Boletín de Innovación, Logística y Operaciones

BILO Vol. 2 no. 2, Julio - Diciembre de 2020

MODELADO MATEMÁTICO EN JAVA UTILIZANDO GUROBI

DOI: https://doi.org/10.17981/bilo.2.2.2020.06
Fecha de Recepción: 11/01/2021. Fecha de Publicación: 12/02/201

Jairo Rafael Coronado Hernández
Universidad de la Costa, CUC. Barranquilla (Colombia).
Jcoronad18@cuc.edu.co

1. Introducción

De forma muy breve, pero práctica se exponen en este documento como modelar los problemas de programación matemática desde la programación en Java para utilizar Gurobi como Solver. Básicamente consiste en llenar el Tableau Simplex que recibe el Solver GUROBI. Para crear el tableau hay que tener claro los coeficientes de la función objetivo y cada uno de los coeficientes tecnológicos que acompañan las restricciones. Es importante que cuando se declaran todas las variables de acuerdo a su rango y tipo (int, >=0, ect.) siempre se declara el coeficiente de la función de costes de todas las variables; significa esto que cada vez que se desee declarar una variable se debe colocar su coeficiente de costes así sea "0". La declaración de una variable libre ("free") no está por defecto y este debe realizarse con un rango desde -infinito a el infinito. Después de creadas las variables se actualiza el modelo, es importante, de lo contrario no funciona. Las restricciones se declaran por trozos que se van agregando uno a uno, a un objeto que se llaman expresiones. Yo considero que la forma más práctica es crear una para el lado izquierdo y otra para el lado derecho de cada ecuación. Después de creadas estas expresiones se crea la restricción añadiendo obviamente el lado izquierdo, el (>=,= , <=), el lado derecho, y el nombre de la ecuación. El nombre de la ecuación es muy importante aunque no afecte la resolución; lo digo porque cuando pasas el modelo a CPLEX ("*.lp"), este nombre es el que se escribe(lo he probado) dentro del modelo *.lp. Así que escribirlas como en AMPL cuando se declare. Finalizado lo anterior se procede a seleccionar el método de resolución que es opcional, si un Simplex-Dual, etc. Una vez seleccionado o no el método de resolución (opcional, repito), se da la instrucción de resolver el problema (obligatorio).

La extracción de variables es muy fácil. Simplemente se escribe la variable y se toma el valor con una instrucción muy sencilla como se observará más adelante. El manual de referencia de Gurobi para Java esta disponible en http://www.gurobi.com/doc/40/refman/node227.html.

2. Pasos previos

Antes comenzar a escribir los modelos en Java tener en cuenta los siguientes pasos.

1. Tener definido cual es el modelo y en ese sentido DEFINIR MUY BIEN LAS VARIABLES.
2. Depurar el modelo en un lenguaje de programación matemática conocido (aunque hay cosas que se podrán hacer en Java que en AMPL no). Probar que el modelo tiene solución, para que cuando este terminado el modelo se puedan hacer ajustes más rápidos.
3. Crear una clase en el proyecto Java para el modelo
4. Tener identificados de que objetos se van a extraer los parámetros del modelo

3. Estructura de una clase en Java para modelos Gurobi

En general siguen la misma estructura que la programación matemática

1. Definir los indices del modelo(en este caso se extraen del objeto que contiene los parámetros)
2. Definir los parámetros (del objeto)

3. Crear un entorno de trabajo: ***GRBEnv env = new GRBEnv();***
4. Crear el modelo : ***GRBModel model = new GRBModel(env);***
5. Definir con el constructor ***GRBVar*** cada una de las variables. Aun no se han definido los tipos, solo se crea el objeto: ***GRBVar [][] kap = new GRBVar[le.numR][le.numT];***
6. Agregar cada variable al modelo considerando el tipo, el coeficiente de la función objetiva y su nombre: ***v[i][t]=model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "v("le.getSKU[i]","(t+1)")");***
7. Definir si va a maximizar o minimizar, con -1 y 1, respectivamente: ***model.set(GRB.IntAttr.ModelSense, -1);***
8. Actualizar el modelo: ***model.update();***
9. Escribir cada una de las restricciones:
    a. Crear objeto del lado izquierdo: ***GRBLinExpr LI = new GRBLinExpr();***
    b. Crear objeto del lado derecho: ***GRBLinExpr LD = new GRBLinExpr();***
    c. Agregar variables con sus respectivos coeficientes tecnológicos a el lado correspondiente:***LI.addTerm(1, con[i][k][La]);***
    d. Agregar restricción al modelo matemático: ***model.addConstr(LI, GRB.EQUAL, LD,"ConsumoUnit1("le.getSKU[i]","le.getStroke[k]")");***
10. Seleccionar método de resolución (opcional): ***model.getEnv().set(GRB.IntParam.RootMethod, GRB.LPMETHOD_DUAL);***
11. Solucionar modelo: ***model.optimize();***
12. Extraer los valores de las variables: ***kap1 [r][t] = kap[r][t].get(GRB.DoubleAttr.X);***

3. Acerca de las Variables y la Función Objetiva

Primero se construyen las variables con el respectivo constructor.

```
GRBVar[] open = new GRBVar[P];
GRBVar[][] transport = new GRBVar[W][P];
```

Donde W y P son respectivamente la cardinalidad de cada uno de los los conjuntos.

Ya teniendo el objeto predefinido se procede a definir el tipo de variable y la construcción de la función objetivo.Gurobi nos dice:

Variables are added through the *addVar()* method on a model object. A variable is always associated with a particular model.
The first and second arguments to the *addVar()* call are the variable lower and upper bounds, respectively. The third argument is the linear objective coefficient (the default objective sense is minimization, so we've negated the coefficients from our maximization objective here). The fourth argument is the variable type. Our variables are all binary in this example. The final argument is the name of the variable. The *addVar()* method has been overloaded to accept several different argument lists. Please refer to the Gurobi Reference Manual for further details.

Veamos el ejemplo para:

```
FO:( sum{w in W, p in P} COSTES[w][p]* transport[w][p] )+( sum{p in P} COSTOFIJO[p]*open[p]);


)

for (int p = 0; p < nPlants; ++p) {
    open[p]=model.addVar(0, 1, COSTOFIJO[p], GRB.BINARY, SetCap[p]);
}

for (int w = 0; w < W; ++w) {
    for (int p = 0; p < P; ++p) {
```

```
        transport[w][p] = model.addVar(0, GRB.INFINITY, COSTES[w][p], GRB.CONTINUOUS, "Trans"  + p + "." + w);
    }
}
```

* Una variable libre "free" se define como :

transport[w][p] = model.addVar(-GRB.INFINITY, GRB.INFINITY, ...

- Si una variable no tiene un coste asociado en la función objetiva simplemente se coloca cero: kap[r][t]= model.addVar(0, GRB.INFINITY, 0, GRB.C...
- Si se está maximizando una función de utilidad y hay términos de costes que restan, simplemente se coloca como negativo el coeficiente de costo.

Una vez realzada estas operaciones con cada una de las variables del modelo, queda construida la función objetiva. Se procede a maximizar(-1) o minimizar(1):

```
model.set(GRB.IntAttr.ModelSense, -1);// -1 para maximizar, 1 para minimizar
model.update();
```

4. Acerca de las expresiones y restricciones

Para construir una restricción, primero hay que construir el lado derecho, luego el lado izquierdo y posteriormente se añade la info a la restricción. Que dice Gurobi:

The first argument to *addConstr()* is the left-hand side of the constraint. We built the left-hand side by first creating an empty linear expression object, and then adding three terms to it. The second argument is the constraint sense (*GRB_LESS_EQUAL*, *GRB_GREATER_EQUAL*, or*GRB_EQUAL*). The third argument is the right-hand side (a constant in our example). The final argument is the constraint name. Several signatures are available for *addConstr()*. Please consult the Gurobi Reference Manual for details.As with variables, constraints are always associated with a specific model. They are created using the *addConstr()* or *addConstrs()* methods on the model object.

Veamos los siguientes ejemplos:

Ejemplo 1: Modelar una restricción cuando solo hay variables

s.t. inv2{i in I, t in (La+1)..Lb, e in 1..E}: y[i,t,e]= y[i,t-1,e]-v[i,t,e]-
consumos[i,t,e]+w[i,t,e]+produccion[i,t,e];

El modelado de esta ecuación en Java es equivalente a:

```
for(int i=0; i<le.numI;i++){
        for(int t=(La+1);t<(tamano);t++){
              GRBLinExpr LI= new GRBLinExpr();
              GRBLinExpr LD= new GRBLinExpr();
              LI.addTerm(1, y[i][t]);
              LD.addTerm(1, y[i][t-1]);
              LD.addTerm(-1, v[i][t]);
              LD.addTerm(-1, consumos[i][t]);
              LD.addTerm(1, w[i][t]);
              LD.addTerm(1, produccion[i][t]);
              model.addConstr(LI, GRB.EQUAL,LD, "inv2{"+le.getSKU[i]+","+le.getPeriodos[t]+"}");
        }
    }
```

Tener en cuenta que en Java una las posiciones de los arrays comienzan en 0. AL AMPL o GAMS por ejemplo, les da igual. Tener mucho cuidado con los índices.

Ejemplo 2: Modelar una restricción cuando hay variables y constantes

s.t. back2{i in I, t in (La+1)..Lb, e in 1..E}: beta[i, t, e]= beta[i, t-1, e]+D[i, t, e]-v[i, t, e];

En este caso la constante corresponde a ***le.dEMito[i][t][ocurr].*** El modelado de esta ecuación en Java es equivalente a:

```
for(int i=0; i<le.numI;i++){
   for(int t=(La+1);t<(tamano);t++){
         GRBLinExpr back2= new GRBLinExpr();
         GRBLinExpr back2d= new GRBLinExpr();
         back2.addTerm(1, beta[i][t]);
         back2d.addTerm(1, beta[i][t-1]);
         back2d.addTerm(-1, v[i][t]);
         back2d.addConstant(le.dEMito[i][t][ocurr]);
         model.addConstr(back2, GRB.EQUAL, back2d, "Back2{ "+le.getSKU[i]+","+le.getPeriodos[t]+"}");
      }
}
```

Ejemplo 3: Modelar una restricción cuando hay condiciones

//s.t. GeneraUnit2b{i in I, k in K, t in La..La+LT[k]-1, e in 1..E}: if LT[k]>0 then x[i,k,t,e]=0 ;

El modelado de esta ecuación en Java es equivalente a:

```
for(int i=0; i<le.numI;i++){
         for(int k=0; k<le.numK;k++){
            for(int t=(La);t<(La+le.lt[k]);t++){
                  if(le.lt[k]>0){
                     GRBLinExpr GeneraUnit2b = new GRBLinExpr();
                     GeneraUnit2b.addTerm(1, x[i][k][t]);
                     model.addConstr(GeneraUnit2b, GRB.EQUAL, 0, "GeneraUnit2b("+i+","+k+","+(t+1)+")");
                  }
             }
          }
      }
```

5. Acerca de la extracción de info y escritura de .lp, *.mps, *.out

Extracción de info

Se puede extraer lo que se quiera. Lo que más interesa son los valores de las variables, aunque tambien es posible las duales de las restricciones, etc.

Ejemplo:

```
produccion1=new double[le.numI][le.numT][le.numE];
          for (int i=0;i<le.numI;i++){
              for(int t=0; t<le.numT;t++){
                for(int e=0; e<le.numE; e++){
                   produccion1[i][t][e] = produccion[i][t][e].get(GRB.DoubleAttr.X);
                 }
               }
```

}

Para escribir modelos en CPLEX y MPS

model.write("modelo.lp";)

model.write("modelo.mps";)

Para escribir resultados

model.write("resultados_variables.out";)

Precios sombras en GUROBI desde JAVA

En este Post se presenta como extraer precios sombras de un modelo desde Java para GUROBI. En la web de GUROBI no se muestra como modelar una restricción para que de ella se extraigan los precios sombras o el valor de las variables duales del problema. Es por ello que este post resulta interesante. Solo funciona para modelos lineales puros. Los pasos son:

1) Declarar la ecuación. Se declara muy similar a una variable, siendo una restricción.

GRBConstr[] Ejem = new GRBConstr[nPlants];

2) Añadir expresiones a la Restricción: Cuando se añaden las expresiones al model, también se asigna directamente ese expresión a la ecuación, como si fuera una variable.

```
for (int p = 0; p < nPlants; ++p) {
      GRBLinExpr ptot = new GRBLinExpr();
      for (int w = 0; w < nWarehouses; ++w) {
        ptot.addTerm(1.0, transport[w][p]);
      }
      GRBLinExpr limit = new GRBLinExpr();
      limit.addTerm(Capacity[p], open[p]);
      Ejem[p]=model.addConstr(ptot, GRB.LESS_EQUAL, limit, "Capacity" + p);
}
```

3) Extraer las variables: Para extraer las variables duales se realiza lo siguiente.

```
for (int p = 0; p < nPlants; ++p) {
      Precio_SombraEjem[p]= Ejem[p].get(GRB.DoubleAttr.Pi);
}
```

Así mismo, es posible extraer más información de las ecuaciones al ver los atributos en la page de GUROBI en el link: http://www.gurobi.com/doc/40/refman/node571.html, sección **Constraint attributes**.

Ejemplos de modelos

Ejemplo del modelo del Transporte con coste de apertura de una instalación(Ejemplo de Gurobi)

```
import gurobi.*;

public class Facility {

  public static void main(String[] args) {
    try {
```

```
        // Warehouse demand in thousands of units
        double Demand[] = new double[] { 15, 18, 14, 20 };

        String SetDemanda[] = new String[] {"a", "b", "c", "d"};

        // Plant capacity in thousands of units
        double Capacity[] = new double[] { 20, 22, 17, 19, 18 };
        String SetCap[] = new String[] {"c1", "c2", "c3", "c4", "c5"};

        // Fixed costs for each plant
        double FixedCosts[] =
            new double[] { 12000, 15000, 17000, 13000, 16000 };

        // Transportation costs per thousand units
        double TransCosts[][] =
            new double[][] { { 4000, 2000, 3000, 2500, 4500 },
                { 2500, 2600, 3400, 3000, 4000 },
                { 1200, 1800, 2600, 4100, 3000 },
                { 2200, 2600, 3100, 3700, 3200 } };

        // Number of plants and warehouses
        int nPlants = Capacity.length;
        int nWarehouses = Demand.length;

        // Model
        GRBEnv env = new GRBEnv();
        GRBModel model = new GRBModel(env);
        model.set(GRB.StringAttr.ModelName, "facility");
               GRBVar[] open = new GRBVar[nPlants];       GRBVar[][] transport =
new GRBVar[nWarehouses][nPlants];
        // Plant open decision variables: open[p] == 1 if plant p is open.
        for (int p = 0; p < nPlants; ++p) {
          //open[p] = model.addVar(0, 1, FixedCosts[p], GRB.BINARY, "Open" + p);
          open[p]=model.addVar(0, 1, FixedCosts[p], GRB.BINARY, SetCap[p]);
        }


        // Transportation decision variables: how much to transport from
        // a plant p to a warehouse w
        for (int w = 0; w < nWarehouses; ++w) {
          for (int p = 0; p < nPlants; ++p) {
            transport[w][p] =
                model.addVar(0, GRB.INFINITY, TransCosts[w][p], GRB.CONTINUOUS,
                             "Trans" + p + "." + w);
          }
        }

        // The objective is to minimize the total fixed and variable costs
        model.set(GRB.IntAttr.ModelSense, 1);

        // Update model to integrate new variables
        model.update();

        // Production constraints
        // Note that the right-hand limit sets the production to zero if
        // the plant is closed
        for (int p = 0; p < nPlants; ++p) {
          GRBLinExpr ptot = new GRBLinExpr();
          for (int w = 0; w < nWarehouses; ++w) {
```

```
      ptot.addTerm(1.0, transport[w][p]);
    }
    GRBLinExpr limit = new GRBLinExpr();
    limit.addTerm(Capacity[p], open[p]);
    model.addConstr(ptot, GRB.LESS_EQUAL, limit, "Capacity" + p);
  }

  // Demand constraints
  for (int w = 0; w < nWarehouses; ++w) {
    GRBLinExpr dtot = new GRBLinExpr();
    for (int p = 0; p < nPlants; ++p) {
      dtot.addTerm(1, transport[w][p]);
    }
    model.addConstr(dtot, GRB.EQUAL, Demand[w], "Demand" + w);
  }

  // Guess at the starting point: close the plant with the highest
  // fixed costs; open all others

  // First, open all plants
  for (int p = 0; p < nPlants; ++p) {
    open[p].set(GRB.DoubleAttr.Start, 1.0);
  }

  // Now close the plant with the highest fixed cost
  System.out.println("Initial guess:");
  double maxFixed = -GRB.INFINITY;
  for (int p = 0; p < nPlants; ++p) {
    if (FixedCosts[p] > maxFixed) {
      maxFixed = FixedCosts[p];
    }
  }
  for (int p = 0; p < nPlants; ++p) {
    if (FixedCosts[p] == maxFixed) {
      open[p].set(GRB.DoubleAttr.Start, 0.0);
      System.out.println("Closing plant " + p + "\n");
     break;
    }
  }

  // Use barrier to solve root relaxation
  //model.getEnv().set(GRB.IntParam.RootMethod, GRB.LPMETHOD_DUAL);

  // Solve
  model.optimize();

  double a=0;
  // Print solution
  System.out.println("\nTOTAL COSTS: " + model.get(GRB.DoubleAttr.ObjVal));
  System.out.println("SOLUTION:");
  for (int p = 0; p < nPlants; ++p) {
    if (open[p].get(GRB.DoubleAttr.X) == 1.0) {
      System.out.println(SetCap[p] + " open:");
      for (int w = 0; w < nWarehouses; ++w) {
          a=  transport[w][p].get(GRB.DoubleAttr.X);
        if (transport[w][p].get(GRB.DoubleAttr.X) > 0.0001) {
          System.out.println("  Transport " +
              transport[w][p].get(GRB.DoubleAttr.X) +
              " units to warehouse " + w);
        }
```

```
            }
          } else {
            System.out.println("Plant " + p + " closed!");
          }
          System.out.println("el valor de a es       ....    =   "+ a);
        }

      } catch (GRBException e) {
        System.out.println("Error code: " + e.getErrorCode() + ". " +
            e.getMessage());
      }
    }
  }
```