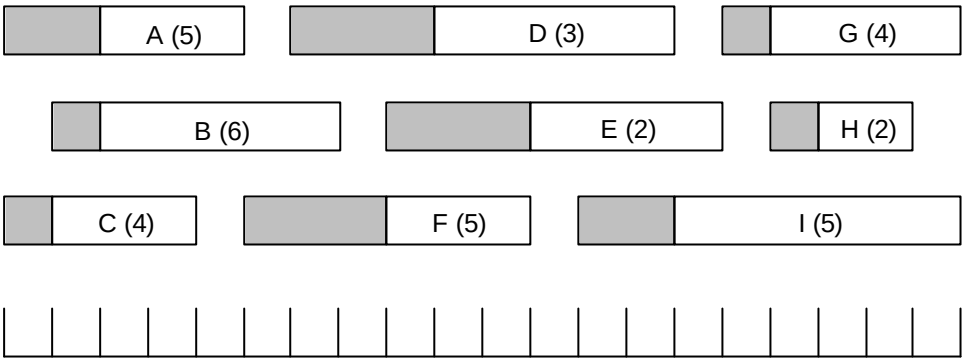

Figura 3.1: Formulación del problema.

Supongamos que tenemos que procesar un conjunto de trabajos como el descrito en la figura en el que las cantidades entre paréntesis indican el valor asociado a cada tarea.

Por restricciones de presupuesto y diseño, sólo disponemos de dos máquinas, M_1 y M_2, pertenecientes a los tipos c_1 y c_2 respectivamente, de forma que cada una de estas máquinas sólo podrá procesar un trabajo en cada instante de tiempo.

Considerando que sólo existen dos clases de trabajos, a_1 y a_2, la matriz de incidencia *L* viene dada por:

L	**c_1**	**c_2**
a_1	1	0
a_2	0	1

Tabla 3.1: Matriz de incidencia del problema a resolver.

La clase de trabajo a la que pertenece cada una de las tareas, se representa en la siguiente tabla:

	a_1	**a_2**
A	1	0
B	0	1
C	1	0
D	0	1

E	1	0
F	0	1
G	1	0
H	0	1
I	1	0

Tabla 3.2: Trabajos incluidos en cada uno de los tipos de trabajo.

Según se ha descrito anteriormente, son varias las posibilidades de resolución planteadas en este proyecto. Como caso particular y teniendo en cuenta que la resolución del resto de los casos es análoga, vamos a considerar una basada en los siguientes parámetros:

- Función Índice: Peso.
- El tamaño de la RCL está fijado en 3 unidades y será constante durante todo el proceso.
- Se asignarán los trabajos a las máquinas compatibles de forma aleatoria.

Como ya hemos comentado, el primer paso de la metaheurística GRASP consiste en construir una solución inicial. Dado que la solución construida aun está vacía, la lista de candidatos (LC) estará formada por todos los trabajos. Ordenándolos según su valor, nos encontramos con la siguiente LC: B, A, F, I, C, G, D, E, H.

Teniendo en cuenta el tamaño de la RCL, los candidatos a entrar en la solución van a ser los trabajos B, A, F, pues son los de mayor peso. Seleccionando uno de ellos al azar, por ejemplo el F, ya tenemos el primer elemento de la solución, que, por razones de compatibilidad, será procesado por la máquina M_2. Los resultados de esta primera etapa son los siguientes:

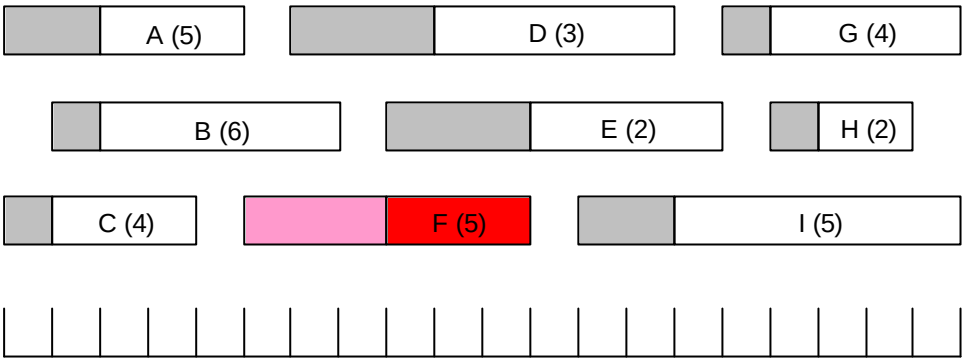

Figura 3.2: Diagrama de trabajos resultante de la primera etapa de la fase constructiva.

Lista de trabajo de M_1: Æ
Lista de trabajo de M_2: F
Valor de la FO: 5
Solución Construida = {F}

Tabla 3.3: Resultados de la primera etapa de la fase constructiva.

Una vez completada esta primera etapa, nos encontramos con que el trabajo D perteneciente a la clase a_2 se solapa con el trabajo F, que ya ha sido asignado a la máquina M_2. La inclusión de dicho trabajo en la solución será pues imposible, por lo que no deberemos incluirlo en la LC. Dicha lista estará formada entonces por las tareas B, A, I, C, G, E y H, pasando a la RCL los trabajos B, A e I. Seleccionando aleatoriamente el trabajo I, los resultados obtenidos son:

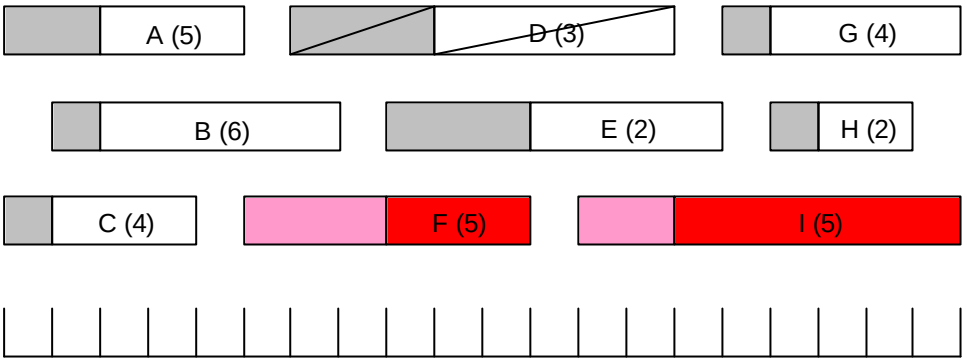

Figura 3.3: Diagrama de trabajos resultante de la segunda etapa de la fase constructiva.

Lista de trabajo de M_1: I
Lista de trabajo de M_2: F
Valor de la FO: 5+5 = 10
Solución Construida = {F, I}

Tabla 3.4: Resultados de la segunda etapa de la fase constructiva.

Al comienzo de la tercera etapa ya tenemos las dos máquinas ocupadas. Siguiendo el razonamiento descrito hasta ahora, la LC estará formada por los trabajos B, A, C, G, E y H, entrando a formar parte de la RCL las tareas B, A y C. Seleccionando de nuevo uno de ellos al azar, por ejemplo el C los resultados que obtenemos son los siguientes:

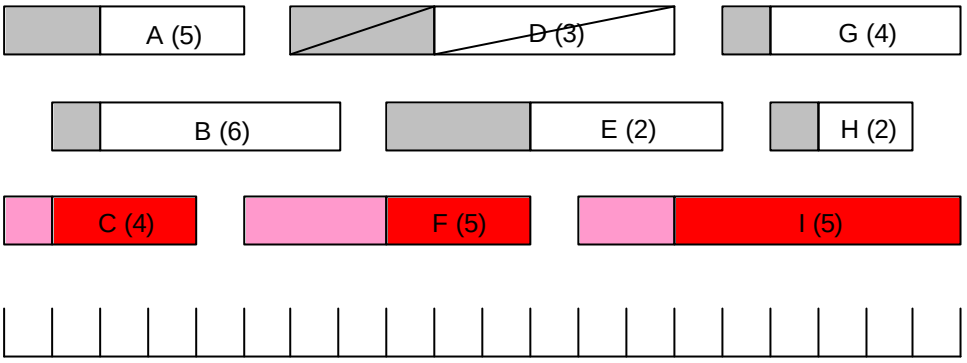

Figura 3.4: Diagrama de trabajos resultante de la tercera etapa de la fase constructiva.

Lista de trabajo de M_1: C, I
Lista de trabajo de M_2: F
Valor de la FO: 4+5+5 = 14
Solución Construida = {C, F, I}

Tabla 3.5: Resultados de la tercera etapa de la fase constructiva.

En la cuarta etapa nos encontramos con un caso similar al descrito anteriormente. Al incluir el trabajo C en la lista de asignaciones de la máquina M_1, el trabajo A, que es de la misma clase y está solapado con el anterior, queda fuera de la LC, quedando ésta constituida por los trabajos B, G, E y H. Añadiendo las tareas B, G y E a la RCL y seleccionando de forma aleatoria el trabajo B, los resultados son los descritos a continuación:

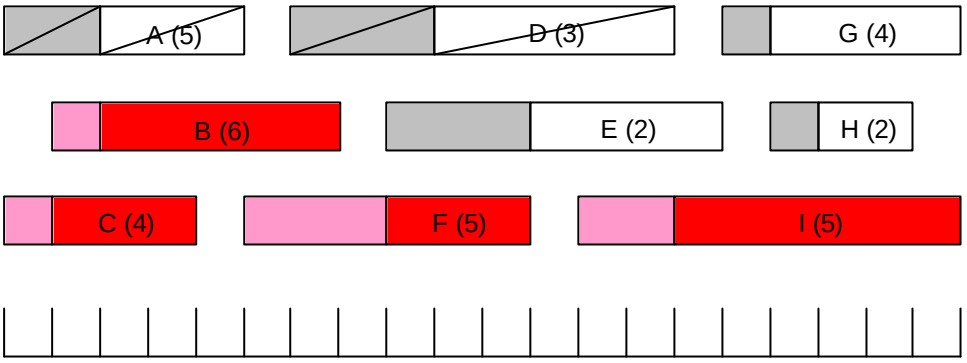

Figura 3.5: Diagrama de trabajos resultante de la cuarta etapa de la fase constructiva.

Lista de trabajo de M_1: C, I
Lista de trabajo de M_2: B, F
Valor de la FO: 6+4+5+5 = 20
Solución Construida = {B, C, F, I}

Tabla 3.6: Resultados de la cuarta etapa de la fase constructiva.

Dado que el proceso se repite y no se van a producir más solapamientos, vamos a resolver el resto de las etapas de la construcción de la solución de forma esquemática:

Etapa 5:

- LC: G, E y H.
- RCL: G, E y H
- Trabajo seleccionado: G.

Etapa 6:

- LC: E y H.
- RCL: E y H.
- Trabajo seleccionado: E.

Etapa 7:

- LC: H.
- RCL: H.
- Trabajo seleccionado: H.

El resultado de la fase de construcción de la solución es el siguiente:

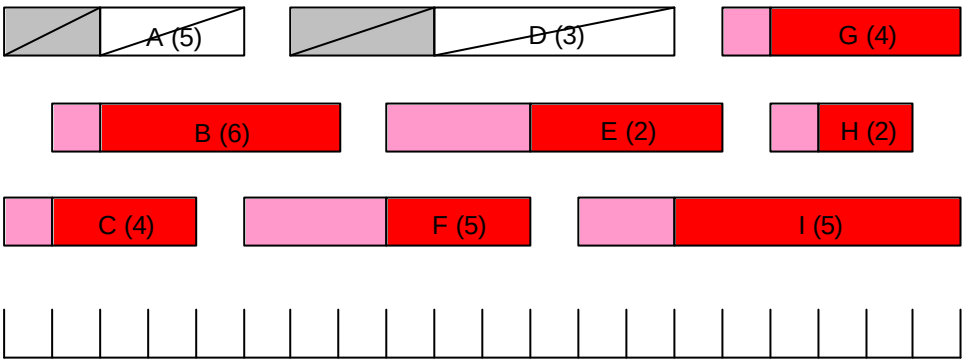

Figura 3.6: Diagrama de trabajos resultante de la fase constructiva.

Lista de trabajo de M_1: C, E, I, G
Lista de trabajo de M_2: B, F, H
Valor de la FO: 6+4+2+5+4+2+5 = 28
Solución Construida = {B, C, E, F, G, H, I}

Tabla 3.7: Resultados de la fase constructiva.

Una vez terminada la fase de construcción, la metaheurística GRASP somete a la solución construida a un proceso de búsqueda local en el que se determinará la vecindad de la misma y las variaciones de la función objetivo con cada uno de los movimientos. Este proceso finalizará cuando el valor asociado al mejor movimiento encontrado no mejore el valor de la FO. En la tabla siguiente se muestran los posibles movimientos asociados a la solución de partida así como los incrementos que se producen en la FO:

MOVIMIENTO	VARIACIÓN FO
Sale C y entra A	+1
Sale F y entra D	-2

Tabla 3.8: Movimientos asociados a la solución construida.

De los dos movimientos existentes, el único que mejora el valor de la FO es el de considerar la salida de C y la entrada de A. Llevando a cabo este movimiento, el resultado obtenido es el siguiente:

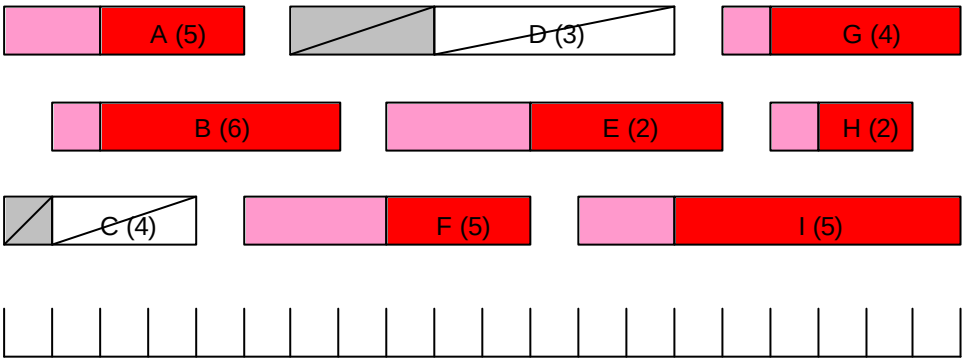

Figura 3.7: Diagrama de trabajos tras la aplicación de la búsqueda local por primera vez.

Lista de trabajo de M_1: A, E, I, G
Lista de trabajo de M_2: B, F, H
Valor de la FO: 5+6+2+5+4+2+5 = 29
Solución Construida = {A, B, E, F, G, H, I}

Tabla 3.9: Resultados tras la aplicación de la búsqueda local por primera vez.

Una vez realizado el cambio, volvemos a estudiar los movimientos existentes, que son:

MOVIMIENTO	VARIACIÓN FO
Sale A y entra C	-1
Sale F y entra D	-2

Tabla 3.10: Movimientos asociados a la solución tras la aplicación de la búsqueda local por primera vez.

Dado que ninguno de los movimientos produce un incremento positivo en el valor de la FO, se guarda la solución y se da por terminada la iteración. Los resultados obtenidos serian:

- **Solución Construida = {A, B, E, F, G, H, I}**
- **Valor de la FO: 29**

A continuación, la metaheurística comenzaría de nuevo todo el proceso, realizándolo tantas veces como iteraciones se hayan fijado de antemano. En cada una de estas iteraciones tomaría la mejor solución encontrada, la compararía con la mejor solución guardada hasta ese momento y ofrecería, al final, la mejor de todas.