Algoritmo para la resolución de una heurística Simple

Daniela Simancas Mateus

Datos de entrada

- Matrices: Nik, Mik
- Demanda o requerimientos proyectados
- Tamaño fijo de lote
- Inventario inicial
- Costos de almacenamiento
- Costos de preparación
- Costos de operación
- Costos por falta de inventario
- Lead times
- Horizonte de planeación

Clases

Principales:

- Datos
- FreshStart

Complementarias:

- MatrixHandling
- LotSizingRule
- NodeDoulbe
- TreeDouble

De tipo:

- ChildComponents
- DoubleInt
- ProductTable
- Struct

Heurística en FreshStart

```java
/**
 * Calculates the when, which and how many are the required strokes for the entire production plan.
 * @param productsTables
 * @param strokeMatrix
 * @return A matrix with length: the number of strokes and width: the period length.
 */
private double[][] freshHeuristic(List<ProductTable> productsTables, double[][] strokeMatrix){

    productsTables = sortProductList(productsTables);

    for (ProductTable productsTable : productsTables) {

        int lot_size_rule = 2;
        productsTable.productRequirements = MPSof(productsTable,lot_size_rule);

        for (int t = 0; t<PERIOD_LENGTH; t++) {

            if (productsTable.productRequirements[t] > 0) {
                Struct beststroke = bestStroke(productsTable.productKey, t, productsTable.productRequirements[t]);

                if(beststroke != null){
                    double value = beststroke.RunsXTimes;
                    strokeMatrix[beststroke.strokeKey-1][beststroke.timeWhenRuns] += value;

                    productsTable.productRequirements[t] = 0.0;
                    productsTables = updateProductsTables(beststroke, productsTables);
                }
                else{
```

Heurística en FreshStart

```
29                      productsTable.productRequirements[t] = 0.0;
30                      System.out.println("Couldn't complete request: ["+productsTable.productKey+"] with amount: "
31                      + "["+productsTable.productRequirements[t]+"]. Not enough time: t="+t+". Moving on to day "+(t+1)+".");
32                  }
33              }
34          }
35          for (int p=0;p<productsTables.size();p++) {
36
37              if (productsTables.get(p).hasNoRequirements() && productsTables.size()>1) {
38                  productsTables.remove(p);
39                  p=-1; //Para que comience a evaluar la lista desde el principio nuevamente
40              }
41              else if(productsTables.get(p).hasNoRequirements() && productsTables.size()==1){
42                  return strokeMatrix;
43              }
44          }
45
46          return freshHeuristic(productsTables, strokeMatrix);
47      }
48
49      return strokeMatrix;
50
51  }
52
```

Algoritmo en formato generalizado

```
SimpleHeuristic(OCk, SCk, HCi, Nik, Mik){

  SSk = Nik(t)*Mik;
  G(N,A)/N <- k, A <- Nik;
  estimateAccumOC(G(N,A),OCk);
  estimateAccumSC(G(N,A),SCk);

  List recursiveAlgorithm(productList,strokeList){

    sortProductList();
    for(int i = 0; i < CARD(i); i++){

      for(int t = 0; t < CARD(t); t++){
        MPSHeuristic();

        if(projectedRequirements(i,t) > 0){
          for(int k = 0; k < CARD(k); k++){
            if( isRelatedTo(i,k) && (t - LeadTimeOf(k) > 0) ){
              optionCost(i,k,t - LeadTimeOf(k));
            }
          }

          stroke(k,t) = minArg(optionCost(i,k,t - LeadTimeOf(k)));
          strokeList.add(stroke(k,t));
          productList.add(productsUnder(stroke(k,t)));
        }
      }

      return recursiveAlgorithm(productList,strokeList);
    }
    return strokeList;
  }
}
```

Datos de salida

- Producción final de strokes
  - Costos de producción
- Recepciones planeadas
- Inventario
  - Costos de inventario
- Stock outs
  - Costos por stock out