Resolución de la instancia “case 8.3” y generalización del método

1. Es posible reciclar la primera parte del código del programa que ya había realizado. Esta parte simplemente **almacena los datos de la instancia** (ver Excel “case 8.3” adjunto) **en un vector de estructuras**. El programa pide ahora que se teclee el número entero de horas en el que queremos que se complete la instancia, el “**takttime**”. Se supone para cada día una duración de 8 horas de cara a facilitar el redondeo al aplicar “**GAMMA**” distinto de uno (sólo se trabajará con horas sin decimales). Se recomienda ver la **Figura 1** (Esquema del “case 8.3”, página 5).

2. Para determinar las actividades que no tienen predecesores, se emplea un bucle que compruebe que **PREC(itask, j)==0 para todo itask manteniendo j fijo**, empezando por j=1. Esto se cumple para las tareas 1, 2 y 9. Cuando algún PREC(itask, j) sea distinto de 0, el bucle termina (con un break). De esta forma la columna PREC(itask, 9), que marca el final del programa, no es tenida en cuenta y a partir de la tarea 2 ya no entra en el set E1 ninguna más. **En el set E1 tendremos pues, las tareas 1 y 2**.

Ahora se deberá evaluar el peso de cada tarea según la regla de prioridad elegida. Para este ejemplo, emplearé **GRPW** (**suma de la duración de la tarea y de la duración de todos sus sucesores**), ya que me parece la más prometedora entre las dos que había pensado.

3. La tarea 1 tiene como sucesores la tarea 3 (PREC(1,3)==1) y la tarea 6 (PREC(3,6)==1). Con un bucle se comprobará la fila uno de la matriz de precedencias y se almacenará la/s componente/s j de PREC(itask, j) tales que PREC(1, j)==1. Puesto que PREC(1,3)==1, ahora se tendrá que comprobar la fila tres. PREC(3,6)==1 con lo que tenemos como sucesor la tarea 6 también. Al comprobar la fila seis se observa que ya no hay ningún uno más y el bucle termina con un break. Se suman las duraciones de las tareas 1, 3 y 6 Dur(1)+Dur(3)+Dur(6) obteniéndose un **peso para la tarea 1 de 88 horas**.

   Siguiendo la lógica empleada para la tarea 1, se calcula el peso de la tarea 2.
   La tarea 2 tiene como sucesores las tareas 4 (PREC(2,4)==1) y 5 (PREC(2,5)==1), la tarea 6 (PREC(4,6)==1), la tarea 7 (PREC(5,7)==1) y la tarea 8 (PREC(7,8)==1). Esto se es determinado siguiendo los siguientes pasos:

   - Se comprueba la fila dos y se obtienen como sucesores las tareas 4 y 5.
   - Se comprueba la fila cuatro y se obtiene como sucesor la tarea 6.
   - Se comprueba la fila cinco y se obtiene como sucesor la tarea 7.
   - Después de la tarea 6 no hay sucesores pues en la fila seis de la matriz de precedencias sólo hay ceros.
   - Se comprueba la fila siete y se obtiene como sucesor la tarea 8.
   - Después de la tarea 8 no hay sucesores pues en la fila ocho de la matriz de precedencias sólo hay ceros.

   Se suman las duraciones de las tareas 2, 4, 5, 6, 7 y 8 **para obtener un peso de 144**.

4. **Estos pesos nos sirven de base para aplicar "Biassed Randomization"**. Se podrá emplear una distribución triangular u otras, como ejemplo yo simplemente calcularé el **peso relativo de cada tarea respecto al total**. Experimentar con el código será necesario para realizar la "Biassed Randomization" de la manera más efectiva posible. El total será 144+88=232. El peso relativo de la tarea 1 es 88÷232=0,3793 y de la tarea 2 será de 144÷232=0,6206.

   Multiplicando por 100 y redondeando, tenemos para la tarea 1 un peso relativo 38 y para la tarea 2, de 62. Creo un **vector de 38+62 componentes**, conteniendo un **uno** (**tarea 1**) en las **38 primeras componentes** y un **dos** (**tarea 2**) en las **62 siguientes**. Con "**srand(time(NULL)+i)**" se inicializa la semilla con el reloj del ordenador, y un "**rand()%(38+62+1)**" nos permite obtener una componente del vector de manera pseudo-aleatoria (se puede asumir que es totalmente aleatorio).

   Pongamos, por ejemplo, que hemos obtenido la componente 36 del vector, **la tarea a programar será la tarea 1**. Cada vez que el programa pase de este punto se hace **i++** con lo que se **asegura que la semilla se inicializa con valores distintos en cada pasada** (si no, habría que esperar un segundo a que variara el reloj del ordenador).

5. Carece de sentido evaluar si existe una restricción **"non-parallel"** (INC(itask, j)==1) o una de **"time-lag=0"** (CONSEC(itask, j)==1) para este primer set ya que al ser la primera tarea programada estas dos restricciones no tienen influencia.

6. Como normal general, **las tareas se programarán en el primer intervalo en el que haya disponibilidad de recursos y en el que se respeten las dos restricciones mencionadas** anteriormente. En este caso, por ser la primera, se programaría en el intervalo (0, 24) donde 24 es la duración de la tarea 1 en horas.

Como estipular cuántos trabajadores se emplean, el tipo (M o S) me parece la parte más problemática del código. Voy a suponer que **se fijan los trabajadores disponibles de M y S al principio del programa**. Esta parte del programa necesitaré ejecutarla para ver si existen mejores opciones. En cuanto a la manera de estipularlos, **en la primera ejecución se toma el máximo de la columna "MinOP" de la instancia tanto de M como de S**. Para este caso serán 2 de S y 3 de M (**en rojo en el Excel**).

Tras 10 ejecuciones (por ejemplo) del código completo, se supone que el tiempo en completarse la instancia ha sido de 220 horas como mínimo (en otras ejecuciones se habría obtenido 230, 240, 235… por ejemplo) y el "takttime" introducido en el principio del programa es 150 (por ejemplo). **La solución obtenida se aleja más de un 20% (por ejemplo) del "takttime" introducido, por lo serán necesarios más M o más S**.

Se ejecuta todo el programa de nuevo con 1 trabajador S más, de manera que empleamos 3 de S y de 3 M. Si ocurre lo mismo probamos con 3 de S y 4 de M. Si no se consigue, una vez más, una solución próxima al "takttime", se probaría con 4 de S y 4 de M.

Si siguiendo este sistema se comprueba que por mucho que aumenten los trabajadores no nos acercamos a 150 horas lo suficiente, aparecerá un mensaje por pantalla **"El takttime introducido es demasiado pequeño para resolver la instancia, introduzca uno mayor"**.

Puede que este método no sea el mejor, podría ser necesario modificarlo. Si no se limitan los trabajadores S y M creo que sería difícil decidir cuándo dejar de usar más trabajadores por ello me parece que éste es el mejor enfoque.

7. Se supone que ya estamos con los **trabajadores limitados a 3 de S y 3 de M.** Se crean **4 vectores** de 500 componentes (o más si es necesario para asegurar que el tiempo para resolver la instancia es menor que el total de componentes de este vector). El vector correspondiente a **S** contendrá un 3 en todas sus componentes y el correspondiente a **M** contendrá un 3 en todas sus componentes. Habrá otro vector correspondiente al **área A** (todas sus componentes contendrán un 6 dado que **Cap(A)=6** en este caso) y otro para el **área B** (todas sus componentes contendrán un 6 dado que **Cap(B)=6** en este caso).

   La componente "0" de cada uno de los vectores representa la hora 0, la "1" la hora 1 y así sucesivamente. **La tarea 1 puede emplear 2 o 3 trabajadores S y sólo puede usar la zona A**. Si se emplean 3 trabajadores, dado que GAMMA(1,3)=0.75, el intervalo en el que se programa la tarea 1 pasaría a ser (0, 18) pues 24*0,75=18 horas. Para la decisión de emplear 2 o 3 trabajadores se deberá tener en cuenta que estén disponibles (en este caso sí, pues hay 3 de S disponibles). También se tendrá en cuenta el peso relativo de cada tarea. La tarea 1 tenía peso 38 y la tarea 2 peso 62. **En este caso no parece interesante emplear 3 trabajadores de S**.

   Un criterio para decidir si aumentar los trabajadores respecto del mínimo (2 para la tarea 1) podría ser que el **peso relativo de la tarea 1, fuera mayor que el total del resto de tareas a programar en esta etapa** (aunque en este caso sólo hay otra, la tarea 2). En caso de ser posible el aumento en dos unidades (o tres, cuatro, cinco…) respecto al mínimo de trabajadores, un criterio a seguir podría ser que el **peso relativo de la tarea fuera el doble que el de la segunda de mayor peso** (por ejemplo). **Se podría introducir "Biassed Randomization" en esta parte del programa también**. La tarea 1 debe realizarse en la zona A. Se descontarán por tanto 2 trabajadores S del vector de S con lo que las primeras 24 componentes pasarán a ser 1 (3-2), y se descontarán 2 del vector del área A con lo que las primeras 24 componentes tendrán valor 4 (6-2).

8. **Ya programada la tarea 1, y descontados los recursos empleados**, ésta se elimina del conjunto de actividades a programar y **pasa a formar parte de la solución**. Se creará un vector solución de manera que este tenga tantas componentes como tareas tenga la instancia siendo la tarea 1 la componente "0" la tarea 2 la componente "1" y así sucesivamente. **Inicialmente este vector sólo contendrá ceros y a medida que entren tareas en la solución éstas pasarán a ser 1**. En este caso **la componente "0" pasa a ser 1 ya que la tarea 1 entra en la solución**.

9. Ahora, se deberán añadir al conjunto de actividades a programar, los **sucesores de la tarea 1 tales que todas sus tareas predecesoras tienen un valor 1 en el vector solución**. A la tarea 2 por tanto, se deberá añadir la tarea 3 en el conjunto de actividades a programar. La tarea 3 tiene como único predecesor la tarea 1, y ésta tiene un valor 1 en el vector solución.

10. En ésta y en las siguientes programaciones, ahora sí, **se deberá tener en cuenta la/s restricción/es non-parallel y la/s restricción/es time-lag=0**. En este caso CONSEC(4,6)=1 y el resto son 0 con lo que **la tarea 6 se debe ejecutar inmediatamente después de la tarea 4**. En este caso INC(6,7)=1 con lo que **la tarea 6 y la tarea 7 no pueden compartir intervalo temporal en el que se estén ejecutando**. Si dentro del conjunto de tareas a programar, está la tarea 6, no será necesario calcular los pesos de cada una de las tareas puesto que **se programará directamente la tarea 6 en un intervalo inmediatamente posterior a la tarea 4**.

Si esto no fuera posible, se volvería al principio del programa. Respecto a la restricción "non-parallel", suponiendo que la tarea 6 se ha programado primero, y resulte ser la tarea 7 la tarea a programar tras aplicar la regla de prioridad y "Biassed Randomization", **hasta que la 6 no termine no se podrá programar la 7**.

En general y respetando estas dos últimas restricciones, las tareas se programarán en el tiempo más temprano en el que haya recursos suficientes. En el caso de que haya que decidir entre **área A o B** (para éste caso en ninguna tarea) **se elegirá la que tenga mayor disponibilidad y lo mismo se hará en caso de que se deba elegir entre trabajadores M o S** (tareas 4 y 7 en este caso).

Si sólo hay una tarea en el conjunto de tareas a programar, no será necesario aplicar la regla de prioridad ni "Biassed randomization" y **se programará directamente en el primer intervalo que sea factible.**

11. **El programa termina cuando todas las tareas se han programado**. Como se ha comentado previamente, si tras 10 ejecuciones (por ejemplo), no se ha conseguido completar la instancia en un tiempo suficientemente cercano al "takttime" estipulado, se incrementará la disponibilidad de recursos de trabajadores M y S (realizando los incrementos de la forma expuesta entre los apartados 6. y 7.).

Figura 1: Esquema del “case 8.3”. Los recuadros **rellenos de azul** corresponden a las tareas que han de ejecutarse con **time-lag=0** (de manera consecutiva) y los recuadros con los **bordes más gruesos** corresponden a las **tareas que no pueden ejecutarse a la vez**.