**3.2. Fase 2: Adaptación del modelo MILP a un escenario en el que podría llegar un trabajo urgente en el tiempo t**

El modelo anterior representaba un escenario en el que no se suponía que existieran solicitudes urgentes. Sin embargo, en la práctica, llegan solicitudes urgentes a diario y deben ser tratadas con un alto grado de prioridad en comparación con el resto de los trabajos en la línea de espera.

Así, si una solicitud urgente llega cuando una solución óptima de planificación ya ha comenzado a ejecutarse (j^urg), tendremos que detener el proceso actual y recalcular una nueva solución que considere la pieza urgente como prioritaria y que actualice los datos de entrada iniciales, ya que probablemente se hayan iniciado y terminado operaciones de un lote fraccionado de piezas.

Una de las modificaciones que se deben hacer será actualizar la lista de trabajos listos para ser planificados colocando el trabajo j^urg como el primer elemento en la línea de espera. Así, siendo J̃ un subconjunto de J que contiene trabajos que ya han comenzado sus operaciones, tendremos un nuevo conjunto de trabajos a replanificar: J’ = J - J̃ + {j^urg}. Además, tendremos que asegurarnos de que los datos asociados con los elementos de J’ estén correctamente cargados en las matrices y vectores vinculados con los tiempos de procesamiento, fechas de entrega, longitudes e indicadores de última máquina.

Por otro lado, al momento de la llegada de la orden urgente, ciertos trabajos pueden estar en proceso en las máquinas; por lo tanto, al no tener la posibilidad de interrumpir trabajos, nos vemos obligados a esperar hasta el final de las operaciones en curso. Estos tiempos restantes y las demás restricciones asociadas al uso de estaciones y máquinas por los trabajos en proceso pueden considerarse en el modelo como constantes de inicialización para el nuevo modelo de recuperación.

Entonces, dado que el concepto de urgencia se expresa en el sentido de prioridad de tratamiento en comparación con otros trabajos, podemos considerar que la orden j^urg tiene una fecha de entrega más estricta pero factible en comparación con los demás trabajos. Por ejemplo, igual a la suma de sus tiempos operativos en las máquinas, más los tiempos de transferencia relacionados con los trabajos en curso y consigo mismo, más los tiempos operativos de los trabajos en proceso.

En el modelo original, se cargaban trabajos en estaciones justo antes de ser enviados a su primera máquina. Esto nos permite asumir que no tendremos el problema de descargar un trabajo esperando por su estación al momento de la llegada de j^urg. Por otro lado, si un trabajo está bloqueando esta estación porque está siendo procesado en el sistema (trabajo en proceso j^c), será necesario esperar hasta que regrese a la estación antes de colocar la orden urgente. Sin embargo, el número de trabajos de este tipo no puede ser elevado, ya que el sistema admite como máximo un trabajo en cada máquina al mismo tiempo, por lo que una estación será necesariamente accesible cuando llegue j^urg.

Los trabajos en curso en el tiempo t corresponderán a todos los trabajos que salieron de las estaciones antes de la llegada de j^urg pero que aún no han regresado a sus estaciones. Así, podemos definir estos trabajos como j^c ∈ J̃̃, dado que en el modelo y solución original:  
:   
Podemos asumir que, cuando llega j^urg, el peor escenario corresponde al uso paralelo de ambas máquinas, pero por razones de viabilidad, uno de estos dos trabajos (j^c2) debe estar ya en su segunda operación; por lo tanto, si la máquina liberada por este trabajo corresponde a la primera máquina en la secuencia de j^urg, podrá iniciar su operación en esa máquina; de lo contrario, tendrá que esperar hasta el final de la primera operación del otro trabajo (j^c1).

A partir de la solución parcial obtenida mediante el modelo original, se puede identificar la estación libre (s) , where a station está libre si, en la solución y modelo original,  
,  
y podemos usarla para asignar j^urg si se respeta la condición de longitud.

Sin embargo, esta elección será realizada por el modelo y no se hará como una inicialización de parámetros por parte del usuario.

Si j^urg debe esperar a que el trabajo j^c1 acceda a una máquina, eso significará que dicha máquina sería la primera para j^c1 (dejando α = 1), y que también sería la primera para j^urg. Las restricciones temporales relacionadas se presentan en las Ecs. (32) y (33):

Esta restricción me dice que: primero debe salir Jc2 para que cualquier otra pieza pendiente entre (cualquiera)

Este me dice que primero debe salir jc1 para que entren las pendientes (cualquiera) pero la pieza urgente si podria entrar y terminar antes, ya que mientras el otro sigue en proceso el ya puede empezar en la maquina inicial que el otro ya abandono.

Por otro lado, si para comenzar su primera operación j^urg debe esperar a que j^c2 acceda a la máquina, eso significará que esa máquina fue la segunda para j^c2 (dejando β = 1) y la primera para j^urg. La restricción temporal relacionada se presenta en la Ec. (34):

Me dice que, la pieza urg solo saldrá de su primera maquina luego de que jc2 salga primero

CREO QUE HAY QUE EXPLICAR MEJOR QUE ES J^C1 Y J^C2

**Fase 3: Construcción de la heurística basada en MILP**

El método propuesto corresponde a una heurística constructiva basada en un modelo lineal que aplica tanto reglas de orden de prioridad como un enfoque de dimensionamiento y descomposición de lotes.

Con la descomposición de un problema se disminuye el tamaño del mismo dividiéndolo en subproblemas según subconjuntos creados (ya sean variables, restricciones o ambos). Así, al aplicar una descomposición de un problema de tamaño significativo en varios problemas lineales pequeños, pretendemos poder resolverlos de forma exacta, rápida e iterativa.

**Paso 1: Ordenamiento de la lista de piezas**

Tomando las notaciones de parámetros utilizadas en el modelo lineal, podemos representar todos los trabajos en el conjunto J como una lista de elementos que contiene n = |J| trabajos. Dado que existen restricciones temporales que deben respetarse, como fechas de entrega, al crear grupos o subproblemas y resolverlos de forma encadenada, es posible que forcemos al problema a encontrar una solución irrealizable. Esto se tuvo en cuenta al organizar los trabajos antes de particionar los subconjuntos; por lo tanto, los trabajos se ordenarán en orden creciente en la lista según sus fechas de entrega, aplicando así la regla del **Earliest Due Date First (EDD)**, con el fin de garantizar al menos que la solución construida sea realizable. Luego, si dos o más trabajos tienen la misma fecha de entrega, se tendrán en cuenta los valores totales de los tiempos operativos colocándolos en orden creciente con respecto a este indicador, es decir, aplicando la regla del **Shortest Processing Time first (SPT)**.

**Paso 2: Descomposición por lotes**

En la práctica, como se muestra en las Tablas 1 y 2, después de resolver el problema con y sin trabajos urgentes utilizando diferentes instancias y tamaños, nos dimos cuenta de que el tiempo de ejecución (tiempo de resolución) para resolver el problema presentaba un comportamiento de crecimiento exponencial, pero se mantenía relativamente bajo (<0.5 segundos) hasta un lote de 6 trabajos en el sistema.

**Tabla 1. Tiempos de ejecución computacional del modelo MILP no adaptado para trabajos urgentes**

|  |  |  |
| --- | --- | --- |
| **Instance** | **Size (# of jobs)** | **Time (sec.)** |
| 1 | 4 | 0,211 |
| 2 | 6 | 0,592 |
| 3 | 8 | 8,377 |
| 4 | 9 | 69,67 |
| 5 | 10 | 284 |
| 6 | 12 | >3600 |

**Table 2.** Computational execution times of the MILP model adapted for urgent jobs

|  |  |  |
| --- | --- | --- |
| **Instance** | **Size (# of jobs)** | **Time (sec.)** |
| 1 | 4 | 0,211 |
| 2 | 6 | 0,432 |
| 3 | 8 | 7,542 |
| 4 | 9 | 59,118 |
| 5 | 10 | 298,772 |
| 6 | 12 | >3600 |

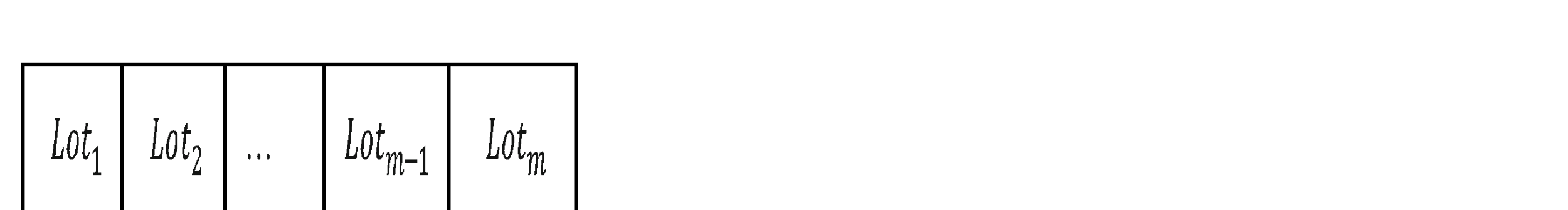
Así, para aprovechar la rapidez de ejecución en problemas pequeños, se busca **descomponer el problema formando lotes que contengan entre 3 y 6 trabajos**. Para llegar a esta idea, inspirada en [2], se realizó una partición del problema global dividiéndolo en subproblemas de **tamaño igual**.

Por otro lado, en nuestro caso, esto solo sería posible si el número de trabajos del problema es un **múltiplo del número de elementos** que se define como tamaño de lote. Entonces, sabiendo que los tamaños serán diferentes para cada instancia evaluada, hemos definido **dos parámetros** que son utilizados por el usuario para dividir el problema, de forma que **siempre se obtenga una división de la lista formando grupos de entre 3 y 6 trabajos**.

En la **Figura 2** se puede apreciar la **secuencia de pasos** seguida en esta fase tanto para **ordenar como para descomponer** un problema de planificación dado para un conjunto de **n piezas**. En el lado derecho de la figura, los **círculos de colores** representan diferentes trabajos e ilustran **cómo se reordenan y agrupan** después de cada paso realizado.

**Fig. 2. Secuencia de pasos seguidos para ordenar y descomponer el problema**  
Como se muestra en la **Figura 3**, para resolver el problema MILP para cada uno de los **m lotes** en los que se descompuso el lote inicial de piezas, se ha definido un **tamaño de lote igual a b** para la mayoría de los grupos. Esta descomposición se realizó con el objetivo de **uniformar la distribución de trabajos entre los lotes** y así lograr, en lo posible, **tiempos de ejecución similares para todos los subproblemas**.

Basándonos en resultados previos del problema estático, se esperaba **emular el rendimiento de los problemas pequeños**, lo cual nos llevó a concluir que el **valor de b debería estar comprendido entre 3 y 6**.



Además, una vez que se han creado todos los lotes, incluso el **último subproblema debe ser resoluble** mediante el modelo lineal con un **tiempo de ejecución similar** al de los subproblemas anteriores; por lo tanto, el **último subproblema generado también debe contener entre 3 y 6 trabajos** (resto).

Dicho esto, **aprovechar la rapidez del modelo original en instancias pequeñas no fue la única razón** que respalda la decisión de elegir lotes con tamaños comprendidos en este rango.

De hecho, si consideramos la **capacidad máxima de la célula de producción de manera simultánea**, podemos ver fácilmente que con **2 o menos trabajos** no tendría sentido usar un método de resolución prolongado para resolver el problema.

Así, asumiendo que **(n - resto)** trabajos deberán agruparse, el **tomador de decisiones podrá establecer el valor de b** para obtener el **número de lotes (m)** utilizando la expresión mostrada en la **Ecuación (35)**.

### Paso 3: Concatenación de lotes de trabajos

Como resultado del paso anterior, se obtendrán **m lotes** para los cuales se llamará al **solucionador de programación lineal (LP)** de forma **iterativa**, **encadenando los resultados** obtenidos de un lote como **entradas para el siguiente subproblema**. Cada lote será etiquetado con un subíndice **k**, el cual también definirá el número de llamadas al modelo lineal hasta ese momento (**número de iteraciones**).

Sabiendo que **cada lote de trabajos se procesará en el orden de su índice**, después de cada ejecución y obtención de resultados del modelo lineal, deberemos **concatenar los resultados finales con las condiciones iniciales del siguiente lote**.

Para ello, es necesario **modificar el modelo lineal básico** agregando **tiempos de disponibilidad distintos de cero** para los lotes desde el **segundo hasta el m-ésimo**. Con este fin, se añadió un **vector que contiene el parámetro rⱼ** para cada trabajo **j** perteneciente al lote actual.

Estas **fechas de disponibilidad** se importarán directamente desde la solución obtenida con el lote anterior; es decir, las fechas de disponibilidad de cada trabajo en el nuevo problema lineal a resolver serán **iguales al tiempo de finalización del trabajo en el problema anterior**.

El **esquema general** de la concatenación propuesta se muestra en la **Figura 4**, y la **función objetivo resultante** será entonces obtenida a partir del **makespan correspondiente al último lote**.