Gurobi and Java

Erasmus University Rotterdam

Part 1 - Content

What is gurobi?

- A black-box optimizer
- Given a mathematical model, gives solution
  - Linear Programs (LPs)
  - Integer Programs (IPs)
  - Mixed-Integer Programs (MIPs)
  - Convex Quadratic Programs, Second-order Cone Programs, etc.
- How to "give" model to gurobi? $\rightarrow$ use programming interface
- AIMMS is such an interface
- We can also use Java to communicate with gurobi

Advantages of using Java

"*If Java can do the same as AIMMS, why learn Java instead of just using AIMMS?*"

Java offers a number of advantages:

- You can use existing Java knowledge
- All Java classes (`ArrayList`, `Scanner`) are available
- Better documented than AIMMS
- Incredibly easy to make general model (that works with all kinds of data)
- Adjust model after/while solving

Gurobi in Java

General idea

- You build your model as an object. More specifically, an `GRBModel` object.
- This object contains
  - Variable objects (`GRBVar`)
  - The objective
  - The constraints
- Then, calling the method `optimize()` on the object, lets gurobi solve the model.
- The solution values can be retrieved from the object.

Important classes and usage — `GRBModel`

`GRBModel`

- Class that contains entire LP/IP/MIP model
- Build all other components from this object
- Creation:

  ```
  GRBModel model = new GRBModel(env);
  ```

  Given a `GRBModel` object, you can also make a new `GRBModel` object that is a copy of the old model:

  ```
  GRBModel modelCopy = new GRBModel(model);
  ```

Important classes and usage — `GRBEnv`

`GRBEnv` (environment object)

- Typically the first gurobi object you create and the last one you destroy
- Needed as input for the `GRBModel` constructor
- Not very important for now (but holds the gurobi license and captures a set of parameter settings)
- Creation:

```
GRBEnv env = new GRBEnv();
env.start();
```

Important classes and usage — GRBVar

`GRBVar`

- Class that defines optimization variables
- Created by adding it to a model (using `GRBModel.addVar`), rather than by using a `GRBVar` constructor
- Creation of continuous variable (with `double` bounds `a` and `b` and name `x_cont`):

```
GRBVar x_cont = model.addVar(a, b, 0.0, GRB.CONTINUOUS,
    "x_cont");
```

- Creation of integer variable (with `int` bounds `a` and `b` and name `x_int`):

```
GRBVar x_int = model.addVar(a, b, 0.0, GRB.INTEGER,
    "x_int");
```

- Creation of boolean variable (with name `x_bool`):

```
GRBVar x_bool = model.addVar(0, 1, 0.0, GRB.BINARY,
    "x_bool");
```

Important classes and usage — GRBVar

- Creating of array of n continuous variables:

```
GRBVar[] x_array = new GRBVar[n];
for (int i = 0; i < n; i++) {
    x_array[i] = model.addVar(a, b, 0.0, GRB.CONTINUOUS,
     "x("+i+")");
}
```

- Creating an n-by-m matrix of continuous variables:

```
GRBVar[][] x_matrix = new GRBVar[n][m];
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        x_matrix[i][j] = model.addVar(a, b, 0.0,
         GRB.CONTINUOUS, "x("+i+","+j+")");
    }
}
```

- In the same way, you can store variables in a List, HashMap, ...

Important classes and usage — GRBLinExpr

- Contains a numeric expression, usually of the form $\sum_i a_i x_i + b$, with a_i, b arbitrary constants and x_i optimization variables.
- Used to make constraints and the objective
- Creation of a new (empty) linear expression (for a quadratic expression, use `GRBQuadExpr`):

```
GRBLinExpr expr = new GRBLinExpr();
```

- Adding a variable `GRBVar x` with coefficient `a` to `GRBLinExpr expr`:

```
expr.addTerm(a, x);
```

- Adding a constant `b` to `GRBLinExpr expr`:

```
expr.addConstant(b);
```

- Removing all terms associated with variable `GRBVar x` from `GRBLinExpr expr`:

```
expr.remove(x);
```

Important classes and usage — GRBLinExpr

- Adding `GRBLinExpr expr2` to `GRBLinExpr expr1`:

```java
expr1.add(expr2);
```

- Adding `GRBLinExpr expr2` to `GRBLinExpr expr1` `k` times (use `k=-1` to obtain the difference between `expr1` and `expr2`):

```java
expr1.multAdd(k, expr2);
```

- Creating an expression that is the sum-product of `GRBVar[] x` and `double[] a` (that is, $\sum_i$ `a[`i`]x[`i`]`):

```java
GRBLinExpr sumprod_expr = new GRBLinExpr();
for (int i = 0; i < x.length; i++) {
    sumprod_expr.addTerm(a[i], x[i]);
}
```

  Alternatively (only works for two arrays `GRBVar[] x`, `double[] a`):

```java
GRBLinExpr sumprod_expr = new GRBLinExpr();
sumprod_expr.addTerms(a, x);
```

Important classes and usage — adding the objective

- Adding the objective is as simple as creating an `GRBLinExpr` containing the objective
- Assuming `GRBLinExpr obj` contains the objective:
  - Maximize `obj`
    ```
    model.setObjective(obj, GRB.MAXIMIZE);
    ```
  - Minimize `obj`
    ```
    model.setObjective(obj, GRB.MINIMIZE);
    ```

Optional — alternative for specifying the objective

- Instead of defining a `GRBLinExpr` object and adding it to the model with the `setObjective()` method, you can specify the objective coefficients of the variables directly, when creating them, e.g. creation of continuous variable with `double` bounds `a` and `b` and objective coefficient `c`:

```
GRBVar x_cont = model.addVar(a, b, c, GRB.CONTINUOUS);
```

- When using this, you can also set the optimization sense to maximization (default is minimization):

```
model.set(GRB.IntAttr.ModelSense, -1);  // -1 for
    maximization, 1 for minimization
```

- Disadvantages compared to creating `GRBLinExpr` or `GRBQuadExpr` and adding it using `setObjective`:
  - Limited Flexibility: you cannot easily include nonlinear terms or complex expressions in the objective function using this method.
  - Dynamic Updates Difficulty: changing the objective function dynamically during optimization is more challenging with this approach.

Important classes and usage — adding constraints

- Adding constraints is as simple as creating `GRBLinExpr` for the lhs and rhs
- You can also use a `double` or `GRBVar` for the lhs or rhs
  - Adding the constraint `lhs <= rhs` (with name `c1`):
    ```
    model.addConstr(lhs, GRB.LESS_EQUAL, rhs, "c1");
    ```
  - Adding the constraint `lhs >= rhs` (with name `c2`):
    ```
    model.addConstr(lhs, GRB.GREATER_EQUAL, rhs, "c2");
    ```
  - Adding the constraint `lhs == rhs` (with name `c3`):
    ```
    model.addConstr(lhs, GRB.EQUAL, rhs, "c3");
    ```
- Example: $2x - 3y \leq 7$

```
GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(2.0, x);
expr.addTerm(-3.0, y);
model.addConstr(expr, GRB.LESS_EQUAL, 7.0, "constr");
```

Important classes and usage — solving the model

- To solve the model, call the method

```
model.optimize();
```

- To get the solution status, use (after `model.optimize()`)

```
model.get(GRB.IntAttr.Status);
```

- This method returns an enum which (among others) takes the values
  - `GRB.Status.INFEASIBLE` if the model is infeasible
  - `GRB.Status.UNBOUNDED` if the model is unbounded
  - `GRB.Status.INF_OR_UNBD` if the model is infeasible or unbounded (determined by presolve, gurobi does not know which one is true)
  - `GRB.Status.OPTIMAL` if an optimal solution is found

Important classes and usage — retrieving the solution

- If `model.get(GRB.IntAttr.Status)` equals `GRB.Status.OPTIMAL`, you can query the optimal solution
- Query of (and print) `GRBVar x`:

```
double xVal = x.get(GRB.DoubleAttr.X);
System.out.println(x.get(GRB.StringAttr.VarName) + " " +
    xVal);
```

- Query of optimal objective value:

```
double objVal = model.get(GRB.DoubleAttr.ObjVal);
```

- Query of `GRBLinExpr expr`:

```
double exprVal = expr.getValue();
```

Important classes and usage — printing the model

- Sometimes, the model is wrongfully infeasible due to coding errors
- To “see” the model, you can print it to an .lp file and inspect it manually
- Exporting the model to the text file “Model.lp”:

```
model.write("Model.lp");
```

- Note: the .lp extension is necessary!
- Also, the names of your variables (and/or constraints) show up in this file $\rightarrow$ use descriptive names

Important classes and usage — output logging

- By default, gurobi starts printing diagnostic information to the console whenever `model.optimize()` is called
- Useful for large, one-time only solve models
- Annoying when solving hundreds of tiny models
- You can suppress this output by calling, before starting the `GRBEnv` object (i.e. before calling `env.start();`):

```
env.set(GRB.IntParam.OutputFlag, 0);
```

Important classes and usage — closing and memory

- `GRBModel` models can take a lot of memory from the computer
- When done with the model, it is good practice to dispose of it:

```
model.dispose();
```

- Make sure to do the same for the `GRBEnv` object:

```
env.dispose();
```

Simple model

Let's solve the following model:

$$\begin{aligned} \max \quad & 10x + 4y - 2z \\ \text{s.t.} \quad & 2x + y \le z \\ & 3y + z \ge -1 \\ & x + y + z = 5 \\ & 0 \le x \le 10 \\ & y \ge 0 \\ & z \le 5 \\ & z \in \mathbb{Z} \end{aligned}$$

Simple model — Java code

Model creation and solving method (part 1):

```java
public static void solveSimpleModel() throws GRBException {
    // Create the environment
    GRBEnv env = new GRBEnv();
    // If you want to suppress automatic output: env.set(GRB.IntParam.OutputFlag, 0);
    env.start();

    // Create empty model
    GRBModel model = new GRBModel(env);

    // Create the variables and their domain restrictions
    GRBVar x = model.addVar(0, 10, 0.0, GRB.CONTINUOUS, "x");
    GRBVar y = model.addVar(0, Double.POSITIVE_INFINITY, 0.0, GRB.CONTINUOUS, "y");
    GRBVar z = model.addVar(Integer.MIN_VALUE, 5, 0.0, GRB.INTEGER, "z");

    // Create the objective
    GRBLinExpr obj = new GRBLinExpr();
    obj.addTerm(10.0, x);
    obj.addTerm(4.0, y);
    obj.addTerm(-2.0, z);
    model.setObjective(obj, GRB.MAXIMIZE);

    // Add the restrictions
    GRBLinExpr expr = new GRBLinExpr();
    expr.addTerm(2.0, x);
    expr.addTerm(1.0, y);
    model.addConstr(expr, GRB.LESS_EQUAL, z, "c1");

    ...
```

Simple model — Java code

Model creation and solving method (part 2):

```
    ...

    expr = new GRBLinExpr();
    expr.addTerm(3.0, y);
    expr.addTerm(1.0, z);
    model.addConstr(expr, GRB.GREATER_EQUAL, -1, "c2");

    expr = new GRBLinExpr();
    expr.addTerm(1.0, x);
    expr.addTerm(1.0, y);
    expr.addTerm(1.0, z);
    model.addConstr(expr, GRB.EQUAL, 5, "c3");

    // Solve the model
    model.optimize();

    // Query the solution
    if (model.get(GRB.IntAttr.Status) == GRB.Status.OPTIMAL) {
        System.out.println("Found optimal solution!");
        System.out.println("Objective = " + model.get(GRB.DoubleAttr.ObjVal));
        System.out.println(x.get(GRB.StringAttr.VarName) + " " + x.get(GRB.DoubleAttr.X)
           + "\n" + y.get(GRB.StringAttr.VarName) + " " + y.get(GRB.DoubleAttr.X)
           + "\n" + z.get(GRB.StringAttr.VarName) + " " + z.get(GRB.DoubleAttr.X));
    } else {
        System.out.println("No optimal solution found");
    }

    // Close the model
    model.dispose();
    env.dispose();
```

Simple model – Java code

Main method:

```
public static void main(String[] args) {
    try {
        solveSimpleModel();
    }
    catch (GRBException e) {
        System.out.println("A Gurobi exception occured: " + e.getMessage());
        e.printStackTrace();
    }
}
```

Java console output:

```
Found optimal solution!
Objective = 8.0
x 1.0
y 1.0
z 3.0
```

Vector model

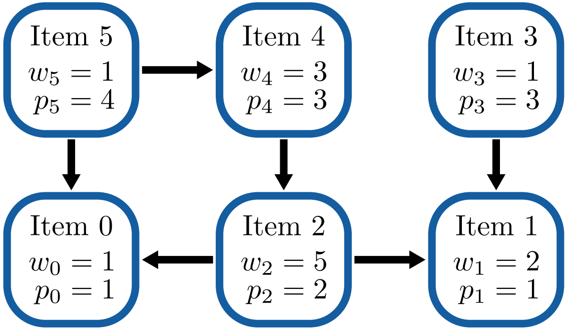
Let's solve the Precedence Constrained Knapsack Problem

- List of n items, with weight w_i and profit p_i
- Capacity of the knapsack b
- Directed precedence graph G on the items

The aim is to select the items such that

- The sum of their profits is maximized
- The sum of their weights does not exceed the capacity
- A particular item can only be included if all its successors in G are also included

Vector model



- Item 0 can always be included
- Item 3 needs item 1
- Item 2 needs both item 0 and 1

Vector model

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n} p_i x_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leq b \\
& x_i \leq x_j & \forall (i,j) \in G \\
& x_i \in \mathbb{B} & \forall i \in \{1, \ldots, n\}
\end{aligned}
$$

Vector model — Java code

Model creation and solving method (part 1):

```java
public static void solveVectorModel(double[] w, double[] p, double b, int[][] G) throws
     GRBException {
    // Create the environment and empty model
    GRBEnv env = new GRBEnv();
    env.start();
    GRBModel model = new GRBModel(env);

    // Get n
    int n = w.length;

    /* INPUT CHECKS HERE! */

    // Create the variables and their domain restrictions
    GRBVar[] x = new GRBVar[n];
    for (int i = 0; i < n; i++) {
        x[i] = model.addVar(0, 1, 0.0, GRB.BINARY, "x(" + i + ")");
    }

    // Create the objective
    GRBLinExpr objExpr = new GRBLinExpr();
    objExpr.addTerms(p, x);
    model.setObjective(objExpr, GRB.MAXIMIZE);

    // Add the capacity restriction
    GRBLinExpr weightExpr = new GRBLinExpr();
    weightExpr.addTerms(w, x);
    model.addConstr(weightExpr, GRB.LESS_EQUAL, b, "CapacityRestr");

    ...
```

Vector model — Java code

Model creation and solving method (part 2):

```
    ...

    // Add the precedence constraints
    for (int i = 0; i < G.length; i++) {
        model.addConstr(x[G[i][0]],  GRB.LESS_EQUAL, x[G[i][1]],
            "Precedence("+G[i][0]+","+G[i][1]+")");
    }

    // Solve the model
    model.optimize();

    // Query the solution
    if (model.get(GRB.IntAttr.Status) == GRB.Status.OPTIMAL) {
        System.out.println("Found optimal solution!");
        System.out.println("Objective = " + model.get(GRB.DoubleAttr.ObjVal));
        System.out.print("Used items: ");
        for (int i = 0; i < n; i++) {
            if (x[i].get(GRB.DoubleAttr.X) >= 0.5) {
                System.out.print(i + " ");
            }
        }
    }
    else {
        System.out.println("No optimal solution found");
    }

    // Close the model
    model.dispose();
    env.dispose();
}
```

Vector model — Java code

Main method, with hardcoded instance:

```java
public static void main(String[] args) {
    double[] w = {1, 2, 5, 1, 3, 1};
    double[] p = {1, 1, 2, 3, 3, 4};
    double b = 10;
    int[][] G = {{2, 0}, {2, 1},
                 {4, 2}, {3, 1},
                 {5, 0}, {5, 4}};
    try {
        solveVectorModel(w, p, b, G);
    }
    catch (GRBException e) {
        System.out.println("A Gurobi exception occured: " + e.getMessage());
        e.printStackTrace();
    }
}
```

Console output:

```
Found optimal solution!
Objective = 7.0
Used items: 0 1 2 3
```

Rounding and feasibility tolerance

- Note that when printing the solution in the previous example, we used "`if (x[i].get(GRB.DoubleAttr.X) >= 0.5)`" instead of "`if (x[i].get(GRB.DoubleAttr.X) == 1)`"
- This is because a solution is deemed integer if all integer variables are within a tolerance value (`IntFeasTol`) of an integer solution
- Similarly, a solution is deemed feasible if all constraint violations are smaller than a certain tolerance value (`FeasibilityTol`)
- Finally, a solution is deemed optimal if all reduced costs are smaller than a certain tolerance value (`OptimalityTol`)
- You can set these parameters:

```
model.set(GRB.DoubleParam.FeasibilityTol, 0.000000001);
model.set(GRB.DoubleParam.OptimalityTol, 0.000000001);
model.set(GRB.DoubleParam.IntFeasTol, 0.000000001);
```

- However, only change the default value to a smaller tolerance when this is really necessary, as the runtime might increase dramatically

Matrix model

Let’s solve the Bin Packing Problem

- List of n items with size s_i
- Infinite number of bins with capacity B

The aim is to determine the minimum number of bins needed to pack the items

Matrix model

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n} y_i & \\
\text{s.t.} \quad & \sum_{i=1}^{n} s_i x_{ij} \leq B y_j & \forall j \in \{1, \dots, n\} \\
& \sum_{j=1}^{n} x_{ij} = 1 & \forall i \in \{1, \dots, n\} \\
& x_{ij} \in \mathbb{B} & \forall i, j \in \{1, \dots, n\} \\
& y_j \in \mathbb{B} & \forall j \in \{1, \dots, n\}
\end{aligned}
$$

Note: we set the number of available bins to n (why?)

Matrix model — Java code

Model creation and solving method (part 1):

```java
public static void solveMatrixModel(double[] s, double B) throws GRBException {
    // Create the environment
    GRBEnv env = new GRBEnv();
    env.start();

    // Create empty model
    GRBModel model = new GRBModel(env);

    // Get n
    int n = s.length;

    /* INPUT CHECKS HERE! */

    // Create the variables and their domain restrictions
    GRBVar[][] x = new GRBVar[n][n];
    GRBVar[] y = new GRBVar[n];
    for (int i = 0; i < n; i++) {
        y[i] = model.addVar(0, 1, 0.0, GRB.BINARY, "y(" + i + ")");
        for (int j = 0; j < n; j++) {
            x[i][j] = model.addVar(0, 1, 0.0, GRB.BINARY, "x(" + i + "," + j + ")");
        }
    }

    // Create the objective
    GRBLinExpr objExpr = new GRBLinExpr();
    for (int i = 0; i < n; i++) {
        objExpr.addTerm(1.0, y[i]);
    }
    model.setObjective(objExpr, GRB.MINIMIZE);
```

Matrix model — Java code

Model creation and solving method (part 2):

```
    ...

    // Add the capacity restrictions
    for (int j = 0; j < n; j++) {
        GRBLinExpr sizeExpr = new GRBLinExpr();
        for (int i = 0; i < n; i++) {
            sizeExpr.addTerm(s[i], x[i][j]);
        }

        GRBLinExpr sizeExprRhs = new GRBLinExpr();
        sizeExprRhs.addTerm(B, y[j]);

        model.addConstr(sizeExpr, GRB.LESS_EQUAL, sizeExprRhs, "SizeRestr(" + j + ")");
    }

    // Add the assignment restrictions
    for (int i = 0; i < n; i++) {
        GRBLinExpr assignExpr = new GRBLinExpr();
        for (int j = 0; j < n; j++) {
            assignExpr.addTerm(1.0, x[i][j]);
        }
        model.addConstr(assignExpr, GRB.EQUAL, 1.0, "AssignRestr(" + i + ")");
    }
    ...
```

Matrix model — Java code

Model creation and solving method (part 3):

```
...

// Solve the model
model.optimize();

// Query the solution
if (model.get(GRB.IntAttr.Status) == GRB.Status.OPTIMAL) {
    System.out.println("Found optimal solution!");
    System.out.println("Objective = " + model.get(GRB.DoubleAttr.ObjVal));
    System.out.print("Bin contents: ");
    int numBin = 0;
    for (int j = 0; j < n; j++) {
        if (y[j].get(GRB.DoubleAttr.X) >= 0.5) {
            System.out.print(numBin + ": { ");
            for (int i = 0; i < n; i++) {
                if (x[i][j].get(GRB.DoubleAttr.X) >= 0.5) {
                    System.out.print(i + " ");
                }
            }
            System.out.println("}");
            numBin++;
        }
    }
}
else {
    System.out.println("No optimal solution found");
}

// Close the model
model.dispose();
```

Matrix model — Java code

Main method, with hardcoded instance:

```java
public static void main(String[] args) {
    double[] s = {1, 9, 5, 3, 7, 1, 8, 3, 2};
    double B = 10;
        try {
        solveMatrixModel(s, B);
    }
    catch (GRBException e) {
        System.out.println("A Gurobi exception occured: " + e.getMessage());
        e.printStackTrace();
    }
}
```

Console output for example (there are multiple optimal solutions, all with objective value 4.0 of course):

```
Found optimal solution!
Objective = 4.0
Bin contents: 0: { 2 3 8 }
1: { 0 5 6 }
2: { 4 7 }
3: { 1 }
```

Gurobi documentation links

Gurobi documentation links

When unsure of what methods to use, always good practice to read the documentation! All text in the following list are clickable URL links.

- Main Gurobi in Java documentation
- Very useful guide with examples
- Examples for Java
- Documentation for `GRBModel`

Part 2 - Content

- Parameters and attributes
- Advanced MIP techniques
  - Warm start
  - Retrieving multiple solutions
  - Second-best solutions
- Model modifications
  - Modifying objective coefficients
  - Modifying constraints
  - Modifying variables
  - Adding variables
- Callbacks (very advanced)
- Programming assignment
- Gurobi documentation links

Parameters and attributes

Parameters and attributes

- Parameters are settings that control the behavior of the optimization solver or the modeling environment
  - Depending on the values that the parameter can take, you have
    - GRB.DoubleParam (e.g. `TimeLimit`, `FeasibilityTol`)
    - GRB.IntParam (e.g. `StartNumber`, `OutputFlag`)
    - GRB.StringParam (e.g. `ResultFile`)
  - For example, setting the maximum runtime (in seconds):

```
model.set(GRB.DoubleParam.TimeLimit, 600.0);
```

- Attributes refer to properties of the optimization model itself
  - Depending on the values that the attribute can take, you have
    - GRB.DoubleAttr (e.g. `ObjVal`, `Runtime`)
    - GRB.IntAttr (e.g. `NumStart`, `Status`)

    (- GRB.StringAttr)

    (- GRB.CharAttr)
  - For example, getting the objective value of the current model:

```
model.get(GRB.DoubleAttr.ObjVal);
```

  - Variables/expressions also have attributes (`GRB.CharAttr.VType`, `GRB.DoubleAttr.VarName`, `GRB.DoubleAttr.Slack`)

Advanced MIP techniques

Advanced MIP techniques — warm start

- In future assignments, MIP models can take a really long time
- In heavily constrained problems, much time is spent finding a solution
- Often, heuristics can find a good initial solution quickly
- How to give this to gurobi? $\rightarrow$ warm start
- What a warm start does:
  - Gives a good UB/LB to prune bad branches more quickly
  - Saves time for finding any solution
  - Guides branch-and-bound search (gurobi magic)

Advanced MIP techniques – warm start

- To add a warm start, you need to set the value of each of your variables by setting the `Start` attribute on the variables
- Given a variable `var` and corresponding MIP start value `val`

```
var.set(GRB.DoubleAttr.Start, val);
```

- For example, adding a warm start with `vars` as variable array and `vals` as value array

```
for (int i = 0; i < vars.length; i++) {
    vars[i].set(GRB.DoubleAttr.Start, vals[i]);
}
```

- You can also provide a partial warm start (i.e. specifying the value for only part of the variables) - the MIP solver will attempt to fill in values for missing start values

Advanced MIP techniques – warm start

- You can provide multiple MIP starts (Gurobi will only use the one with the best objective value so you usually provide only one)
- You need attribute `NumStart` and parameter `StartNumber`
  - `NumStart`: number of MIP starts in the model
  - `StartNumber`: index of the MIP start that you are currently working with, is 0 by default and can take values `0,1,...,NumStart-1`
- Suppose you have an array of variables `vars` and two feasible solutions `vals1` and `vals2`, then you can add two MIP starts:

```java
model.set(GRB.IntAttr.NumStart, 2);
for (int i = 0; i < vars.length; i++) {
    vars[i].set(GRB.DoubleAttr.Start, vals1[i]);
}
model.set(GRB.IntParam.StartNumber, 1);
for (int i = 0; i < vars.length; i++) {
    vars[i].set(GRB.DoubleAttr.Start, vals2[i]);
}
```

Advanced MIP techniques — multiple solutions

- In IPs especially, multiple optimal solutions are not uncommon
- Sometimes, decision makers might have hard-to-express preferences
- Or, you just want to know whether the solution is unique
- Use gurobi's solution pool feature
    - Used to enumerate several solutions to a MIP model
    - Objective within specified tolerance
    - Maximum number of generated solutions

Advanced MIP techniques — multiple solutions

- Set some solution pool parameters to find all optimal solutions:

```
model.set(GRB.DoubleParam.PoolGap, 0);
model.set(GRB.IntParam.PoolSearchMode, 2);
```

- `PoolSearchMode`
  - is 0 by default (find one optimal solution),
  - should be set to 1 to find more solutions, but with no guarantee on the quality, and
  - should be set to 2 to find the n best solutions.
- Add another parameter to find at most 25 solutions:

```
model.set(GRB.IntParam.PoolSolutions, 25);
```

- Retrieving the solutions:
  - Get number of found solutions

    ```
    model.get(GRB.IntAttr.SolCount)
    ```

  - Get objective of solution `i` and value of `GRBVar x` in solution `i`

    ```
    model.set(GRB.IntParam.SolutionNumber, i-1);
    obj[i-1] = model.get(GRB.DoubleAttr.PoolObjVal)
    xval[i-1] = x.get(GRB.DoubleAttr.Xn)
    ```

- Be careful: can slow down the solving process

Advanced MIP techniques — second best solutions

- If you do not like the “black-box-ness” of the solution pool, you can find second-best solutions manually
- Achieved by manually <u>cutting off solutions</u>
- Example:
  - Model with only binary parameters x_i, $i \in \{1, \ldots, n\}$
  - First run of model found solution $(x_1^*, \ldots, x_n^*)$
  - Add cut
$$\sum_{i:x_i^*=0} (1 - x_i) + \sum_{i:x_i^*=1} x_i \leq n - 1$$
  - Resolving model gives <u>second-best</u> solution
- For general integer solution, more complicated techniques exist
- With continuous variables, even more complicated due to continuum of solutions

Model modifications

Model modifications

- Sometimes you want to update your model after solving
- Modifications can include:
  - Modifying objective coefficients
  - Modifying constraints
  - Adding variables
  - Modifying variables
- Prominent techniques that feature model modifications are:
  - Branch-and-cut
  - Column generation (branch-price-and-cut)
  - Dantzig-Wolfe decomposition
  - Bender's decomposition
  - (integer) L-shaped method

Model modifications — modifying objective

- Set objective coefficient of `GRBVar x` to `double c`

```
x.set(GRB.DoubleAttr.Obj, c);
```

- Change the objective to a new `GRBLinExpr obj`

```
model.setObjective(obj);
```

Model modifications — modifying constraints

- You need the “constraint” object, `GRBConstr`, for modifications
- First, store the constraint (similar to variables) when making them:

```
GRBConstr constr = model.addConstr(lhs, GRB.
    GREATER_EQUAL, rhs, "constraint");
```

- Set coefficient of `GRBVar x` in `GRBConstr constr` to `double c`

```
model.chgCoeff(constr, x, c);
```

- Adding extra constraints is the same as “normal” constraints
- Removing constraint `GRBConstr constr`

```
model.remove(constr);
```

Model modifications — adding variables

- You can either first add a new variable the way you have seen before, by specifying only the bounds and type, and then add the new variable to the objective and constraints by changing the objective and constraints as explained on the previous slides

- Or you can directly specify the objective coefficient (and constraint coefficients) when defining the new variable
  - First, create an array of constraints `GRBConstr[] constr` to which the new variable needs to be added
  - Next, create an array of coefficients `double[] constrCoeffs` of the variable in each of these constraints
  - Finally, add new variable `x` with objective coefficient `double c`

```java
GRBVar x = model.addVar(0, 1, c, GRB.BINARY, constr,
     constrCoeffs, "new variable");
```

Model modifications — modifying variables

- Setting upper/lower bound of `GRBVar x` to `double ub/lb`:

```
x.set(GRB.DoubleAttr.LB, lb);
x.set(GRB.DoulbeAttr.UB, ub);
```

- Relaxing integer/binary variable `GRBVar x` to a continuous variable

```
x.set(GRB.CharAttr.VType, GRB.CONTINUOUS);
```

- Create a new model which is a relaxation of the old `GRBModel model` (relaxing all integer/binary variables)

```
GRBModel relaxedModel = model.relax();
```

Callbacks — Advanced!

Callbacks

- All model modifications discussed happen after solving the model
- Can be quite inefficient to re-explore the whole branch-and-bound tree
- You can modify the model during solving using callbacks
- Note: you cannot add variables during solving
- Callbacks can let you do so much more, including:
  - Determine whether to branch on or prune some branch-and-bound node
  - Add cuts at integer or fractional solutions
  - Query, modify or reject potential best known solutions
  - Insert your own heuristics for quickly finding integer solutions
- Basically, they let you modify the inner workings of gurobi!

Callbacks

Gurobi callbacks make use of a pair of arguments: `where` and `what`.

- `where`: indicates from where in the Gurobi optimizer the callback is being called (presolve, simplex, MIP, etc.), has possible values
  - `GRB.CB_POLLING`: periodic polling callback, only invoked if other callbacks have not been called in a while
  - `GRB.CB_PRESOLVE`: currently performing presolve
  - `GRB.CB_SIMPLEX`: currently in simplex
  - `GRB.CB_MIPSOL`: found a new MIP solution
  - `GRB.CB_MIPNODE`: currently exploring a MIP node
  - ...
  - You can carry out different actions depending on the value of `where` (e.g. using `if(where == GRB.CB_MIPSOL)`)

Callbacks

Gurobi callbacks make use of a pair of arguments: `where` and `what`.

- `what`: can be used to obtain more detailed information about the state of the optimization; can be passed to `getDoubleInfo`, `getIntInfo` or `getStringInfo` method.
  Different possibilities for each of the different values of `where`:
  - `PRESOLVE`
    - `getIntInfo(GRB.CB_PRE_COLDEL)`: the number of columns removed by presolve to this point
    - `getIntInfo(GRB.CB_PRE_ROWDEL)`: the number of rows removed by presolve to this point
  - `MIPSOL`
    - `getDoubleInfo(GRB.CB_MIPSOL_OBJ)`: objective value for new solution
    - `getIntInfo(GRB.CB_MIPSOL_SOLCNT)`: current count of feasible solutions found

  ...

Callbacks

Gurobi callbacks make use of a pair of arguments: `where` and `what`.

- `what`: can be used to obtain more detailed information about the state of the optimization; can be passed to `getDoubleInfo`, `getIntInfo` or `getStringInfo` method.
  Different possibilities for each of the different values of `where`:
  ...
  - `MIPNODE`
    - `getIntInfo(GRB.CB_MIPNODE_STATUS)`: optimization status of current MIP node
    - `getIntInfo(GRB.CB_MIPNODE_SOLCNT)`: current count of feasible solutions found
  - Except for when in `POLLING`, you can obtain the elapsed solver runtime (in seconds) by calling `getDoubleInfo(GRB.CB_RUNTIME)`
  - And many more!!

Callbacks

Some examples of things you can do using callbacks

- You can terminate optimization early using `model.terminate()`
- You can retrieve values from the node relaxation solution at the current node using `getNodeRel(vars)`
  - only possible when `where` is equal to `GRB.CB_MIPNODE` and `GRB.CB_MIPNODE_STATUS` is equal to `GRB.OPTIMAL`
- You can retrieve values from the new MIP solution using `getSolution(vars)`
  - only possible when `where` is equal to `GRB.CB_MIPSOL`
- After retrieving values, you can add a constraint to cut off that solution using `addCut(lhs,sense,rhs)` with `lhs` and `rhs` `GRBLinExpr`'s and `sense` for example `GRB.LESS_EQUAL`
  - only possible when `where` is equal to `GRB.CB_MIPNODE`
- You can also use the retrieved values to build a heuristic solution, and import this solution using `setSolution(vars,sol)`
  - only possible when `where` is equal to `GRB.CB_MIPNODE`, `GRB.CB_MIPSOL`, or `GRB.CB_MIP`

Callbacks

Example: callback that prints every new MIP solution found. Callback class

```java
public class ExampleCallback extends GRBCallback {
  private GRBVar[] vars;

  public ExampleCallback(GRBVar[] xvars) {
    this.vars = xvars;
  }

  protected void callback() {
    try {
      if (where == GRB.CB_MIPSOL) { // MIP solution callback: new MIP solution was found
        // Retrieve the current node count and count of feasible solutions found
        int nodeCount = (int) getDoubleInfo(GRB.CB_MIPSOL_NODCNT);
        int solCount = getIntInfo(GRB.CB_MIPSOL_SOLCNT);

        // Retrieve the objective value and variable values of the new solution
        double obj = getDoubleInfo(GRB.CB_MIPSOL_OBJ);
        double[] x = getSolution(vars);

        // Print the solution information
        System.out.println("New solution (number "+ solCount + ") at node " + nodeCount
                    + " with objective " + obj+ ":");
        for (int i = 0; i < x.length; i++) {
          System.out.println("x["+i+"]="+x[i]);
        }
      }
    } catch (GRBException e) {
      System.out.println(e.getMessage());
      e.printStackTrace();
    }
```

Callbacks

Example: callback that prints every new MIP solution found.
Implementation in Vector Model class

```
  public static void main(String[] args) throws GRBException{
    double[] w = { 1, 2, 5, 1, 3, 1 };
    double[] p = { 1, 1, 2, 3, 3, 4 };
    double b = 10;
    int[][] G = { { 2, 0 }, { 2, 1 }, { 4, 2 }, { 3, 1 }, { 5, 0 }, { 5, 4 } };

    // Create the environment
    GRBEnv env = new GRBEnv();
    env.set(GRB.IntParam.OutputFlag, 0);
    env.start();

    // Create empty model
    GRBModel model = new GRBModel(env);

    // Get n
    int n = w.length;

    // Create the variables and their domain restrictions
    GRBVar[] x = new GRBVar[n];
    for (int i = 0; i < n; i++) {
        x[i] = model.addVar(0, 1, 0.0, GRB.BINARY, "x(" + i + ")");
    }

    /* Create the objective and constraints */
...
```

Callbacks

Example: callback that prints every new MIP solution found.
Implementation in Vector Model class

```java
...
    // Create callback
    ExampleCallback callback = new ExampleCallback(x);

    model.setCallback(callback);

    // Solve the model
    model.optimize();

    // Query the solution
    if (model.get(GRB.IntAttr.Status) == GRB.Status.OPTIMAL) {
        System.out.println("Found optimal solution!");
        System.out.println("Objective = " + model.get(GRB.DoubleAttr.ObjVal));
        System.out.print("Used items: ");
        for (int i = 0; i < n; i++) {
            if (x[i].get(GRB.DoubleAttr.X) >= 0.5) {
                System.out.print(i + " ");
            }
        }
    } else {
        System.out.println("No optimal solution found");
    }

    // Close the model
    model.dispose();
    env.dispose();
}
```

Callbacks

Closing remarks on callbacks

- Extremely powerful <u>and</u> dangerous
- You can easily mislead output by tinkering with the branch-and-bound
- You can do much more than I showed you
- Given these pointers, check out the manual yourself
- Investigate what methods are available to you, should you ever need them

Gurobi documentation links

Gurobi documentation links

When unsure of what methods to use, always good practice to read the documentation! All text in the following list are clickable URL links.

- Main Gurobi in Java documentation
- Very useful guide with examples

- Documentation for Callbacks
- Documentation for `GRBCallback`, the parent class of all callbacks
- Documentation with more information about the `where` and `what` arguments of Gurobi callback routines