**SOLUCIONES DEL PROBLEMA DE BALANCEO DE LÍNEA SALBP-2. UN ENFOQUE BOOLEANO CON PROGRAMACIÓN EN PARALELO.**

**RESUMEN**

Se presenta un algoritmo para encontrar todas las soluciones del problema de Balanceo de Línea SALBP-2. El algoritmo parte de un ciclo ideal y de la generación de restricciones booleanas para este ciclo. Las restricciones son manipuladas lógicamente, hasta que se encuentra el espacio de soluciones. Si el espacio booleano generado es vacío, se incrementa el ciclo ideal con el máximo común divisor y se repite el ciclo, hasta hallar todas las soluciones con el ciclo óptimo. Además de encontrarlas, el algoritmo puede paralelizarse con relativa facilidad, para ejecutarse en ambientes CUDA C. En este trabajo, se describe en forma general el algoritmo y su implantación en la nube en AWS.

**ÍNDICE**

**1. Definición del Problema**

1.1 Línea de Ensamble 3

1.2 Una Definición del Problema Balanceo de Línea de Ensamble Simple 3

1.3 Ejemplo para Ilustrar el Algoritmo Propuesto 4

**2. Algoritmo Propuesto**

2.1 Fundamentos 6

2.2 Uso del Algoritmo con el Ejemplo Mostrado 7

2.2.1 Ciclo Ideal y Máximo Común Divisor 7

2.2.2 Tabla de Antecedentes y Consecuentes 7

2.2.3 Conversión de Restricciones Binarias en Restricciones Booleanas 8

2.2.3.1 Restricciones de Duración 9

2.2.3.2 Restricciones de Precedencia 11

2.2.3.2 Restricciones de Unicidad 12

2.2.4 Negación de las Restricciones Booleanas y Generación del Espacio Factible 12

2.2.5 Diagrama del Algoritmo 15

**3. Implantación en Paralelo**

3.1 Implantación en Computadoras con Procesadores en Paralelo en la Nube 16

3.2 Descripción 16

3.2.1 Procesamiento de las Entradas 16

3.2.2 Proceso de Simplificación 16

3.2.3 Generador de Restricciones 17

3.2.4 Mezclador Binario Secuencial 17

3.2.5 Mezclador Binario en Paralelo 17

3.3 Implantación en la Nube 18

**REFERENCIAS 19**

**1 DEFINICIÓN DEL PROBLEMA**

**1.1 Línea de Ensamble**

Un sistema de ensamble tiene como objetivo el ensamble de algún producto o productos mediante la realización de unidades indivisibles de trabajo, llamadas *actividades*.

Las actividades se realizan en *estaciones de trabajo*. El sistema está formado por un mecanismo de transporte que une a las estaciones de trabajo; además, se especifica el orden de las estaciones y de las actividades que debe seguir el ensamble del producto. Se conoce como *línea de ensamble* al conjunto de estaciones y al mecanismo que transporta el ensamble.

Se llama *tiempo de ciclo w* al intervalo de tiempo que transcurre antes de que el mecanismo de transporte se mueva. Se denomina como *tiempo de procesamiento* al tiempo necesario para realizar una actividad. El *contenido de trabajo o duración* de una estación se obtiene sumando todos los tiempos de procesamiento de las actividades que se realizan en esa estación. Para evitar retrasos en la línea, el contenido de trabajo de las estaciones debe ser menor que el intervalo de tiempo en el cual se mueve el mecanismo de transporte.

Se le llama *holgura* al tiempo de ocio con el que cuenta cada estación, i.e., el tiempo de ciclo menos su duración. Si la holgura de cada una de las estaciones es mayor que cero entonces una unidad se termina de ensamblar cada tiempo *w*, es decir, la tasa de producción de la línea es igual a 1/*w*. Como ya se estableció, el orden en el cual se realizan las actividades está especificado y se llaman *relaciones de precedencia* a la especificación de este orden. Debido a estas relaciones de precedencia, no se puede comenzar una actividad hasta que todos sus predecesores inmediatos hayan sido terminados. Se pueden representar gráficamente estas relaciones a través de un *grafo de precedencias*. En este trabajo, se supone que la línea de ensamble no tiene reprocesos o retroalimentaciones, por lo tanto, el grafo que se obtiene no es cíclico.

Si cada nodo del grafo simboliza a una actividad entonces, para representar una relación de precedencia inmediata de una actividad *i* a una actividad *j*, se utiliza un arco dirigido desde el nodo *i* hacia el nodo *j*.

Si la suma de las holguras de todas las estaciones es la mínima posible, entonces la línea de ensamble se encuentra *balanceada* [Baybars, 1986].

**1.2 Una Definición del Problema Balanceo de Línea de Ensamble Simple**

En términos generales el problema de balanceo de línea (SALBP por sus siglas en inglés) consiste en asignar las actividades en la línea de ensamble a un conjunto de estaciones, optimizando alguna variable objetivo, sin violar ciertas restricciones. Dependiendo del objetivo de optimización y de las restricciones, se derivan varios modelos de SALBP [Baybars, 1986].

A continuación se enumeran los supuestos que el mismo autor establece, sobre los cuales se basa la versión simple de SALBP:

S-1 Todos los parámetros de entrada se conocen con certeza.

S-2 Una actividad no se puede dividir entre dos o más estaciones.

S-3 Las actividades no pueden ser procesadas en secuencias arbitrarias debido a ciertos requerimientos tecnológicos de precedencia.

S-4 Todas las actividades deben ser procesadas.

S-5 Todas las estaciones en consideración están equipadas, en cuanto a máquinas y trabajadores, para procesar cualquiera de las actividades.

S-6 Los tiempos de procesamiento de las actividades son independientes de la estación en la que se llevan a cabo o de sus predecesoras o sucesoras.

S-7 Cualquier actividad puede ser procesada en cualquier estación.

S-8 La línea es considerada en su totalidad como serial, es decir, no existen líneas de ensamble secundarias o líneas paralelas de sub-ensamble.

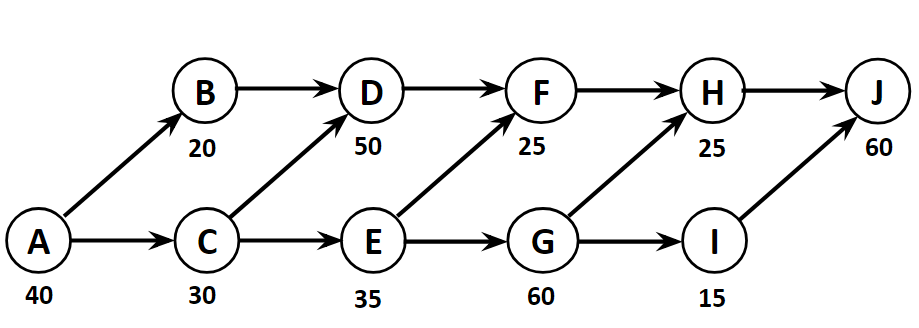
S-9 Se asume que el sistema de ensamble está designado para un único modelo de un único producto.

El objetivo del tipo general de SALBP, SALBP-G, es buscar el número de estaciones *m* y el tiempo de ciclo *c* que minimizan la holgura total del sistema. El problema SALBP-G es de tipo no lineal, por lo que surgen dos versiones lineales simplificadas de ese problema. La versión SALBP-1 consiste en minimizar el número de estaciones *m* dado un tiempo de ciclo *w*, en tanto que la versión **SALBP-2 consiste en minimizar el tiempo de ciclo *w* dado un número de estaciones *m*.**

El procedimiento aquí propuesto se centra en solucionar la versión SALBP-2. En específico, se busca una asignación de las actividades a las estaciones de trabajo de tal manera que se minimice el tiempo de ciclo *w*.

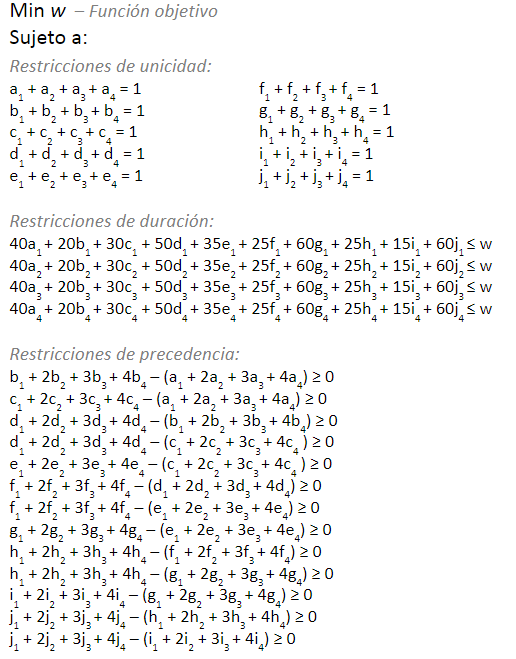
**1.3 Ejemplo para Ilustrar el Algoritmo Propuesto**

Para ilustrar el método aquí propuesto, se usará la siguiente línea de producción:



Esta línea tiene 10 actividades, un tiempo total de 360 minutos y se quiere encontrar el ciclo óptimo con cuatro estaciones de trabajo.

Usando Programación Lineal Mixta, este problema podría ser resuelto mediante la siguiente formulación:



Todas las variables del modelo, con excepción de w, que representa al ciclo, son binarias. La notación xi implica que la actividad x se efectúa en la estación i. Por ejemplo, dentro de las restricciones de unicidad, una para cada actividad, denotadas por las letras **a** hasta la **i**, queda implícito que cada actividad solamente puede realizarse en una y solo una de las cuatro estaciones de trabajo.

El segundo grupo de restricciones son las llamadas restricciones de duración. Si todas las actividades se realizaran en la estación uno, la duración total para esa estación sería 360 minutos y las otras estaciones no tendrían actividades ni duraciones. Este sería el peor caso posible, pues la holgura sería de 360.

El tercer grupo de restricciones define las precedencias en la ejecución de las actividades. Por ejemplo, la actividad b tiene que realizarse después de que se haya ejecutado la actividad a. Por ello, si b1 fuera 1, se forzaría a que a1 también lo fuera.

La función objetivo es una del tipo min-max. Busca minimizar el máximo ciclo posible apoyándose en las restricciones de duración. Sin restricciones, la holgura se minimizaría con un ciclo de 90 minutos, ya que ninguna estación de trabajo tendría holgura.

**2 ALGORITMO PROPUESTO**

**2.1 Fundamentos**

El algoritmo que se propone no resuelve el problema utilizando técnicas de Programación Matemática sino usando conceptos básicos de Álgebra Booleana. En breve, el algoritmo primero asume que existe una solución con un ciclo idealmente pequeño; con éste, reduce el tamaño del problema; luego transforma las variables binarias en variables booleanas y genera el conjunto de todas las restricciones que hacen que el problema no sea factible. A partir de la negación booleana de este conjunto se obtienen dos resultados: el primero, un conjunto vacío, si el ciclo no es factible, por lo que se aumenta el ciclo inicial con el máximo común de la duración de las actividades y se repite el algoritmo, o, el segundo, todas las soluciones posibles que existen para el ciclo elegido, en cuyo caso, el algoritmo termina su ejecución.

Las ventajas que presenta este algoritmo son que genera todas las soluciones óptimas posibles para un problema dado y, lo más importante, que el grueso de los cálculos computacionales se puede implantar en una computadora que tenga procesadores en paralelo. La primera ventaja permite elegir una asignación de actividades a las estaciones que pudiera ser *primera entre pares*. La segunda, es que, en principio, pudieran encontrarse soluciones para problemas relativamente grandes.

**2.2 Uso del Algoritmo con el Ejemplo Mostrado**

2.2.1 Ciclo Ideal y Máximo Común Divisor

En el ejemplo mostrado en el capítulo anterior, la duración total de realizar todas las actividades es de 360 minutos. Entonces, si se quiere obtener el óptimo para cuatro estaciones, la mínima holgura ideal sería cero, por lo que la duración de cada estación de trabajo sería de 90 minutos y, por consiguiente, el *ciclo ideal* es de 90 minutos. Otra información inicial que se obtiene fácilmente de la red mostrada es el Máximo Común Divisor (MCD) que para esta red es de 5 minutos.

2.2.2 Tabla de Antecedentes y Consecuentes

Para la red utilizada como ejemplo y con un ciclo de 90 minutos es posible construir la tabla siguiente:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Actividad** | **Duración** | **Tiempo  anterior** | **Tiempo posterior** | **Estaciones posibles con tiempo  anterior** | **Estaciones posibles con tiempo posterior** | **Intersección** |
| **A** | 40 |  |  |  |  | 1 |
| **B** | 20 | 60 | 180 | 1,2,3,4 | 1,2,3 | 1,2,3 |
| **C** | 30 | 70 | 300 | 1,2,3,4 | 1 | 1 |
| **D** | 50 | 140 | 160 | 2,3,4 | 1,2,3 | 2,3 |
| **E** | 35 | 105 | 220 | 2,3,4 | 1,2 | 2 |
| **F** | 25 | 200 | 110 | 3,4 | 1,2,3 | 3 |
| **G** | 60 | 165 | 160 | 2,3,4 | 1,2,3 | 2,3 |
| **H** | 25 | 285 | 85 | 4 | 1,2,3,4 | 4 |
| **I** | 15 | 180 | 75 | 2,3,4 | 1,2,3,4 | 2,3,4 |
| **J** | 60 |  |  |  |  | 4 |

En la tabla, en cada renglón se muestran las actividades de la línea de producción usada como ejemplo. Para cada una de ellas, en la segunda columna se muestra su duración. La primera actividad de la red, la actividad **a**, tiene que efectuarse en la primera estación de trabajo, ya que precede a todas las demás actividades; siguiendo el mismo razonamiento, la última actividad, la actividad **j**, tiene que realizarse en la última estación de trabajo, ya que es precedida por todas las demás actividades.

Para las actividades intermedias entre la primera y la última, en la tercera columna se muestra el tiempo que la actividad toma en realizarse más los tiempos de las actividades que la preceden y que dependen directamente de ella. Por ejemplo, para la actividad **b**, su duración es de 20 minutos y solamente la precede la actividad **a**, que dura 40 minutos. En la cuarta columna se muestra, para cada actividad, un concepto similar, pero registrando los tiempos de las actividades mismas más los tiempos de las actividades que directamente las suceden. Por ejemplo, la actividad **b** tiene una duración de 20 minutos y es sucedida directamente por las actividades **d**, **f**, **h** y **j**, que tienen duraciones respectivas de 50, 25, 25 y 60 minutos; la suma de estas duraciones es de 180 minutos.

Ahora bien, como ya se expresó, la duración de las actividades de cada estación es de 90 minutos, entonces, si se toma como ejemplo a la actividad **b**, que tiene como tiempo anterior 60 minutos, esto implica que podría realizarse en cualesquiera de las estaciones de trabajo; sin embargo, con un tiempo posterior de 180 minutos, no podría realizarse en la última estación de trabajo y solamente podría efectuarse en las tres primeras estaciones. Esta información se presenta en la quinta y sexta columna de la tabla mostrada.

En la séptima columna se muestra la intersección de la información de las dos columnas anteriores. Este método permite, para un ciclo dado, simplificar el modelo de programación matemática presentado en el capítulo anterior. En el caso en cuestión, se eliminan seis variables, pues a1 = c1 = e2 = f3 = h4 = j4 = 1.

2.2.3 Conversión de Restricciones Binarias en Restricciones Booleanas

Usando la información de la tabla anterior, fácilmente se puede encontrar que las restricciones del modelo se simplifican a:

*Restricciones de unicidad:*

b1 + b2 + b3 = 1

d2 + d3 = 1

g2 + g3 = 1

i2 + i3 + i4 = 1

*Restricciones de duración:*

20b1 + 70 ≤ 90

20b2 + 50d2 + 60g2 + 15i2 + 35 ≤ 90

20b3 + 50d3 + 60g3 + 15i3 + 25 ≤ 90

15i4 + 85 ≤ 90

*Restricciones de precedencia:*

2d2 + 3d3 – (b1 + 2b2 + 3b3) ≥ 0

2i2 + 3i3 + 4i4 – (2g2 + 3g3) ≥ 0

Ahora se procede a convertir estas restricciones binarias en restricciones booleanas. Conviene empezar con las restricciones de duración, pues su tratamiento permite simplificar aún más el modelo. (No se mostrará aquí el método de simplificación, pues en este ejemplo lo que interesa es el tratamiento booleano.)

2.2.3.1 Restricciones de Duración

La conversión de las restricciones de duración es un problema combinatorio, pero puede ser simplificado mediante el método que se presenta enseguida. Se inicia con la segunda restricción de duración:

20b2 + 50d2 + 60g2 + 15i2 + 35 ≤ 90

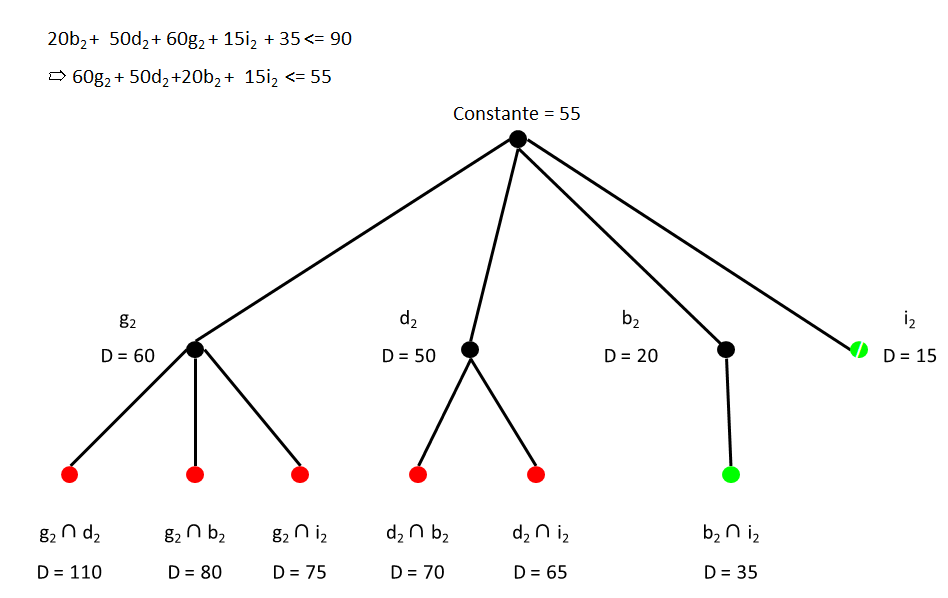
Primero, se ordena la ecuación de acuerdo a la magnitud de los coeficientes de las variables binarias:

60g2 + 50d2 +20b2 + 15i2 ≤ 55

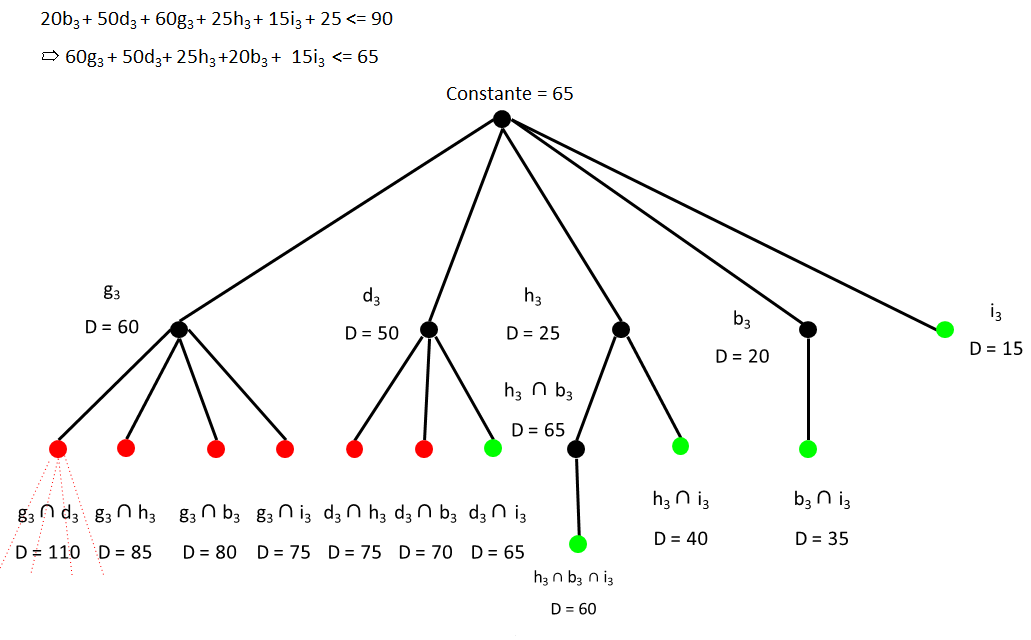
En segundo, se construye un árbol TOP-DOWN (depth-first) con un nodo para cada variable. Cada nodo tiene como nodos hijos las variables de coeficientes menores, es decir, las que le siguen en la lista ordenada en forma descendente.

En tercero, en cada nodo se evalúa la duración acumulada de los nodos hasta llegar a la raíz. Si la duración acumulada es mayor que la restricción, se crea una ecuación booleana de imposibilidad de duración. Una vez que se llega a esta situación, se detiene la expansión de ese nodo.

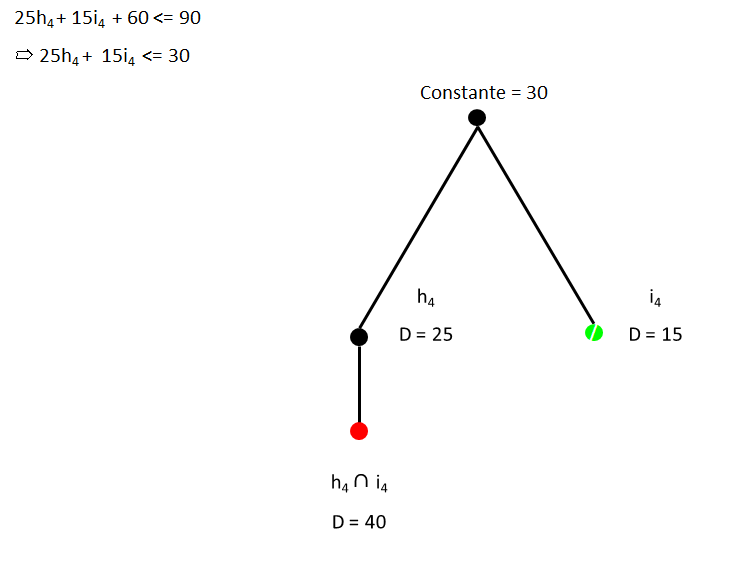
En la gráfica siguiente se ilustra el método.



Para la tercera ecuación de duración el árbol quedaría como se muestra enseguida:



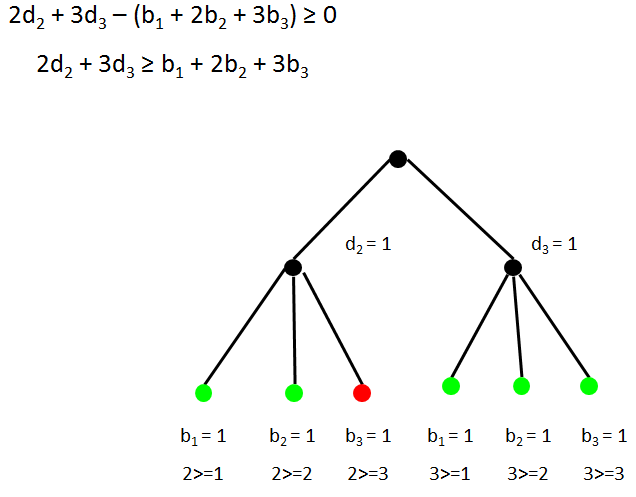
Y para la cuarta restricción de duración queda como:

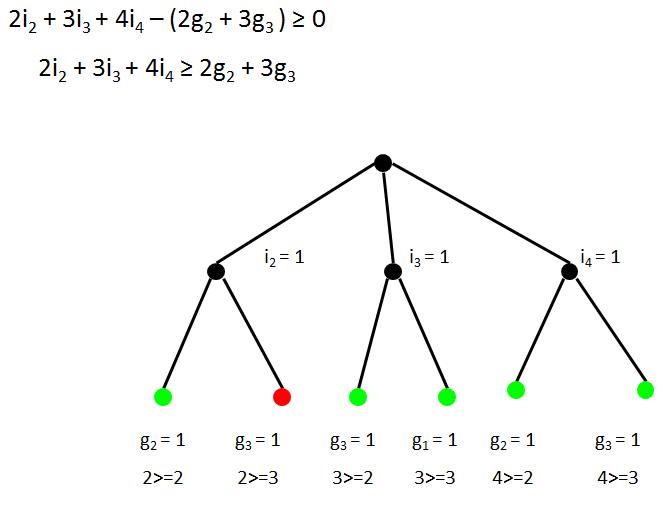


La primera restricción no genera restricciones binarias.

2.2.3.2 Restricciones de Precedencia

Para las restricciones de precedencia solamente es necesario registrar las asignaciones de las variables binarias que generan una violación en la desigualdad. Esto se ilustra con las siguientes gráficas:





Las cuales generan las ecuaciones booleanas d2 ∩ b3 = 0 e i2 ∩ g3 = 0.

2.2.3.2 Restricciones de Unicidad

Para estas restricciones no es necesario generar restricciones booleanas, pues se pueden ir tratando una a una en el proceso que se muestra más adelante.

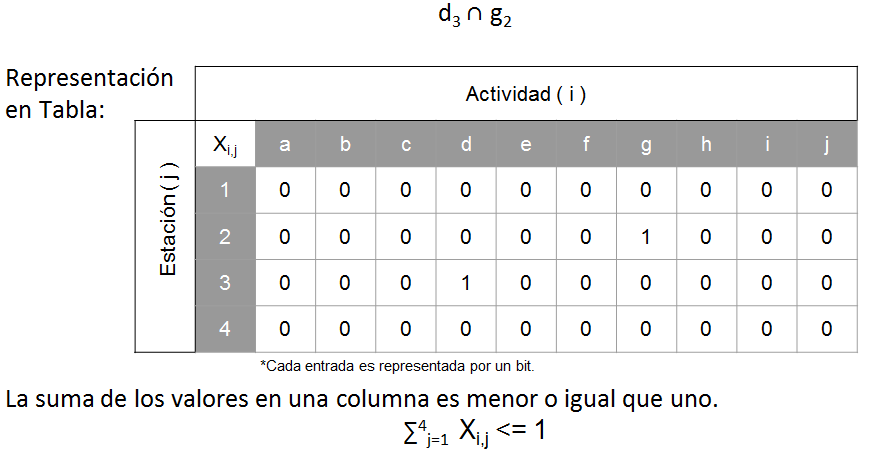
2.2.4 Negación de las Restricciones Booleanas y Generación del Espacio Factible

Nótese que el modelo no es viable si cualquiera de las ecuaciones booleanas generadas está operando. Pero esto se puede extender a que el modelo no es viable si la primera ecuación booleana no es viable, o si la segunda no es viable, o si la tercera no lo es, etc. Entonces, usando un teorema básico de Álgebra Booleana, si se niega una unión de expresiones booleanas nulas, el espacio generado por la intersección de la negación de cada una de las expresiones booleanas es un espacio factible.

Por ejemplo, si g3 ∩ h3 = 0, su negación sería g2 ∪ h4 = 1. Ahora solamente faltaría hacer la intersección binaria de todas las expresiones booleanas negadas. Por ejemplo, la intersección de las expresiones (g2 ∪ h4) ∩ (g3 ∪ d3)= 1 sería:

(g2 ∩ d3) ∪ (h4 ∩g3) ∪ (h4 ∩ d3) = 1

Ya que la intersección de g2 y g3 es nula. La intersección genera tres uniones de la intersección de dos variables. Para efectuar esta operación computacionalmente se generan tres tablas como la que se muestra a continuación:



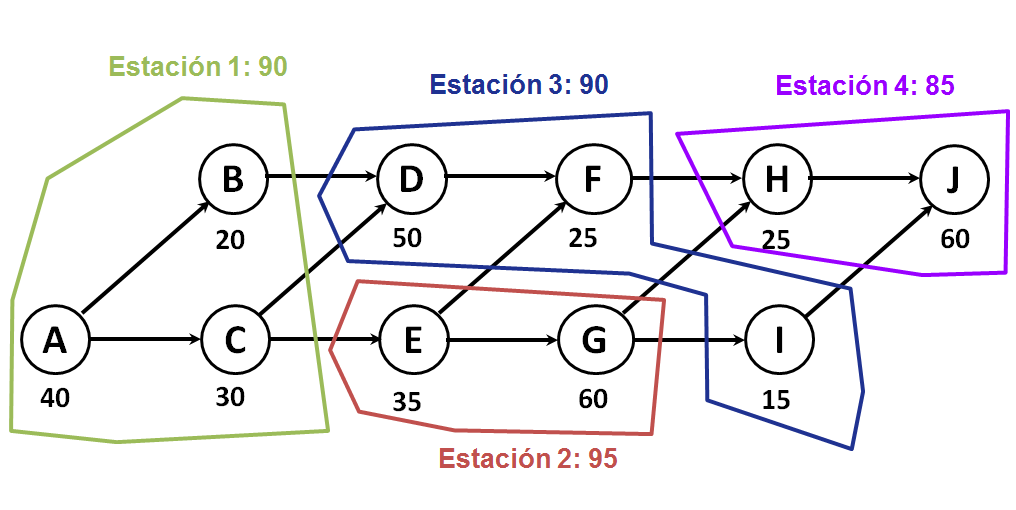
La tabla puede eliminarse si existen dos o más variables en la misma columna o si la tabla ya existe. Las tablas son modificadas con la intersección de las otras expresiones booleanas negadas, hasta terminar con todas ellas. El número de tablas crece conforme se van intersectando más restricciones; sin embargo, dadas las restricciones de unicidad, este número alcanza un máximo que luego decrece hasta encontrar el óptimo del problema o encontrar un conjunto vacío. Si este es el caso, entonces se aumenta el ciclo ideal con el Máximo Común Divisor y se repite el algoritmo.

En las tablas siguientes se muestra cómo crecen el número de tablas para el ejemplo utilizado con un ciclo de 90 minutos, que no es solución, y para un ciclo de 95 minutos, que resuelve el problema.

La solución del problema se muestra en la tercera gráfica.

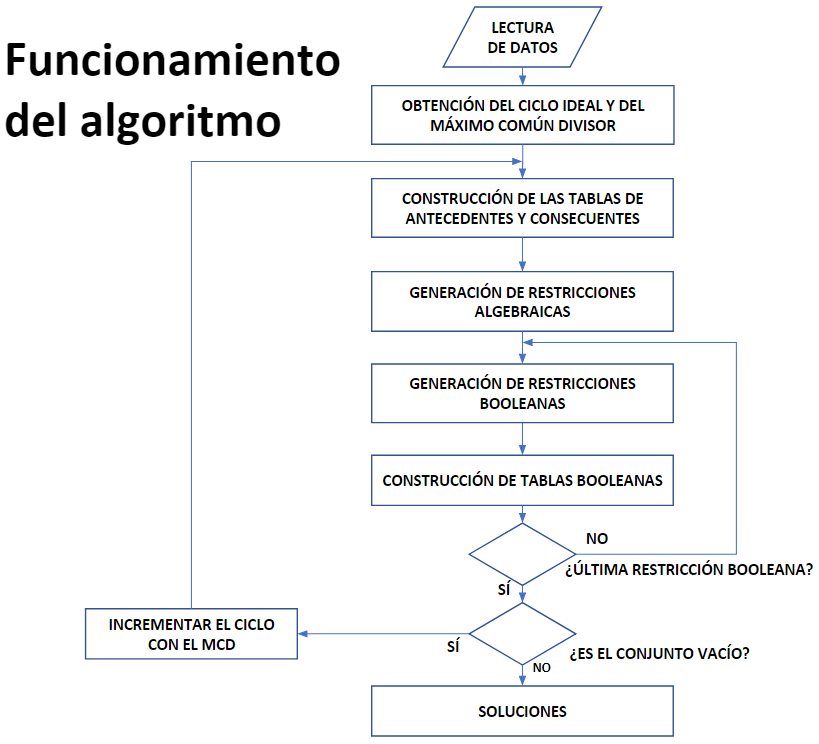






2.2.5 Diagrama del Algoritmo

El algoritmo general se puede mostrar el diagrama siguiente:



**3. Implantación en Paralelo**

3.1 Implantación en Computadoras con Procesadores en Paralelo en la Nube

El algoritmo fue implantado exitosamente en un procesador secuencial, con la desventaja de que el número de tablas que se generan puede ser inmenso para problemas grandes de balanceo de línea. Dado que el grueso de las labores de cómputo está en el manejo de las tablas de restricciones booleanas, se programó en una computadora en paralelo, también obteniendo resultados satisfactorios.

En este proyecto, se reelaboró el código para poder implantar el algoritmo en la nube. Lo que sigue en adelante está expresado en lenguaje técnico.

3.2 Descripción

El algoritmo del método está programado en C++ bajo el paradigma de Programación Orientada a Objetos. A partir de un archivo de texto que contiene la línea de producción, las duraciones de las actividades y sus precedencias, regresa todas las matrices que denotan las posibles soluciones óptimas. Específicamente, el kernel de CUDA, mediante el cual se lleva a cabo el mezclador en paralelo está progamado en C.

3.2.1 Procesamiento de las Entradas

A partir de la lectura del input, se crea una instancia del grafo de actividades, la línea de producción y una matriz con relaciones de precedencia.

Luego se obtiene el tiempo del ciclo ideal, dividiendo el tiempo total de ejecución de las actividades entre el número de estaciones deseadas. Se calcula el máximo común divisor para las duraciones de las actividades.

3.2.2 Proceso de Simplificación

Mediante el objeto *Simplifier*, se eliminan las restricciones redundantes tomando en cuenta los tiempos antecedentes y los consecuentes de cada una de las actividades. Este objeto regresa otro objeto, denominado *Allcandidatestations*, que contiene para cada estación sus posibles actividades candidatas. Si existe una estación sin actividades o actividades sin estación, se incrementa el ciclo ideal con el máximo común divisor y se repite el *Simplifier*.

3.2.3 Generador de Restricciones

El problema tiene tres tipos de restricciones algebraicas. El objeto *Restrictiongenerator* , a partir de *Allcandidatestations*, se encarga de generar las restricciones de duración, precedencia y unicidad. Estas restricciones se empaquetan en un objeto llamado *Model*, que las guarda en memoria.

Desde *Model*, se convierten las restricciones algebraicas mencionadas anteriormente en restricciones booleanas. A través del método *Getnegation*, se obtienen las restricciones negadas. No es necesario negar las restricciones de unicidad.

3.2.4 Mezclador Binario Secuencial

El conjunto de restricciones negadas se representa en tablas binarias las cuales se procesan a través de un objeto llamado *Binarymixer.* Este objeto, a partir de un proceso que realiza iteraciones sobre las tablas binarias obtenidas en las restricciones, genera un vector de matrices con las asignaciones óptimas para el balanceo de línea.

El proceso ocurre de la siguiente forma: Primero, se instancia un vector de tablas binarias, el cual se denomina vector de soluciones temporales. Se inserta la tabla binaria correspondiente a la primera restricción de duración en el mismo. A partir de un ciclo for, el vector de soluciones temporales realiza la operación booleana ‘and’ con cada uno de las tablas generadas de las restricciones de duración. Antes de insertar la tabla como asignación posible al vector de soluciones temporales, se verifica que ésta sea válida y no haya sido insertada previamente en el vector. Este último proceso se lleva a cabo mediante una tabla hash. Posteriormente se toma el vector de soluciones temporales para repetir el proceso con las restricciones de precedencia y las de unicidad. Una vez que este proceso termina, se verifica la cantidad de tablas en el vector de soluciones temporales. Si no existe al menos una solución, se le incrementa el máximo común divisor al ciclo ideal y se repite el algoritmo. Por otra parte, si el vector de soluciones temporales contiene tablas, estas últimas constituyen las soluciones óptimas.

Vale la pena mencionar que conforme se van añadiendo las restricciones, el número de tablas crece y luego se reduce al incluir más restricciones.

3.2.5 Mezclador Binario en Paralelo

Primero se ejecuta la función Runtest, la cual instancia apuntadores de cuatro gigas para las restricciones booleanas y 1GB para el vector de soluciones temporales, estos apuntadores son copiados a la GPU. Se calculan los threads necesarios, los cuales son tantos como tablas existan en el vector de soluciones temporales. Previo a la invocación del kernel, se calcula el tamaño posible que se puede asignar en el sistema de la GPU tratando de asignar el mínimo entre 4GB/entre el número de tablas o 500 MB. Si no se puede asignar el espacio, se reduce en un MB y se vuelve a intentar hasta que se pueda asignar. A partir del tamaño que quedó sin alocar del vector de soluciones temporales se lleva a cabo un sistema de batches mediante el cual se invoca al kernel para que procese tantos threads/tablas del batch en cuestión. Este proceso, al igual que en el método secuencial, se lleva a cabo con todas las tablas de los 3 tipos de restricciones booleanas. La verificación de que las tablas adosadas al vector de soluciones temporales sean válidas se realiza dentro del device/gpu, no así, la verificación de que éstas no estén duplicadas, proceso que se lleva a cabo en el host. Una vez terminado el proceso, el flujo es análogo al método secuencial. En caso de que el vector de soluciones temporales contenga tablas, estas constituyen las asignaciones óptimas. De otra forma, se incrementa el máximo común divisor y se repite el proceso.

Éste se encarga de procesar en paralelo las tablas de restricciones. Para problemas grandes, si el GPU se queda sin memoria, el resultado temporal se copia al Host para liberar memoria y se prosigue con el proceso.

**3.3 Implantación en la Nube**

Para implantar el método en la nube, se utilizó un servidor Ubuntu, con GPU Tesla K8, en una instancia tipo p2.xlarge, con cuatro núcleos, 61 Gigas de memoria. Ya instalado tenía el Cuda Toolkit 9.

**REFERENCIAS**

Baybars, İlker. «A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem.» Management Science 32, nº 8 [August 1986]: 909-932.

Presentación del Algoritmo

[https://docs.google.com/presentation/d/1QFajvdVtlx-0lgN-7gol41Sk\_bMVp9cwIbEXb-fNUUM/edit?ts=5aa965fd#slide=id.p3](https://docs.google.com/presentation/d/1QFajvdVtlx-0lgN-7gol41Sk_bMVp9cwIbEXb-fNUUM/edit?ts=5aa965fd" \l "slide=id.p3)