1. Información de entrada

Para que nuestra heurística funcione, es necesario contar con cierta información inicial. Sean:

Relacionados con los strokes

N_{ik}	Matriz de inputs. Es decir, el número de unidades del producto i que requiere el stroke k para ser ejecutado.
M_{ik}	Matriz de outputs. Es decir, el número de unidades de producto i que genera el stroke k luego de ser ejecutado. En nuestro problema representamos un sistema en el que cada stroke puede generar sólo un producto, de no ser así estaríamos hablando de coproducción.
LT_k	Vector de lead-time del stroke k.
SC_k	Vector de costo de preparar el stroke k.
OC_k	Vector de costo de operación del stroke k.

Relacionados con los productos

D_{it}	Matriz de proyección de la demanda. Unidades del producto i requeridas en el tiempo t.
R_{it}	Matriz de recepciones programadas. Unidades del producto i que llegarán en el tiempo t.
$IInic_i$	Vector de inventario inicial del producto i.
HP	El horizonte de planeación, medido en unidades de tiempo.
FOQ_i	Vector de cantidad fija a ordenar del producto i.
HC_i	Vector de costo de almacenar una unidad del producto i en inventario.
SOC_i	Vector de costo por falta en inventario de una unidad del producto i.

2. Descripción de la heurística

Sea $SS_{kk'}$ la matriz que representa las relaciones de input y output entre strokes, y que está determinada por $SS_{kk'} = {N_{ik}}^{-1} \times M_{ik}$. Sea también $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ el grafo que muestra las relaciones de input y output strokes, construida a partir de $SS_{kk'}$ con la propiedad adicional de modelar de manera clara la relación AND u OR entre un conjunto de strokes y el producto que generan (Línea 2, Algoritmo 1).

La heurística resuelve el problema de determinar cuáles productos serán generados a partir de cuáles strokes y en qué proporciones. Para esto es necesario conocer de cuántas formas puede ser generado el mismo producto y estimar cuál es el costo acumulado de cada opción (Líneas 4 y 5, Algoritmo 1), la forma en que se realiza este cálculo se explicará más adelante. Además se requiere ordenar la lista de productos en caso de que si algún producto contiene entre sus 'hijos' a otro producto en el mismo nivel, se ejecute este primero (Línea 7, Algoritmo 1). Para definir de qué forma se atenderán los requerimientos de los productos – lo que determinaría la forma en que se distribuirán las llamadas a

los strokes – se calcula el MRP de cada producto en la lista utilizando alguna regla de lotificación que se puede definir como dato de entrada del sistema (Línea 10, Algoritmo 1).

Luego de determinar el mejor stroke para generar un producto i en el periodo t y haber decidido el momento en el que será ejecutado y cuántas veces, se agrega a la lista de productos los requerimientos del stroke elegido y se iguala a cero el requerimiento actual para hacerle entender al programa que se ha cumplido con el requerimiento de ese producto i en el periodo t (Líneas 12,13 y 14, Algoritmo 1). A continuación se muestra en pseudocódigo el algoritmo que rige la heurística.

Algoritmo 1. Visión global de la heurística

```
1   HeuristicaSimple(OC_k , SC_k , HC_i , N_ik , M_ik, D_it, R_it, IInic_i)
2       SS_kk, = producto matricial entre N_ik y M_ik
3       Sea G = (N, A) el grafo que representa las relaciones de input y output entre productos y strokes
4       accumOC_k = accumOC(G(N, A), OC_k) estimación del costo acumulado de operación
5       accumSC_k = accumSC(G(N, A), SC_k) estimación del costo acumulado de preparación
6       function Recursive (listaProductos, strokeMatrix)
7           ordenarListadeProductos(listaProductos)
8           For(int i=0;i<listaProductos.size();i++)
9               For(int t=0;t<longitudPeriodo;t++)
10                  listaProductos(i).mps = obtenerMPSde(i, D_it, R_it, IInic_i)
11                  If(listaProductos(i).mps(t) > 0)
12                      Stroke = seleccionarStroke(i,t, listaProductos(i).mps(t))
13                      strokeMatrix(stroke.key,stroke.momentoEjecutar) += stroke.vecesQueEjecuta
14                      listaProductos(i).mps(t) = 0
15                  End If
16              End For
17              For(int i=0;i<listaProductos.size();i++)
18                  If(listaProductos(i).noTieneRequerimientos() and listaProductos.size() > 1 )
19                      listaProductos.remover(i)
20                      i=-1;
21                  End If
22                  Elsif (listaProductos.(i).noTieneRequerimientos() and listaProductos.size()==1)
23                      return strokeMatrix
24                  End Elsif
25              End For
26              return Recursive(listaProducto, strokeMatrix)
27          End For
28          return strokeMatrix
29      End Recursive
30  End HeuristicaSimple
```

Como se había mencionado, existen dos funciones en el Algoritmo 1 que merecen atención: `accumOC()` y `accumSC()` las cuales realizan la estimación de los costos acumulados de Operación y *Setup*, respectivamente, de cada opción disponible, refiriéndonos por opción a los caminos que posiblemente se puedan encontrar hacia abajo en el grafo de tomar un stroke de los disponibles. El algoritmo inicia con los strokes que generan un producto final y de ahí hacia abajo a sus componentes. Como se observa en la línea siguiente se ilustra el algoritmo, que es recursivo:

$$OCacum_k = OC_k + \sum OCacum_a + \frac{\sum OCacum_o}{O} \qquad \forall_k$$

Sean $A = card(a)$ el número de 'hijos' en condición AND del stroke k, y $O = card(o)$ los que estén en condición de OR.

En la línea 10 de Algoritmo 1 se llama una función, `obtenerMPSde`, que calcula el MPS de cada componente, los requerimientos que se le pasan como parámetros a la función pueden provenir de la demanda o de la explosión de su correspondiente stroke 'padre'. Aparte de esto es una función típica de cálculo de MPS dentro de la que el usuario tiene la posibilidad de embeber distintas reglas de lotificación. Este MPS se calcula a cada producto, de forma independiente a los strokes.

3. Ejemplo de aplicación:

La finalidad del grafo es permitir tener una visión general de la estructura de los productos y es una herramienta fundamental que permitir el cálculo de los costos acumulados de cada stroke, por las funciones `accumOC()` y `accumSC()`.

A continuación se presenta un ejemplo de 6 productos ($card(i) = 6$) y que son elaborados con 9 strokes ($card(k) = 9$) y las siguientes matrices N_{ik} y M_{ik}:

$$N_{ik} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

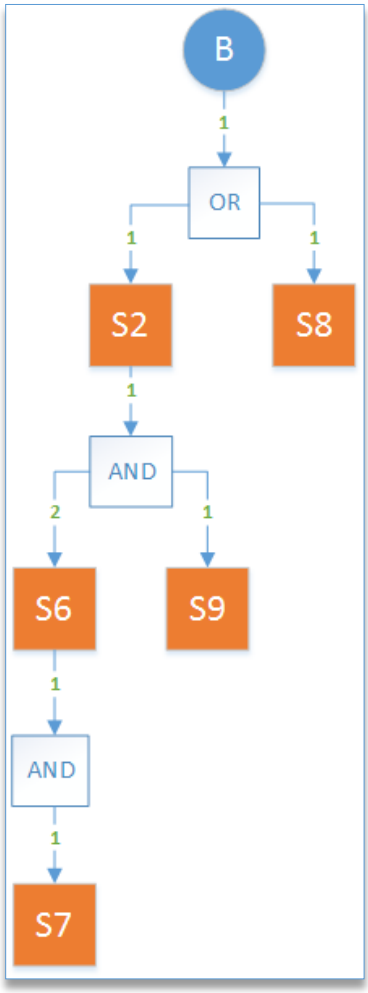
$$M_{ik} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2)$$

$$SS_{kk'} = {N_{ik}}^{-1} \times M_{ik} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (3)$$

Para facilitar la lectura se han enumerado los productos con letras de la A a la Z y a los strokes se les ha colocado la letra 'S' al frente.

Grafo del producto B:

Como se puede observar, el grafo basado en la matriz $SS_{kk'}$ elimina la interacción de stroke-producto con las que se presentan las matrices N_{ik} y M_{ik} para ofrecernos una visión global de los requerimientos stroke-stroke. Datos importantes como el 'número de opciones' o formas de generar o llevar a cabo por producto (o por stroke en este caso) se obtiene de la matriz N_{ik} así:



Grafo 1. Producto B

$$numOpc_i = \sum_{k=0}^{card(k)} N_{ik}$$

Los arcos del grafo guardan información sobre cuántas ejecuciones de un stroke 'hijo' se requieren para habilitar un stroke 'padre'. Por esto en $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, $\mathcal{N}$ son los strokes y $\mathcal{A}$ son las relaciones de requerimiento entre un stroke y sus 'hijos' dadas en $SS_{kk'}$.

En la matriz $SS_{kk'}$ se leen los 'padres' como las columnas y los 'hijos' serán los hijos, así, el stroke S2 requiere 2 *ejecuciones* de S6 y 1 de S9.