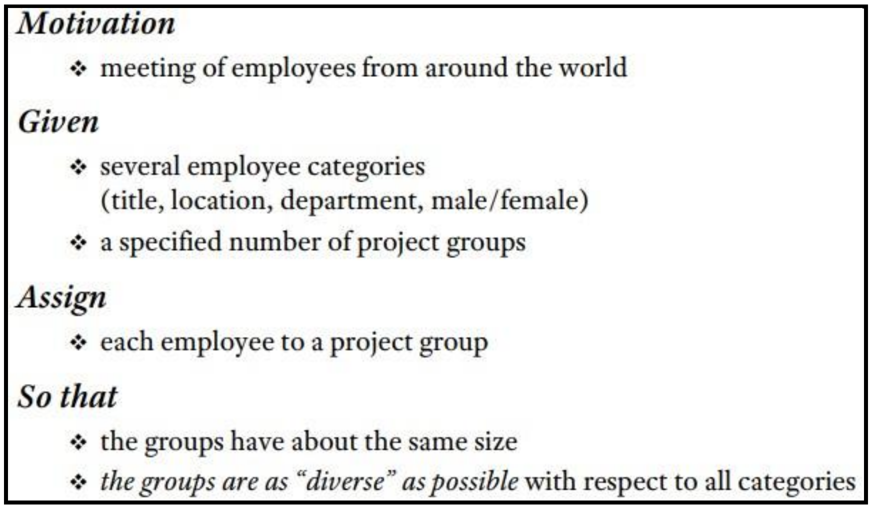
Building Decision Models for DMCommunity.org

Challenge “Balanced Assignment”

Decision Management Community Sep-2018 Challenge “Balanced Assignment” gives an example of a complex business problem with a serious optimization component. This problem was offered by Prof. Robert Fourer and it deals with the assignment of people to different project groups. Usually, such problems require deep knowledge of optimization techniques. My interest was to build a decision model for this problem and to investigate what can be done by business people and where the involvement of optimization experts is necessary. So, I attempted to use a business-friendly approach to represent and to solve this complex problem. It was a long journey, and below I will describe what I did successfully and where I failed.

Table of Contents

Problem Formulation



Motivation

- meeting of employees from around the world

Given

- several employee categories
  (title, location, department, male/female)
- a specified number of project groups

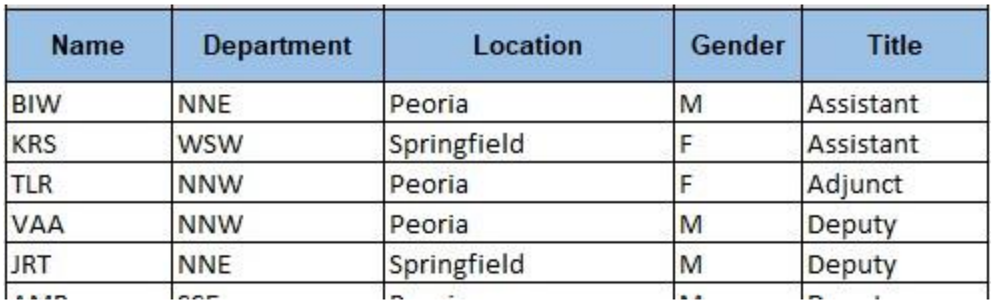
Assign

- each employee to a project group

So that

- the groups have about the same size
- *the groups are as "diverse" as possible* with respect to all categories

Prof. Fourer kindly provided test data described in the file BalancedAssignment.xls:



Name	Department	Location	Gender	Title
BIW	NNE	Peoria	M	Assistant
KRS	WSW	Springfield	F	Assistant
TLR	NNW	Peoria	F	Adjunct
VAA	NNW	Peoria	M	Deputy
JRT	NNE	Springfield	M	Deputy

This example contains 210 people who should be assigned to 12 groups of 16, 17, 18, or 19 people in every group. Each person may be assigned to only one group. The above "diversity" constraint means that it is preferable to assign people with the same categories (department, location, gender, title) to different groups.

Problem Modeling

The main goal of our decision model is to assign all people to all groups in such a way that minimizes what Prof. Fourer called the "Total Sameness" as a measure of the "diversity" of all groups. However, I didn't want to follow Prof. Fourer's definition of the "Total Sameness" as I was afraid it might look too "scientific" for a regular business person. So, I asked myself a question: what would be an intuitive way for a business analyst to define the Total Sameness?

My first impulse approach was a two-step process:

1. *Define penalties when two people with similar characteristics being assigned to the same group*
2. *Find such an assignment of people to groups that minimizes the total penalty.*

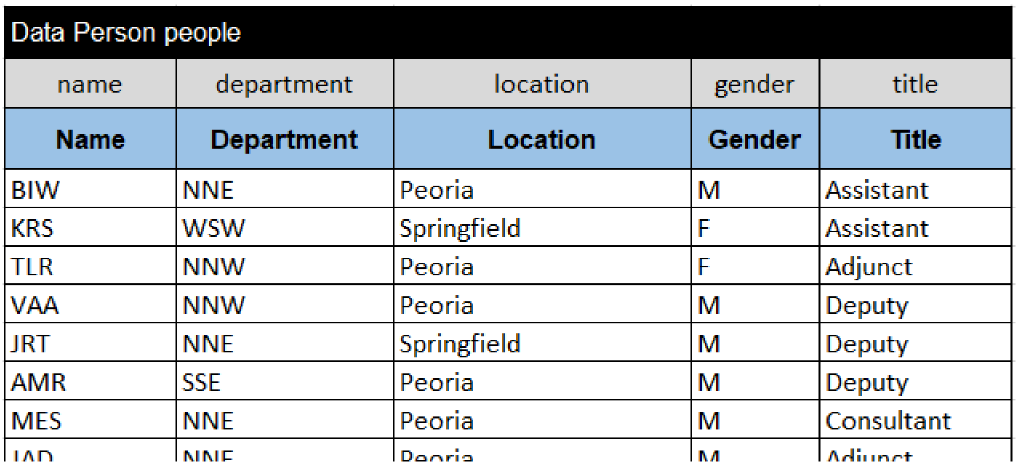
I believe that a business person would like to define penalties in a flexible way that will allow her (or him) to control the behavior of the entire model (Step 1). However, she would probably prefer not to deal with the optimization part (Step 2) at all and let a technical specialist take care of it.

To implement these two steps I decided to apply an integrated Business Rules and Optimization approach that I've already successfully used for much simpler optimization problems (see Vacation Days and Flight Rebooking). This approach splits the problem representation into two parts:

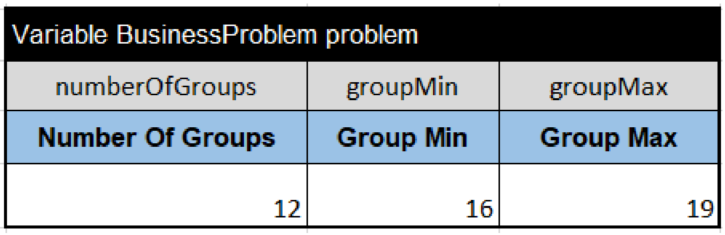
1) **Business problem**: specifies pure business requirements, create a business decision model that will prepare necessary data for the optimization part;
2) **Optimization problem**: defines optimization variables and constraints and uses off-the-shelf constraint or linear solver to find the optimal solution of the problem.

Input Data

We will keep test data for this problem in the following tables of the file "BalancedAssignmentData.xls":



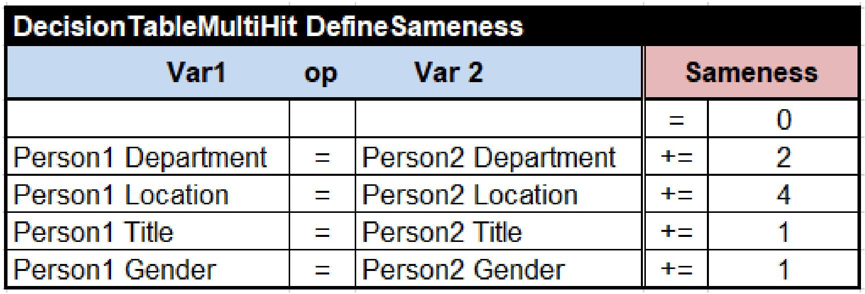
Data Person people				
name	department	location	gender	title
Name	**Department**	**Location**	**Gender**	**Title**
BIW	NNE	Peoria	M	Assistant
KRS	WSW	Springfield	F	Assistant
TLR	NNW	Peoria	F	Adjunct
VAA	NNW	Peoria	M	Deputy
JRT	NNE	Springfield	M	Deputy
AMR	SSE	Peoria	M	Deputy
MES	NNE	Peoria	M	Consultant
JAD	NNE	Peoria	M	Adjunct



Variable BusinessProblem problem		
numberOfGroups	groupMin	groupMax
Number Of Groups	**Group Min**	**Group Max**
12	16	19

Modeling Business Problem

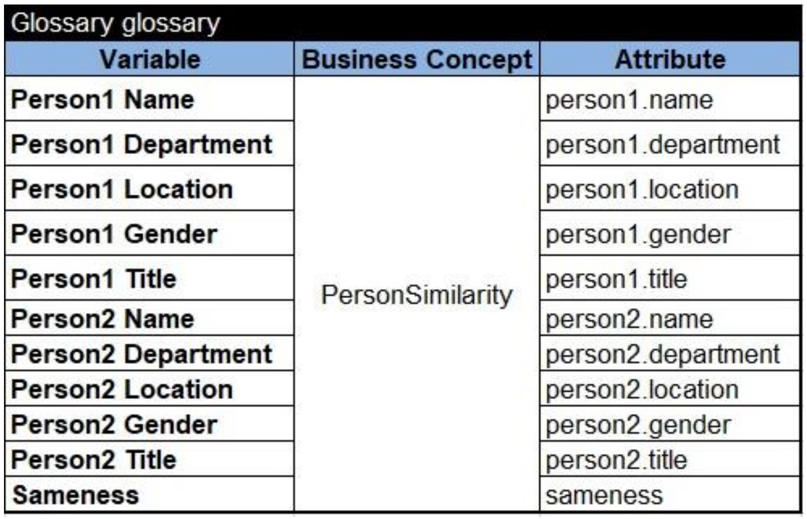
I believe it would be quite natural for a business person to define the business term "Sameness" for any two people in the same group using the following decision table:



DecisionTableMultiHit DefineSameness				
Var1	op	Var 2	Sameness	
			=	0
Person1 Department	=	Person2 Department	+=	2
Person1 Location	=	Person2 Location	+=	4
Person1 Title	=	Person2 Title	+=	1
Person1 Gender	=	Person2 Gender	+=	1

This table accumulates Sameness penalty for Person1 and Person2 adding different values when they belong to the same department, come from the same location, or have the same gender and/or title. A business person can easily understand and change this sameness definition to give different "importance" to different types of sameness. For example, if she wants "Sameness" to be equal to the number of all similarities, she may make all numbers in the column "Sameness" to be equal to 1.

As usual for OpenRules-based decision model, we may define the proper Glossary as the following Excel's table:



Glossary glossary		
Variable	Business Concept	Attribute
Person1 Name	PersonSimilarity	person1.name
Person1 Department		person1.department
Person1 Location		person1.location
Person1 Gender		person1.gender
Person1 Title		person1.title
Person2 Name		person2.name
Person2 Department		person2.department
Person2 Location		person2.location
Person2 Gender		person2.gender
Person2 Title		person2.title
Sameness		sameness

To complete this decision model and to provide data for its optimization counterpart, I needed to define the following Java beans:

Person:

```
        String   name;
        int      number;
        String   department;
        String   location;
        String   gender;
        String   title;
        int      group;
```

Group:

```
        String   id;
```

```
int        size;
Person[] people;
```

Similarity:

```
Person  person1;
Person  person2;
int        sameness;
```

BusinessProblem:

```
Person[] people;
Group[]   groups;
int        numberOfGroups;
int        groupMin;
int        groupMax;
ArrayList<Similarity> similarities;
```

We can read the input data from the file "BalancedAssignmentData.xls" and use it to create an instance of the class BusinessProblem using the following Java snippet:

```
DecisionData data = new DecisionData("file:rules/BalancedAssignmentData.xls");
BusinessProblem problem = (BusinessProblem) data.get("getProblem");
Person[] people = (Person[]) data.get("getPeople");
```

To create an array of similarities for every pair of people, we can use the following BusinessProblem's method:

```
public ArrayList<Similarity> defineSimilarities(String filename,String decisionName) {
    // define sameness for all pairs of people
    Decision decision = new Decision(decisionName,filename);
    similarities = new ArrayList<Similarity>();
    for (int p1 = 0; p1 < people.length-1; p1++) {
        for (int p2 = p1+1; p2 < people.length; p2++) {
            Similarity similarity = new Similarity();
            similarity.person1 = people[p1];
            similarity.person2 = people[p2];
            decision.put("PersonSimilarity", similarity);
            decision.execute();
            if (similarity.sameness > 0) {
                similarities.add(similarity);
            }
        }
    }
    return similarities;
}
```

This method executes our decision table "DefineSameness" (defined by the parameter "decisionName") for every unique pair of people. After the execution of this business decision model, our object

BusinessProblem will know all people and their similarities. The Optimization problem will use this information to assign people to groups while minimizing Total Sameness.

Modeling Optimization Problem

I could minimize the use of Java code when I modeled the Business Problem above by moving two loops over people directly to Excel table. However, I will assume that a reader of this section is a software developer familiar with Java. OpenRules provides a new Java class Optimization that is a based class for different implementation of optimization problems. It's based on the JSR-331 Constraint Programming API and allow a user to define only two declarative methods:

- define() – that usually defines 1) constrained variables; 2) major constraints; 3) optimization objective. They can be defined using an already embedded instance "csp" of a JSR331 constraint satisfaction problem
- saveSolution(Solution solution) – that will be automatically called be the predefined generic method minimize() or maximize() if an optimal solution is found. This method is supposed solution's variables back to the objects defined in the business problem.

So, to implement the optimization problem I need to create a Java class OptimizationProblem inherited from the standard class Optimization.

First Implementation of the Optimization Problem using a Constraint Solver

My very first attempt to write class OptimizationProblem was based on the hope that the default JSR331 constraint solver will be able to solve the problem. Here is the method "define":

```
public void define() { // PROBLEM DEFINITION
    // Define assignment variables
    for (int g = 0; g < groups.length; g++) {
        Var[] peopleGroupVars = new Var[people.length];
        for (int p = 0; p < people.length; p++) {
            peopleGroupVars[p] = csp.variable("G"+g+"P"+p, 0, 1);
        }
        // Post min-max constraints for group sizes
        Var groupSizeVar = csp.sum("Size"+g,peopleGroupVars);
        csp.post(groupSizeVar,">=",businessProblem.getGroupMin());
        csp.post(groupSizeVar,"<=",businessProblem.getGroupMax());
    }
    // Post constraints: each person belongs to one and only one group
    for (int p = 0; p < people.length; p++) {
        Var[] personVarsInAllGroups = new Var[groups.length];
        for (int g = 0; g < groups.length; g++) {
            personVarsInAllGroups[g] = csp.getVar("G"+g+"P"+p);
        }
        Var personOccurancesInAllGroups = csp.sum("P"+p,personVarsInAllGroups);
        csp.post(personOccurancesInAllGroups,"=",1);
    }
    // Define penalty variables
    ArrayList<Var> penaltyVars = new ArrayList<Var>();
    int n = 0;
    for (int g = 0; g < groups.length; g++) {
        for (int p1 = 0; p1 < people.length-1; p1++) {
            for (int p2 = p1+1; p2 < people.length; p2++) {
                int sameness = businessProblem.getSameness(p1,p2);
                if (sameness > 0) {
                    Var var1 = csp.getVar("G"+g+"P"+p1);
                    Var var2 = csp.getVar("G"+g+"P"+p2);
                    Var penaltyVar = var1.multiply(var2).multiply(sameness);
                    csp.add("Penalty"+n,penaltyVar);
                    penaltyVars.add(penaltyVar);
                    n++;
                }
            }
        }
    }
    // Define optimization objective
    Var totalPenalty = csp.sum(penaltyVars);
    csp.add("TotalPenalty",totalPenalty);
    setObjective(totalPenalty);
}
```

This method does the following:

1) Defines assignment variables:
    a. for each group g and for each person p it defines one constrained integer variable *peopleGroupVars[p] = csp.variable("G"+g+"P"+p, 0, 1);* `Each variable can be 1 if a person p assigned to group g or 0 otherwise;`

    b. then it defines *groupSizeVar* as a sum of all *peopleGroupVars* and limits it to *groupMin* and *groupMax*;
2) Posts constraints “each person belongs to one and only one group”:
    a. for every person it defines an array of *personVarsInAllGroups* and states that the sum of these variables should be 1
3) Defines penalty variables
    a. For each unique pair of people (p1;p2) if finds its sameness pre-calculated by the BusinessProblem. If such sameness > 0 it calculates *Var penaltyVar = var1.multiply(var2).multiply(sameness);* and adds it to the array *penaltyVars*.
    b. The sum of the array specifies our optimization objective “TotalPenalty” that we want to minimize.

Here is the method “saveSolution”:

```
public void saveSolution(Solution solution) {
    for (int g = 0; g < groups.length; g++) {
        Group group = groups[g];
        for (int p = 0; p < people.length; p++) {
            Person person = people[p];
            int value = solution.getValue("G"+g+"P"+p);
            if (value == 1) {
                group.add(person);
            }
        }
    }
    csp.log("TotalPenalty: " + solution.getValue("TotalPenalty"));
}
```

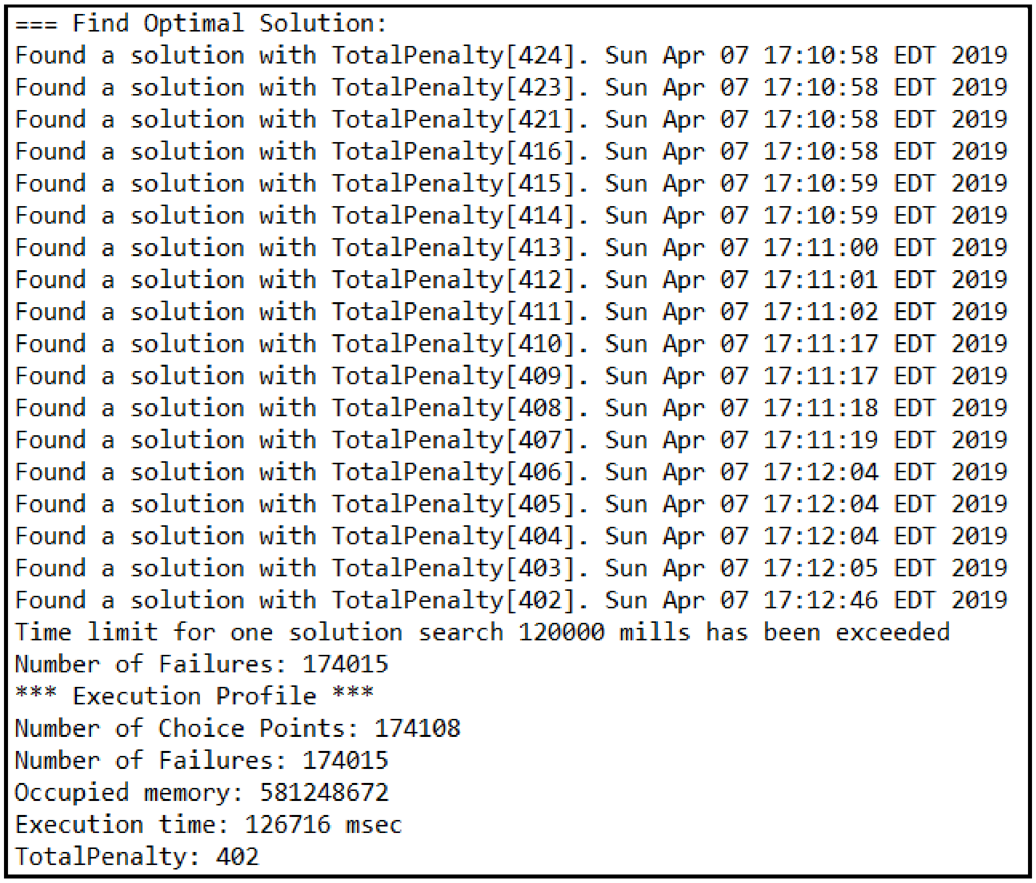
This method will be automatically called by the standard Optimization’s method *minimize()* when a solution is found. For each group g and for each person p, it adds the person p to the group g if the values of its assignment variable with the name *“G”+g+”P”+p* inside the solution is equal to 1.

To run my BusinessProblem and then this OptimizationProblem I used this simple main method:

```
public static void main(String[] args) {
    // Create Business Problem
    DecisionData data = new DecisionData("file:rules/BalancedAssignmentData.xls");
    BusinessProblem problem = (BusinessProblem) data.get("getProblem");
    Person[] people = (Person[]) data.get("getPeople");
    problem.initialize(people);
    problem.defineSimilarities("file:rules/BalancedAssignment.xls", "DefineSameness");
    // Create and execute Optimization Problem
    OptimizationProblem optimization = new OptimizationProblem(problem);
    optimization.setTimeLimit(120);
    optimization.define();
    optimization.minimize();
    problem.showSolution();
}
```

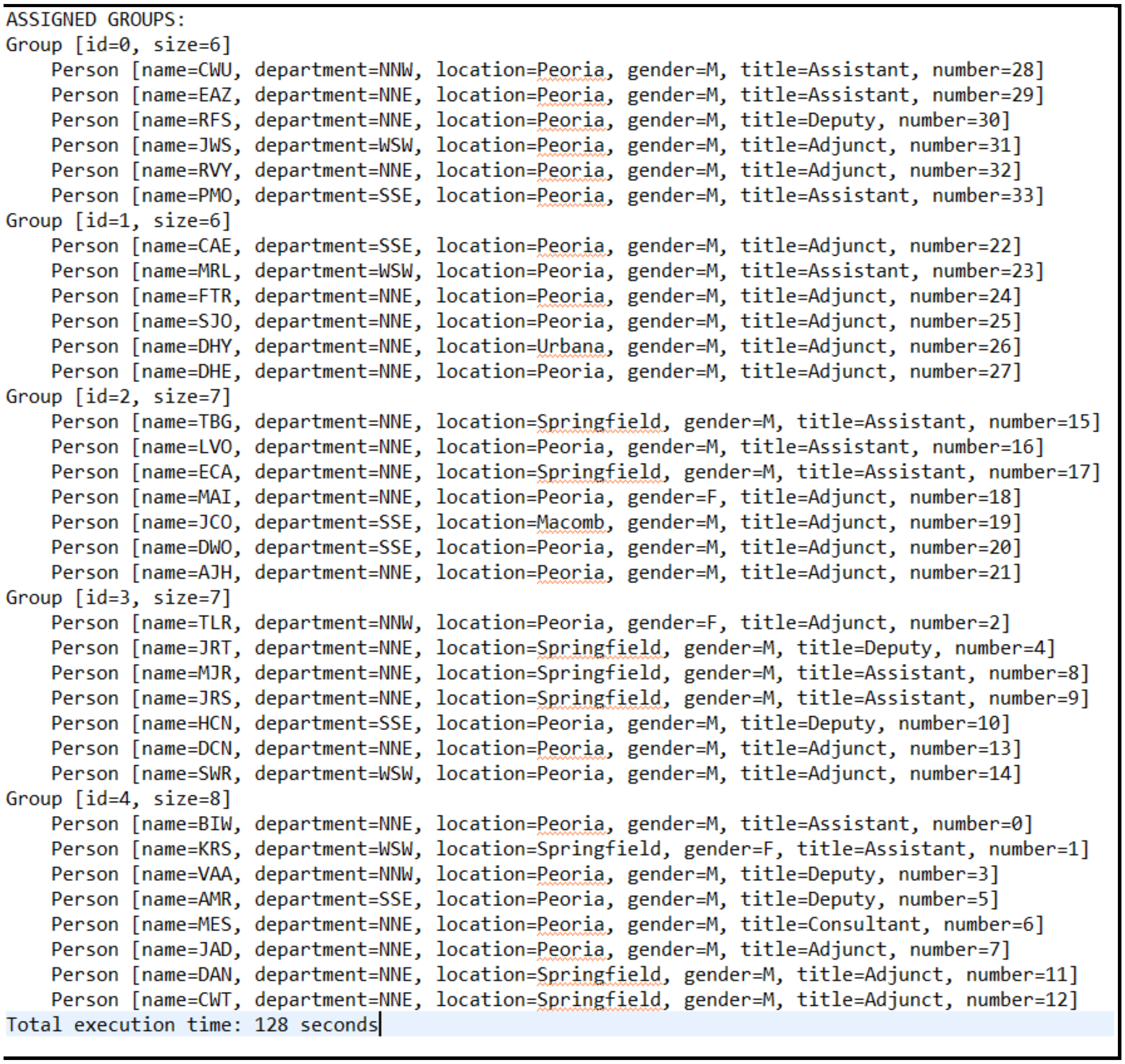
When I tried to run this program against the provided Excel data file with 210 people, it relatively quickly ran out of memory. Adding more memory did not help – it just took more time with the same results.

Apparently, the problem is too big for the default constraint solver (I used an open source Constrainer). So, to check if my implementation can handle the same problem but with a smaller number of people, I limited my array of people in the xls-file to the first 34 people and set Number of Groups to 5, Group Min to 6 and Group Max to 9. When I ran this problem I quickly (within 10 seconds) receive the first solution with the TotalPenalty[424]. Then it started to produce better solutions until it reached TotalPenalty[402]. Then it continued to work "forever". So, I set the time limit to 120 seconds, and the optimization process was interrupted considering the last found solution as an optimal one found within 120 seconds. Here is the execution protocol:



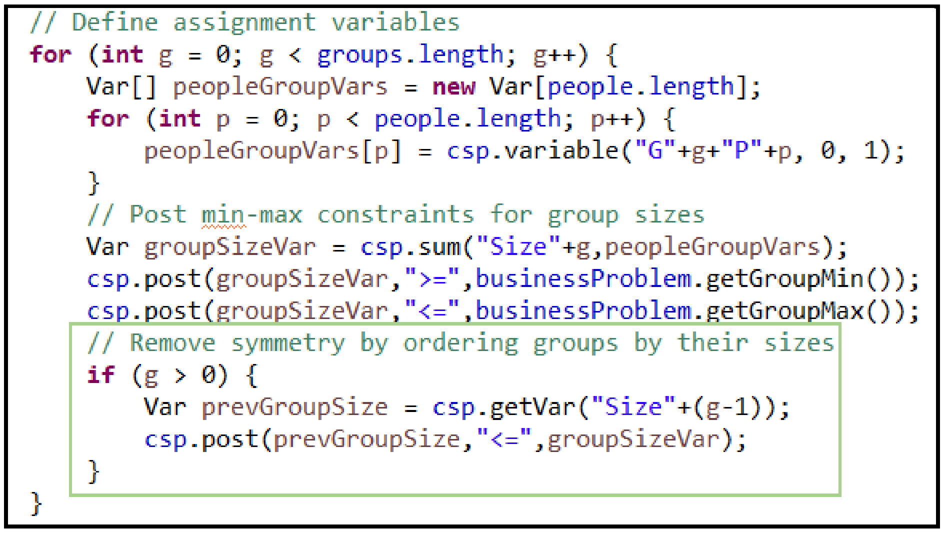
```
=== Find Optimal Solution:
Found a solution with TotalPenalty[424]. Sun Apr 07 17:10:58 EDT 2019
Found a solution with TotalPenalty[423]. Sun Apr 07 17:10:58 EDT 2019
Found a solution with TotalPenalty[421]. Sun Apr 07 17:10:58 EDT 2019
Found a solution with TotalPenalty[416]. Sun Apr 07 17:10:58 EDT 2019
Found a solution with TotalPenalty[415]. Sun Apr 07 17:10:59 EDT 2019
Found a solution with TotalPenalty[414]. Sun Apr 07 17:10:59 EDT 2019
Found a solution with TotalPenalty[413]. Sun Apr 07 17:11:00 EDT 2019
Found a solution with TotalPenalty[412]. Sun Apr 07 17:11:01 EDT 2019
Found a solution with TotalPenalty[411]. Sun Apr 07 17:11:02 EDT 2019
Found a solution with TotalPenalty[410]. Sun Apr 07 17:11:17 EDT 2019
Found a solution with TotalPenalty[409]. Sun Apr 07 17:11:17 EDT 2019
Found a solution with TotalPenalty[408]. Sun Apr 07 17:11:18 EDT 2019
Found a solution with TotalPenalty[407]. Sun Apr 07 17:11:19 EDT 2019
Found a solution with TotalPenalty[406]. Sun Apr 07 17:12:04 EDT 2019
Found a solution with TotalPenalty[405]. Sun Apr 07 17:12:04 EDT 2019
Found a solution with TotalPenalty[404]. Sun Apr 07 17:12:04 EDT 2019
Found a solution with TotalPenalty[403]. Sun Apr 07 17:12:05 EDT 2019
Found a solution with TotalPenalty[402]. Sun Apr 07 17:12:46 EDT 2019
Time limit for one solution search 120000 mills has been exceeded
Number of Failures: 174015
*** Execution Profile ***
Number of Choice Points: 174108
Number of Failures: 174015
Occupied memory: 581248672
Execution time: 126716 msec
TotalPenalty: 402
```

The produced "optimal" solution (or better to say the best solution found within 120 seconds) was printed as follows:



```
ASSIGNED GROUPS:
Group [id=0, size=6]
    Person [name=CWU, department=NNW, location=Peoria, gender=M, title=Assistant, number=28]
    Person [name=EAZ, department=NNE, location=Peoria, gender=M, title=Assistant, number=29]
    Person [name=RFS, department=NNE, location=Peoria, gender=M, title=Deputy, number=30]
    Person [name=JWS, department=WSW, location=Peoria, gender=M, title=Adjunct, number=31]
    Person [name=RVY, department=NNE, location=Peoria, gender=M, title=Adjunct, number=32]
    Person [name=PMO, department=SSE, location=Peoria, gender=M, title=Assistant, number=33]
Group [id=1, size=6]
    Person [name=CAE, department=SSE, location=Peoria, gender=M, title=Adjunct, number=22]
    Person [name=MRL, department=WSW, location=Peoria, gender=M, title=Assistant, number=23]
    Person [name=FTR, department=NNE, location=Peoria, gender=M, title=Adjunct, number=24]
    Person [name=SJO, department=NNE, location=Peoria, gender=M, title=Adjunct, number=25]
    Person [name=DHY, department=NNE, location=Urbana, gender=M, title=Adjunct, number=26]
    Person [name=DHE, department=NNE, location=Peoria, gender=M, title=Adjunct, number=27]
Group [id=2, size=7]
    Person [name=TBG, department=NNE, location=Springfield, gender=M, title=Assistant, number=15]
    Person [name=LVO, department=NNE, location=Peoria, gender=M, title=Assistant, number=16]
    Person [name=ECA, department=NNE, location=Springfield, gender=M, title=Assistant, number=17]
    Person [name=MAI, department=NNE, location=Peoria, gender=F, title=Adjunct, number=18]
    Person [name=JCO, department=SSE, location=Macomb, gender=M, title=Adjunct, number=19]
    Person [name=DWO, department=SSE, location=Peoria, gender=M, title=Adjunct, number=20]
    Person [name=AJH, department=NNE, location=Peoria, gender=M, title=Adjunct, number=21]
Group [id=3, size=7]
    Person [name=TLR, department=NNW, location=Peoria, gender=F, title=Adjunct, number=2]
    Person [name=JRT, department=NNE, location=Springfield, gender=M, title=Deputy, number=4]
    Person [name=MJR, department=NNE, location=Springfield, gender=M, title=Assistant, number=8]
    Person [name=JRS, department=NNE, location=Springfield, gender=M, title=Assistant, number=9]
    Person [name=HCN, department=SSE, location=Peoria, gender=M, title=Deputy, number=10]
    Person [name=DCN, department=NNE, location=Peoria, gender=M, title=Adjunct, number=13]
    Person [name=SWR, department=WSW, location=Peoria, gender=M, title=Adjunct, number=14]
Group [id=4, size=8]
    Person [name=BIW, department=NNE, location=Peoria, gender=M, title=Assistant, number=0]
    Person [name=KRS, department=WSW, location=Springfield, gender=F, title=Assistant, number=1]
    Person [name=VAA, department=NNW, location=Peoria, gender=M, title=Deputy, number=3]
    Person [name=AMR, department=SSE, location=Peoria, gender=M, title=Deputy, number=5]
    Person [name=MES, department=NNE, location=Peoria, gender=M, title=Consultant, number=6]
    Person [name=JAD, department=NNE, location=Peoria, gender=M, title=Adjunct, number=7]
    Person [name=DAN, department=NNE, location=Springfield, gender=M, title=Adjunct, number=11]
    Person [name=CWT, department=NNE, location=Springfield, gender=M, title=Adjunct, number=12]
Total execution time: 128 seconds
```

So, for a smaller size of the problem, this implementation can find a solution but we do not know it is optimal or not. I tried to improve help the constraint solver by removing some obvious symmetries. For example, if we exchange Group 1 and Group 2 it would be exactly the same solution but the solver considers them to be different. So, I decided to order the groups by their size and added the following code (see the green rectangle):



```java
// Define assignment variables
for (int g = 0; g < groups.length; g++) {
    Var[] peopleGroupVars = new Var[people.length];
    for (int p = 0; p < people.length; p++) {
        peopleGroupVars[p] = csp.variable("G"+g+"P"+p, 0, 1);
    }
    // Post min-max constraints for group sizes
    Var groupSizeVar = csp.sum("Size"+g,peopleGroupVars);
    csp.post(groupSizeVar,">=",businessProblem.getGroupMin());
    csp.post(groupSizeVar,"<=",businessProblem.getGroupMax());
    // Remove symmetry by ordering groups by their sizes
    if (g > 0) {
        Var prevGroupSize = csp.getVar("Size"+(g-1));
        csp.post(prevGroupSize,"<=",groupSizeVar);
    }
}
```

Unfortunately, it did not help much. Besides, the problems of much larger sizes remain unsolvable for this implementation.

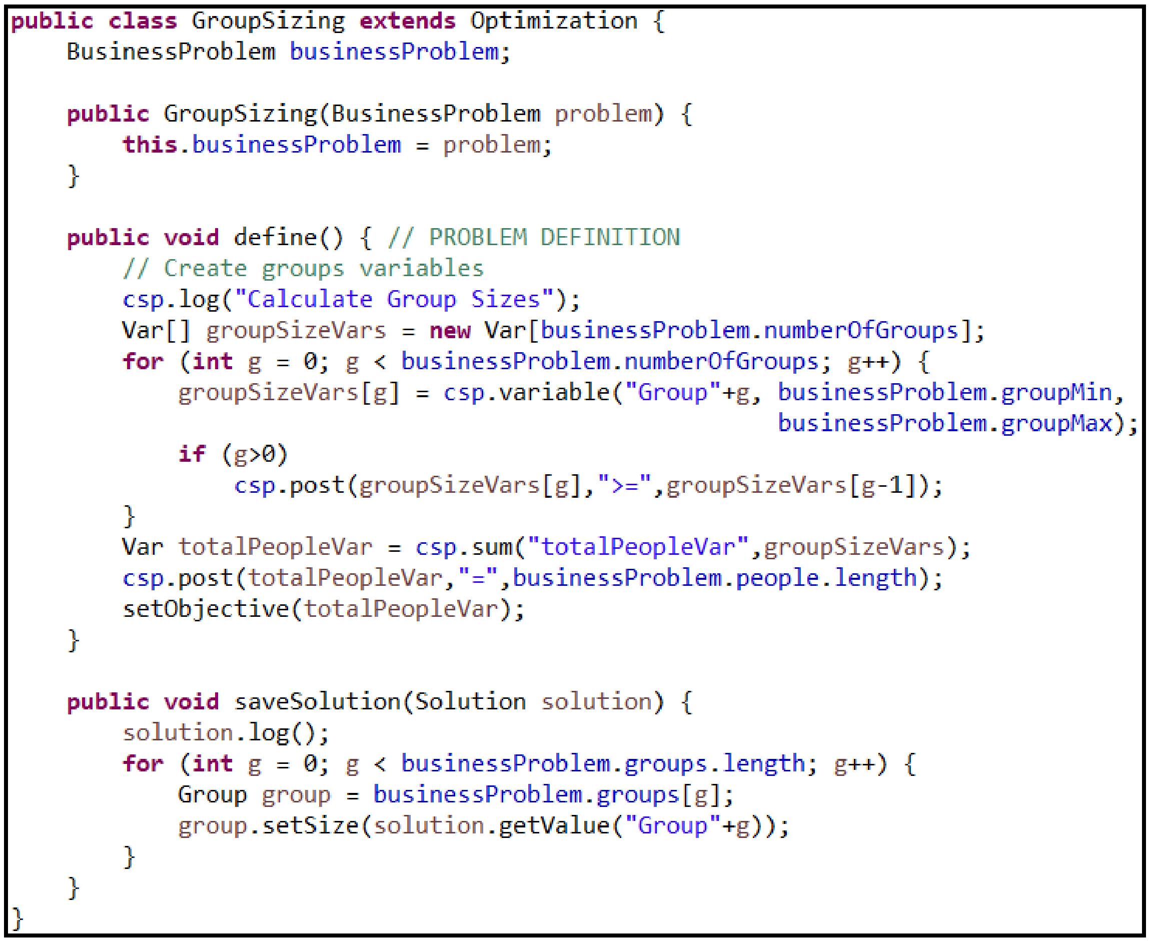
Second Implementation of the Optimization Problem with a Constraint Solver

Looking for a more efficient implementation, I thought: What if I pre-calculate group sizes ahead of time and then assign people to groups with already known sizes? It would probably sacrifice with optimality but dealing with one group in time might essentially limit the total number of all possible combinations. So, I decided to split the problem into two optimization sub-problems:

1) Define Group Sizes
2) Assign People to Groups while minimizing the total penalty inside each group.

Determining Group Sizes

This is a very simple optimization problem: given the total number of people, the number of groups, and their min/max limits find the group sizes. So, I quickly created the Java class GroupSizing inherited from the standard class Optimization with only two methods "define" and "saveSolution". Here it is:



```java
public class GroupSizing extends Optimization {
    BusinessProblem businessProblem;

    public GroupSizing(BusinessProblem problem) {
        this.businessProblem = problem;
    }

    public void define() { // PROBLEM DEFINITION
        // Create groups variables
        csp.log("Calculate Group Sizes");
        Var[] groupSizeVars = new Var[businessProblem.numberOfGroups];
        for (int g = 0; g < businessProblem.numberOfGroups; g++) {
            groupSizeVars[g] = csp.variable("Group"+g, businessProblem.groupMin,
                                                         businessProblem.groupMax);
            if (g>0)
                csp.post(groupSizeVars[g],">=",groupSizeVars[g-1]);
        }
        Var totalPeopleVar = csp.sum("totalPeopleVar",groupSizeVars);
        csp.post(totalPeopleVar,"=",businessProblem.people.length);
        setObjective(totalPeopleVar);
    }

    public void saveSolution(Solution solution) {
        solution.log();
        for (int g = 0; g < businessProblem.groups.length; g++) {
            Group group = businessProblem.groups[g];
            group.setSize(solution.getValue("Group"+g));
        }
    }
}
```

Hope by now this code is self-explanatory. The unknown variables here are sizes of groups with domain [groupmind, groupMax] – one per group. The only constraint is the sum of these variables is equal to the total number of people. This is a really very simple problem to solve.

Assigning People to One Group with Known Size

Now we may assume that the group size is known and our new optimization problem (called OptimizationProblemGroup inherited from Optimization) should assign people to one group only. Here is its implementation:

```java
public class OptimizationProblemGroup extends Optimization {

    BusinessProblem businessProblem;
    Group group;
    ArrayList<Person> remainingPeople;

    public OptimizationProblemGroup(BusinessProblem businessProblem, Group group) {
        this.businessProblem = businessProblem;
        remainingPeople = new ArrayList<Person>();
        for(Person person : businessProblem.getPeople()) {
            if (person.getGroup() < 0) //no assigned yet
                remainingPeople.add(person);
        }
        this.group = group;
    }

    public void define() { // PROBLEM DEFINITION
        // Define assignment variables
        int g = group.id;
        Var[] peopleGroupVars = new Var[remainingPeople.size()];
        for (int p = 0; p < remainingPeople.size(); p++) {
            peopleGroupVars[p] = csp.variable("G"+g+"P"+p, 0, 1);
        }
        // Post the fixed group size constraint
        Var groupSizeVar = csp.sum("Size"+g,peopleGroupVars);
        csp.post(groupSizeVar,"=",group.getSize());

        // Define penalty variables
        ArrayList<Var> penaltyVars = new ArrayList<Var>();
        int n = 0;
        for (int p1 = 0; p1 < remainingPeople.size()-1; p1++) {
            for (int p2 = p1+1; p2 < remainingPeople.size(); p2++) {
                int sameness = businessProblem.getSameness(p1,p2);
                if (sameness > 0) {
                        Var var1 = csp.getVar("G"+g+"P"+p1);
                        Var var2 = csp.getVar("G"+g+"P"+p2);
                        Var penaltyVar = var1.multiply(var2).multiply(sameness);
                        csp.add("Penalty"+n,penaltyVar);
                        penaltyVars.add(penaltyVar);
                        n++;
                }
            }
        }
        // Define optimization objective
        Var totalPenalty = csp.sum(penaltyVars);
        csp.add("TotalPenalty",totalPenalty);
        setObjective(totalPenalty);
    }
    public void saveSolution(Solution solution) {
        int g = group.id;
        for (int p = 0; p < remainingPeople.size(); p++) {
            Person person = remainingPeople.get(p);
            int value = solution.getValue("G"+g+"P"+p);
            if (value == 1) {
                group.add(person);
                person.setGroup(g);
            }
        }
        int penalty = solution.getValue("TotalPenalty");
        group.setPenalty(penalty);
        csp.log("TotalPenalty for group " + g + ": " + solution.getValue("TotalPenalty"));
    }
}
```

As you can see, initially in the constructor of OptimizationProblemGroup I initialized an array *remainingPeople* using only those people who haven't been assigned yet to any group. Then I defined assignment and penalty variables similarly to the previous implementation.

The main program that runs these two optimization problems GroupSizing and OptimizationProblemGroup looks as follows:

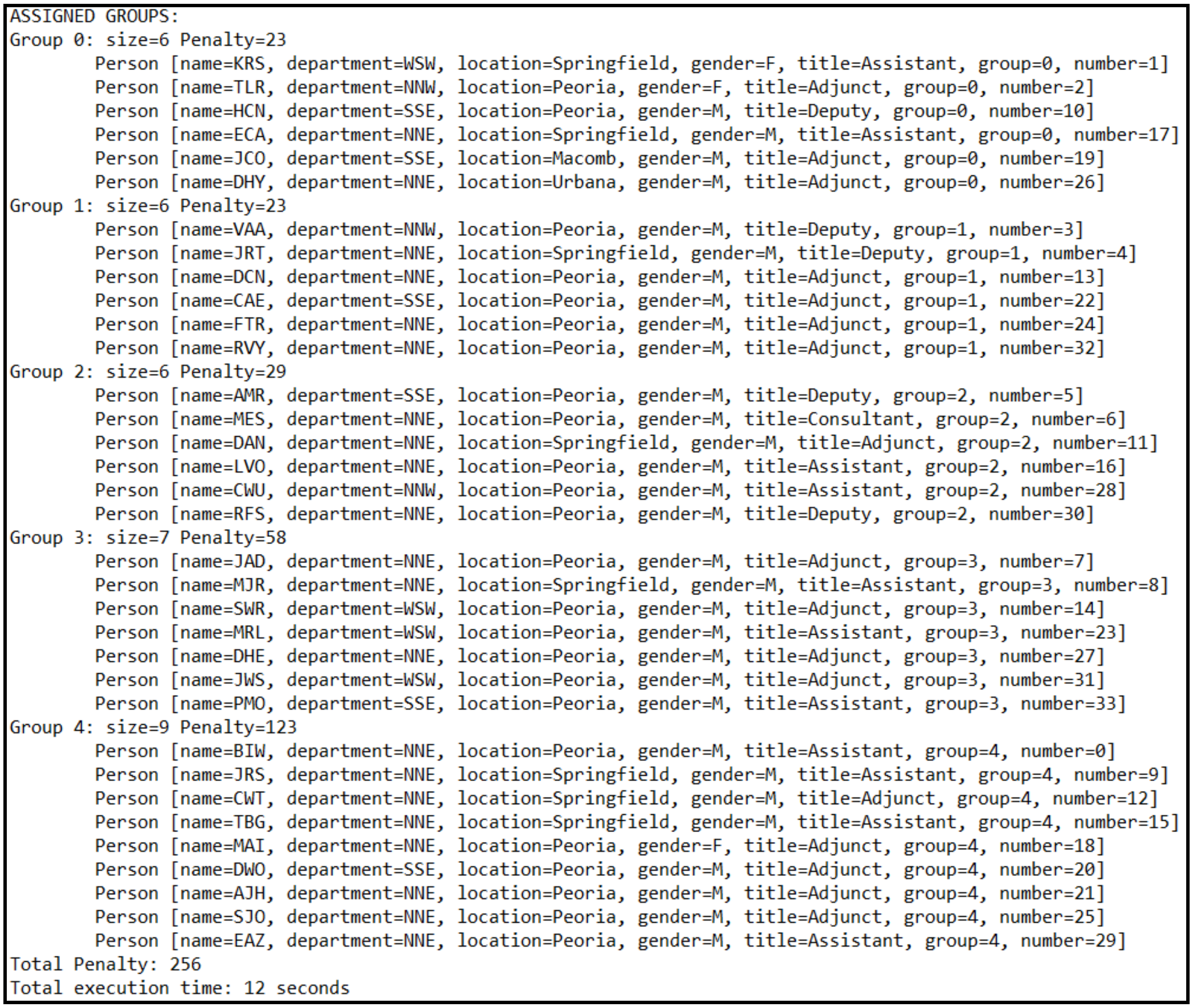
```
public class MainGroups {

    public static void main(String[] args) {
        long startTime = System.currentTimeMillis();
        // Create Business Problem
        DecisionData data = new DecisionData("file:rules/BalancedAssignmentDataSmall.xls");
        //DecisionData data = new DecisionData("file:rules/BalancedAssignmentData.xls");
        BusinessProblem problem = (BusinessProblem) data.get("getProblem");
        Person[] people = (Person[]) data.get("getPeople");
        problem.initialize(people);
        problem.defineSimilarities("file:rules/BalancedAssignment.xls", "DefineSameness");
        // Define Group Sizes
        GroupSizing groupSizing = new GroupSizing(problem);
        groupSizing.define();
        groupSizing.minimize();
        // Iterate through groups assigning remaining people to each of them
        Group[] groups = problem.getGroups();
        int penalty = 0;
        for(Group group : groups) {
            System.out.println("\nAssign people to Group "+group.getId());
            OptimizationProblemGroup optimization = new OptimizationProblemGroup(problem, group);
            optimization.define();
            optimization.minimize();
            penalty += group.getPenalty();
        }
        problem.showSolution();
        System.out.println("Total Penalty: " + penalty);
        long endTime = System.currentTimeMillis();
        System.out.println("Total execution time: " + (endTime - startTime)/1000 + " seconds");
    }
}
```

When I ran this main program, the real optimal solutions for each group were found without exceeding any time limit within the following times:

- Group 0: 8,1 seconds
- Group 1: 2,7 seconds
- Group 2: 0,7 seconds
- Group 3: 0,2 seconds
- Group 4: 0,1 seconds

Here are the results for all groups:



```
ASSIGNED GROUPS:
Group 0: size=6 Penalty=23
        Person [name=KRS, department=WSW, location=Springfield, gender=F, title=Assistant, group=0, number=1]
        Person [name=TLR, department=NNW, location=Peoria, gender=F, title=Adjunct, group=0, number=2]
        Person [name=HCN, department=SSE, location=Peoria, gender=M, title=Deputy, group=0, number=10]
        Person [name=ECA, department=NNE, location=Springfield, gender=M, title=Assistant, group=0, number=17]
        Person [name=JCO, department=SSE, location=Macomb, gender=M, title=Adjunct, group=0, number=19]
        Person [name=DHY, department=NNE, location=Urbana, gender=M, title=Adjunct, group=0, number=26]
Group 1: size=6 Penalty=23
        Person [name=VAA, department=NNW, location=Peoria, gender=M, title=Deputy, group=1, number=3]
        Person [name=JRT, department=NNE, location=Springfield, gender=M, title=Deputy, group=1, number=4]
        Person [name=DCN, department=NNE, location=Peoria, gender=M, title=Adjunct, group=1, number=13]
        Person [name=CAE, department=SSE, location=Peoria, gender=M, title=Adjunct, group=1, number=22]
        Person [name=FTR, department=NNE, location=Peoria, gender=M, title=Adjunct, group=1, number=24]
        Person [name=RVY, department=NNE, location=Peoria, gender=M, title=Adjunct, group=1, number=32]
Group 2: size=6 Penalty=29
        Person [name=AMR, department=SSE, location=Peoria, gender=M, title=Deputy, group=2, number=5]
        Person [name=MES, department=NNE, location=Peoria, gender=M, title=Consultant, group=2, number=6]
        Person [name=DAN, department=NNE, location=Springfield, gender=M, title=Adjunct, group=2, number=11]
        Person [name=LVO, department=NNE, location=Peoria, gender=M, title=Assistant, group=2, number=16]
        Person [name=CWU, department=NNW, location=Peoria, gender=M, title=Assistant, group=2, number=28]
        Person [name=RFS, department=NNE, location=Peoria, gender=M, title=Deputy, group=2, number=30]
Group 3: size=7 Penalty=58
        Person [name=JAD, department=NNE, location=Peoria, gender=M, title=Adjunct, group=3, number=7]
        Person [name=MJR, department=NNE, location=Springfield, gender=M, title=Assistant, group=3, number=8]
        Person [name=SWR, department=WSW, location=Peoria, gender=M, title=Adjunct, group=3, number=14]
        Person [name=MRL, department=WSW, location=Peoria, gender=M, title=Assistant, group=3, number=23]
        Person [name=DHE, department=NNE, location=Peoria, gender=M, title=Adjunct, group=3, number=27]
        Person [name=JWS, department=WSW, location=Peoria, gender=M, title=Adjunct, group=3, number=31]
        Person [name=PMO, department=SSE, location=Peoria, gender=M, title=Assistant, group=3, number=33]
Group 4: size=9 Penalty=123
        Person [name=BIW, department=NNE, location=Peoria, gender=M, title=Assistant, group=4, number=0]
        Person [name=JRS, department=NNE, location=Springfield, gender=M, title=Assistant, group=4, number=9]
        Person [name=CWT, department=NNE, location=Springfield, gender=M, title=Adjunct, group=4, number=12]
        Person [name=TBG, department=NNE, location=Springfield, gender=M, title=Assistant, group=4, number=15]
        Person [name=MAI, department=NNE, location=Peoria, gender=F, title=Adjunct, group=4, number=18]
        Person [name=DWO, department=SSE, location=Peoria, gender=M, title=Adjunct, group=4, number=20]
        Person [name=AJH, department=NNE, location=Peoria, gender=M, title=Adjunct, group=4, number=21]
        Person [name=SJO, department=NNE, location=Peoria, gender=M, title=Adjunct, group=4, number=25]
        Person [name=EAZ, department=NNE, location=Peoria, gender=M, title=Assistant, group=4, number=29]
Total Penalty: 256
Total execution time: 12 seconds
```

The total penalty for all groups is 256 that is much better than the previously found solution with the total penalty 402. This solution is probably very close to the optimal one, but there is still a possibility that we missed the optimal solution for other combinations of group sizes.

Then I tried to apply the same model to a larger problem with 101 people, 8 groups with possible sizes between 12 and 15. Initially, I limited execution time for 1 group to 60 seconds. For all groups, the optimization process found solutions but was interrupted by the time limit. So, within 424 seconds the same model assigned all 101 people to 8 groups with the overall penalty 1767.

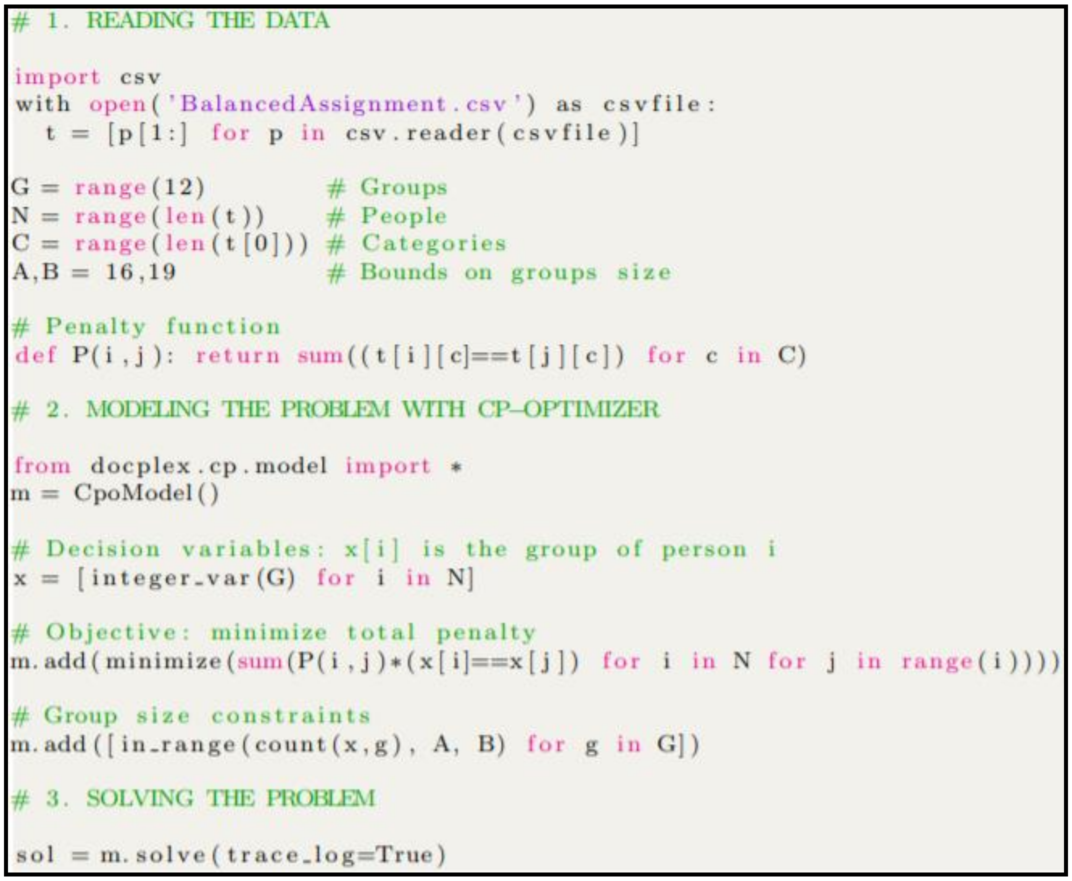
Then I increased the time limit to 120 seconds per group. This time it naturally worked twice longer (844 seconds) but managed to produce a slightly better solution with the overall penalty 1740. **We may conclude that the more time we will give to our model the better solutions it will be able to produce!**

Unfortunately, when I applied the same model to the "big" problem with 210 people and 12 groups, this model again ran out of memory.

Using a Commercial Constraint Solver

JSR331 allows me to switch between different underlying solvers without changing a character in my model. So, to try other available solvers, I simply changed the used solver's libraries (jar-files) and tried to use open source solvers with the "big" problem. Unfortunately, like Constrainer, these solvers also were not able to find a solution. So, I wanted to try IBM CP Optimizer, one of the most powerful commercial constraint solvers. Unfortunately, it is not (yet!) in the list of JSR331 supported solvers. So, I contacted Philippe Laborie, the Principal Scientist at IBM and a constraint programming guru, and asked him to create and solve a similar model using CP Optimizer.

Philippe kindly provided his own implementation and submitted it as a solution for the DMCommunity.org Challenge. He defined the penalty for having two people in the same group as the number of all characteristics they have in common. It is a special case if we make all increments in the decision table "DefineSameness" equal to 1. Philippe produces a very elegant and compact model using Python with CP Optimizer:



```
# 1. READING THE DATA

import csv
with open('BalancedAssignment.csv') as csvfile:
  t = [p[1:] for p in csv.reader(csvfile)]

G = range(12)          # Groups
N = range(len(t))      # People
C = range(len(t[0]))   # Categories
A,B = 16,19            # Bounds on groups size

# Penalty function
def P(i,j): return sum((t[i][c]==t[j][c]) for c in C)

# 2. MODELING THE PROBLEM WITH CP-OPTIMIZER

from docplex.cp.model import *
m = CpoModel()

# Decision variables: x[i] is the group of person i
x = [integer_var(G) for i in N]

# Objective: minimize total penalty
m.add(minimize(sum(P(i,j)*(x[i]==x[j]) for i in N for j in range(i))))

# Group size constraints
m.add([in_range(count(x,g), A, B) for g in G])

# 3. SOLVING THE PROBLEM

sol = m.solve(trace_log=True)
```

If you are a Java developer and have difficulties to understand this Python's "code", the good news is that the same CP Optimizer interface can be used with Java, C++, C#, or Scala. The important fact is that using this model Philippe managed to solve the "big" problem with 210 people and 12 groups and found the optimal solution with the overall penalty 3370! It's interesting to read Philippe's comments:

"Actually, it took some time. CP Optimizer very quickly finds a first solution (something like 0.12s on my laptop) with total penalty 3580 and then quickly improves it to a solution with total penalty around 3430-3420 within 1mn. Then the improvements continue but they are slow. I managed to speed it up by regularly stopping the search and restarting it providing the current best solution as a starting point (we have a notion of starting point solution in CP Optimizer). We need to investigate why this speeds up the search. After a few iterations and a few hours of computation, it finds the optimal solution.

To be honest, on this particular instance, as the value of the optimal solution is equal to the lower bound computed by separating the characteristics, I think that any local search operating on a partition of the persons in the groups would work very well and find this solution; probably faster than CP Optimizer. There are almost no constraints in this version of the problem. Of course, when the problem becomes more realist and you have some constraints as some you mentioned (no woman alone in a group, constraints on the locations inside a group, etc.), then a CP model makes more sense."

Note. I plan to add a JSR331 implementation based on CP Optimizer, so our users will have more modeling and execution choices. It's also not difficult at all to combine my business problem with Philippe's optimization problem.

Philippe made another important note about the use of constraint solvers such as CP Optimizer:

"About the long time it took for CP Optimizer to find the optimal solution. It is only because I was curious to see if I could find and prove an optimal solution that I let the solver spend so much time on it. In general, and I see it every day with our customers, the users solve optimization problems that are way more complex than this balanced assignment problem and they are not focused on "optimal" solutions (in practice, for real-world problems optimality proofs are often too hard to reach, or it is at the price of over-simplifying the problem, meaning that the proof - if any - has no real value) but users rather aim at finding "very good quality" solutions. And this is also, I think, an advantage of tools like CP Optimizer: even if the search algorithm is complete and guaranteed to find optimal solutions if given enough time, usually, good solutions are produced quite fast because it uses an hybrid of complete search space exploration and meta-heuristics (the search approach in CP Optimizer is not a secret by the way, it has been described in papers and publicized, like here: http://ibm.biz/Constraints2018). Here, I'm pretty sure that a solution of penalty around 3420-3430 obtained by the simple model I gave after a couple of minutes would be "good enough" for a user."

I concur! Solving real-world complex decisioning problems for years, I know that business users usually care more about getting a "good enough" solution as quickly as possible instead of looking for an optimal solution. At the same they want to have more control over the problem formulation with an ability to easily modify the model behavior. I believe a good combination or business rule engines and optimization tools provides users with such power.

Using Linear Solvers

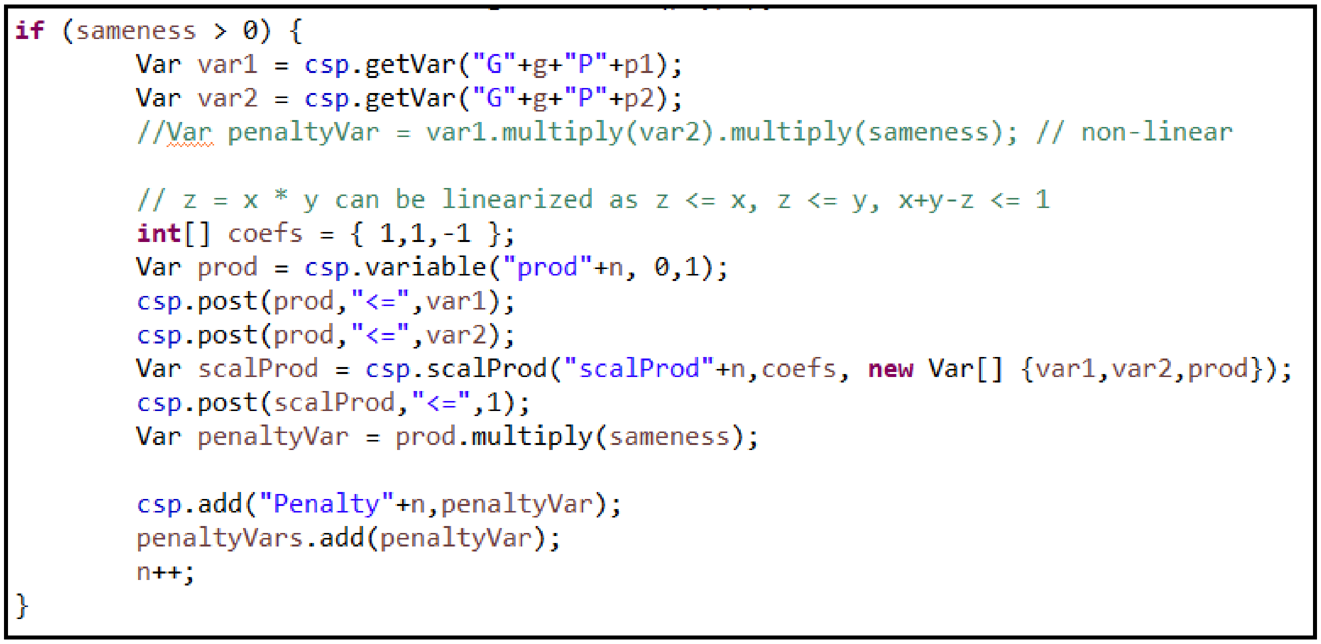
When a problem has only linear constraints, usually, linear solvers are much more efficient to compare with constraint solvers. So, I decided to linearize my model and try it with linear solvers. It's good that JSR331 allows me to easily witch to linear solvers. To do this I needed to replace only two constraint jar-files with two linear jar-files. JSR331 supports 6 different linear solvers. I decided first to try open source linear solver such as SCIP.

To linearize the above optimization problem, I only need to replace this non-linear constraint:

```
Var penaltyVar = var1.multiply(var2).multiply(sameness);
```

It's not so difficult to understand that the non-linear constraint $z = x * y$ is equivalent to 3 linear constraints $z <= x$, $z <= y$, $x+y-z <= 1$ when x, y, and z are Boolean.

So, here is a snippet of a Java code that does exactly such a replacement:



```
if (sameness > 0) {
        Var var1 = csp.getVar("G"+g+"P"+p1);
        Var var2 = csp.getVar("G"+g+"P"+p2);
        //Var penaltyVar = var1.multiply(var2).multiply(sameness); // non-linear

        // z = x * y can be linearized as z <= x, z <= y, x+y-z <= 1
        int[] coefs = { 1,1,-1 };
        Var prod = csp.variable("prod"+n, 0,1);
        csp.post(prod,"<=",var1);
        csp.post(prod,"<=",var2);
        Var scalProd = csp.scalProd("scalProd"+n,coefs, new Var[] {var1,var2,prod});
        csp.post(scalProd,"<=",1);
        Var penaltyVar = prod.multiply(sameness);

        csp.add("Penalty"+n,penaltyVar);
        penaltyVars.add(penaltyVar);
        n++;
}
```

When I tried to execute the linearized model with the SCIP linear solver against our "big" problem. Unfortunately, after running it for a few hours, I could not get a solution for the very first group. Then I tried GLPK and COIN linear solvers but they also failed to find a solution. I didn't have licenses for the best commercial linear solvers CPLEX and GUROBI. But the good news was that JSR331 actually generates a linear problem in the MPS format and such a format is supported by any linear solver. So, I decided to send the generated MPS file to my colleagues in IBM and in Gurobi. The generated file is quite huge: ~15Mb and please remember that this file covers only the very first group.

CPLEX. Philippe Laborie from IBM wrote: "*I run your MIP file with CPLEX. Using default parameter values, on my laptop it finds a solution at cost 214 in a bit less than 30 minutes, with a lower-bound around 65, so still a large gap. I don't know what is the optimal value here.*" Philippe also let CPLEX automatically linearize a variant of his own model for the "big" problem: within 15 minutes, CPLEX found a solution with the overall penalty 3750 while CP Optimizer's optimal solution was 3370. Philippe wrote: "*One can probably do some tricks in the model to improve the results*".

GUROBI. My colleague from Gurobi Greg Glockner kindly responded to my request to run this file using Gurobi solver. Here is his answer: "*We have finished testing your model. For this model, we recommend the following parameters: Presolve 2 and MIPFocus 3. It's easy to get a feasible solution with objective 214. The proof of optimality is challenging. We tested your model using distributed optimization on multiple computers, each with an Intel Xeon E3-1240 processor (3.40 GHz). With 8 computers, the MIP gap reaches 41.6% in about 1 day. A log file is attached. Looking at the zero-half cuts in the logs, I suspect this model contains many disjunctions. It might solve better with another formulation.*"

Note that both CPLEX and Gurobi quickly find a good feasible solution for our "big" problem that in real-world situations could be more than enough.

Applying Expert Knowledge to Redefine the Problem

Prof. Fourer in his introduction of this problem stated that an attempt "to minimize the Total Sameness is problematic" (see slide 16). Instead, he recommended to "minimize the Total Variation that is a sum over all types: most minus least assigned to any group". This modeling approach dramatically minimizes the size of the problem allowing off-the-shelf solvers to quickly solve even the "big" balanced assignment problem. Two solutions (AMPL and OPL) that are based on the concept of the "Total Variation" have been already submitted and can be found here. However, this approach requires expert knowledge and not easy to follow. Even an experienced software developer who used to apply mainly rules-based tools may have difficulties to understand the provided optimization-based solutions. In real-world cases, everything depends on the complexity of the optimization problem and sometimes a similar use of expert knowledge is unavoidable.

Conclusion

I wanted to demonstrate how to use the integrated Business Rules (BR) and Optimization approach for the creation of user-friendly decision models with strong optimization components. So, I've tried to apply this approach to the DMCommunity Challenge "Balanced Assignment". I've split the problem into two parts: Business Problem (created and maintained by a business person) and Optimization Problem (created and maintained by a developer). In real-world situations, it's preferable to move as much knowledge as possible to the Business Problem, but for this decision model, I concentrated on the optimization part. I've presented different modeling techniques using constraint and linear solvers and considering my own and other people models. I can make several important conclusions:

- There are many off-the-shelf BR and Optimization tools that can be used together to create practical business decision models
- The more time we will give to a decision model the better solutions it may produce
- It's usually preferable and possible to use simple and intuitive models whose behavior is controlled by business users. However, with expert knowledge, it is possible to find practical solutions for business problems of almost any complexity.