Assignment 5(b): Integer Programming

Pietro Bonazzi

Economics and Computation, MSc Course, 1st Sem., Fall 2020
University of Zürich
pietro.bonazzi@uzh.ch

Contents

Chapter 1

Going on Vacation

Alice and Bob are going on vacation and they have n different items that they could potentially take with them. Item i weighs wi kilograms and the two would have value vi for taking item i with them. In order to use item i on vacation, it has to be placed in one of two suitcases. Alice has a suitcase which can carry bA kilograms, and Bob has a suitcase which can carry bB kilograms. Alice and Bob would like to choose items for their trip in a way, such that their total value is maximized. We assume that the total weight of items in a suitcase is the only capacity constraint that we have to worry about, i.e., the size or form of items does not matter. Also, we assume that wi bA and wi bB for all i, i.e., each item fits in each of the two suitcases.

1.1 Formulate and implement the problem

The formulation of the problem is illustrated in Figure 1.1.1. The objective function aims at maximizing the total value of items collected in the two suitcases. The first two constraints define the capacity constraints of the two suitcases. The last inequality specifies that one item cannot be inserted in both suitcases at the same time. After Figure 1.1.1, the implementation of the problem in Java/JOpt is shown.

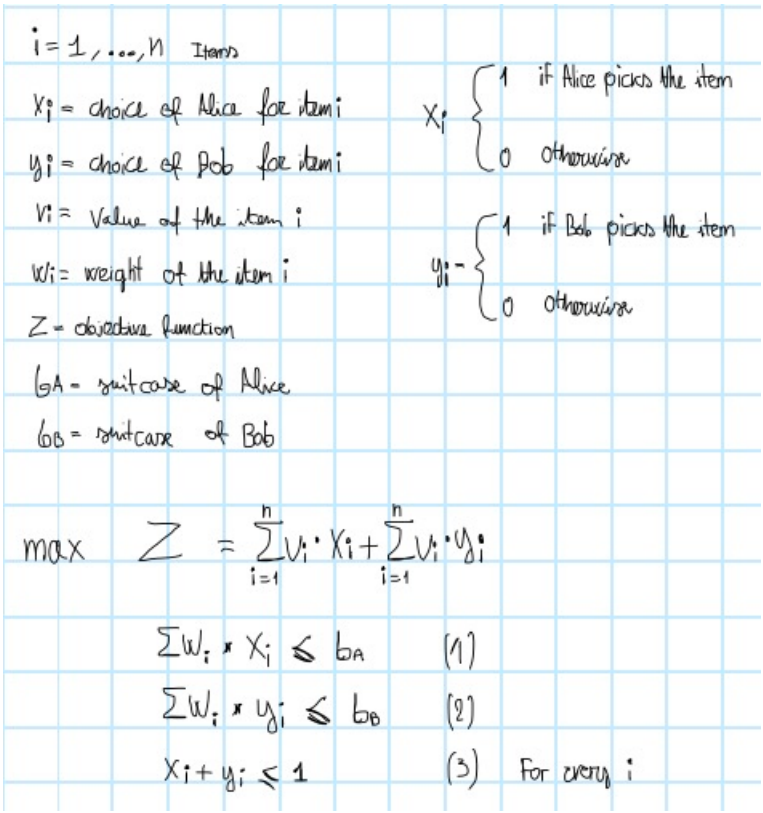

Figure 1.1.1: Formulation of the problem.

```
        ArrayList<Variable> ITEM_ALICE_CHOICE = new ArrayList<>();
        ArrayList<Variable> ITEM_BOB_CHOICE = new ArrayList<>();

        for (int i=0; i<ITEM_VALUES.length; i++) {
            ITEM_ALICE_CHOICE.add(new Variable("x"+i, VarType.INT, 0, 1));
            ITEM_BOB_CHOICE.add(new Variable("y"+i, VarType.INT, 0, 1));
             mip.add(ITEM_ALICE_CHOICE.get(i));
             mip.add(ITEM_BOB_CHOICE.get(i));
        }

        mip.setObjectiveMax(true);
            for (int i=0; i<ITEM_VALUES.length; i++) {
                mip.addObjectiveTerm(ITEM_VALUES[i], ITEM_ALICE_CHOICE.get(i));
                mip.addObjectiveTerm(ITEM_VALUES[i], ITEM_BOB_CHOICE.get(i));
            }

        Constraint c1 = new Constraint(CompareType.LEQ, CAPACITY_A);
        for (int i=0; i<ITEM_VALUES.length; i++) {
            c1.addTerm(ITEM_WEIGHTS[i], ITEM_ALICE_CHOICE.get(i));
        }
        mip.add(c1);

        Constraint c2 = new Constraint(CompareType.LEQ, CAPACITY_B);
        for (int i=0; i<ITEM_VALUES.length; i++) {
            c2.addTerm(ITEM_WEIGHTS[i], ITEM_BOB_CHOICE.get(i));
        }
        mip.add(c2);

        for (int i=0; i<ITEM_VALUES.length; i++) {
            Constraint c3 = new Constraint(CompareType.LEQ, 1);
            c3.addTerm(1, ITEM_ALICE_CHOICE.get(i));
            c3.addTerm(1, ITEM_BOB_CHOICE.get(i));
            mip.add(c3);
        }
```

1.2 Arbitrary number of suitcases

Now, assume that Alice and Bob have an arbitrary number of suitcases at home instead of the two suitcases with capacities bA and bB. Each available suitcase has capacity b. Alice and Bob have now decided to take all n items with them, but they would like to use as few suitcases as possible. Assume that wi b for all i. Note that they will need no more than n suitcases. Write down the corresponding mixed-integer program and implement it in Java/JOpt.

The formulation of this problem employed variables, which were already defined in the previous section. Fundamentally, this problem did not need an optimization set up with constraints. Nonetheless, it has been declared that the objective function aims at minimizing the number of suitcases. Subsequently, the minimum number of suitcase is no more than the solution rounded to the greatest integer of the sum of the weights of all items divided by the capacity of an arbitrary suitcase. The formulation of the problem is illustrated in the Figure below 1.2.1.

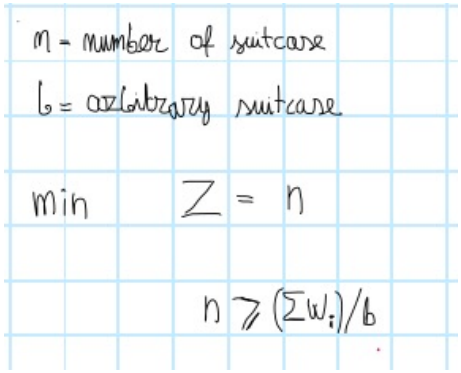

Figure 1.2.1: Formulation of the problem.

```
        Variable numSuitcase = new Variable("x", VarType.INT, -MIP.MAX_VALUE, MIP.MAX_VALUE);
        mip.add(numSuitcase);

        mip.setObjectiveMax(false);
        mip.addObjectiveTerm(1, numSuitcase);

        int coeff = (Arrays.stream(ITEM_WEIGHTS).sum()+CAPACITY-1)/CAPACITY;
        Constraint c1 = new Constraint(CompareType.GEQ, coeff);
        c1.addTerm(1, numSuitcase);
        mip.add(c1);
```

Chapter 2

Planning a Roadtrip

You are planning to visit 17 different cities C = 1, ..., 17 by car during the summer holiday. For each pair of cities (i, j) you are given the distance between i and j. Starting from city 1, your goal is to visit each city exactly once and then to return to city 1 again, while minimizing the total travelling distance.

2.1 Traveling salesman problem (TSP), formulation

The above road trip problem can be interpreted as an instance of the traveling salesman problem (TSP). The TSP can be formalized as an integer program. You can find the IP formulation on the web (e.g., on Wikipedia).

Two formulation of this problem can be found in this Wikipedia's page.

One solution has been formulated by Miller–Tucker–Zemlin and another one by Dantzig–Fulkerson–Johnson. The former was chosen over the latter to exclude *a priori* the risk of having an exponential number of constraints.

In Figure 2.1.1, five constraints can be identified. First, every city must be reached only from one other city. Second, from every city just one other city can be reached. Third, no more than one tour that cover all cities is accepted.

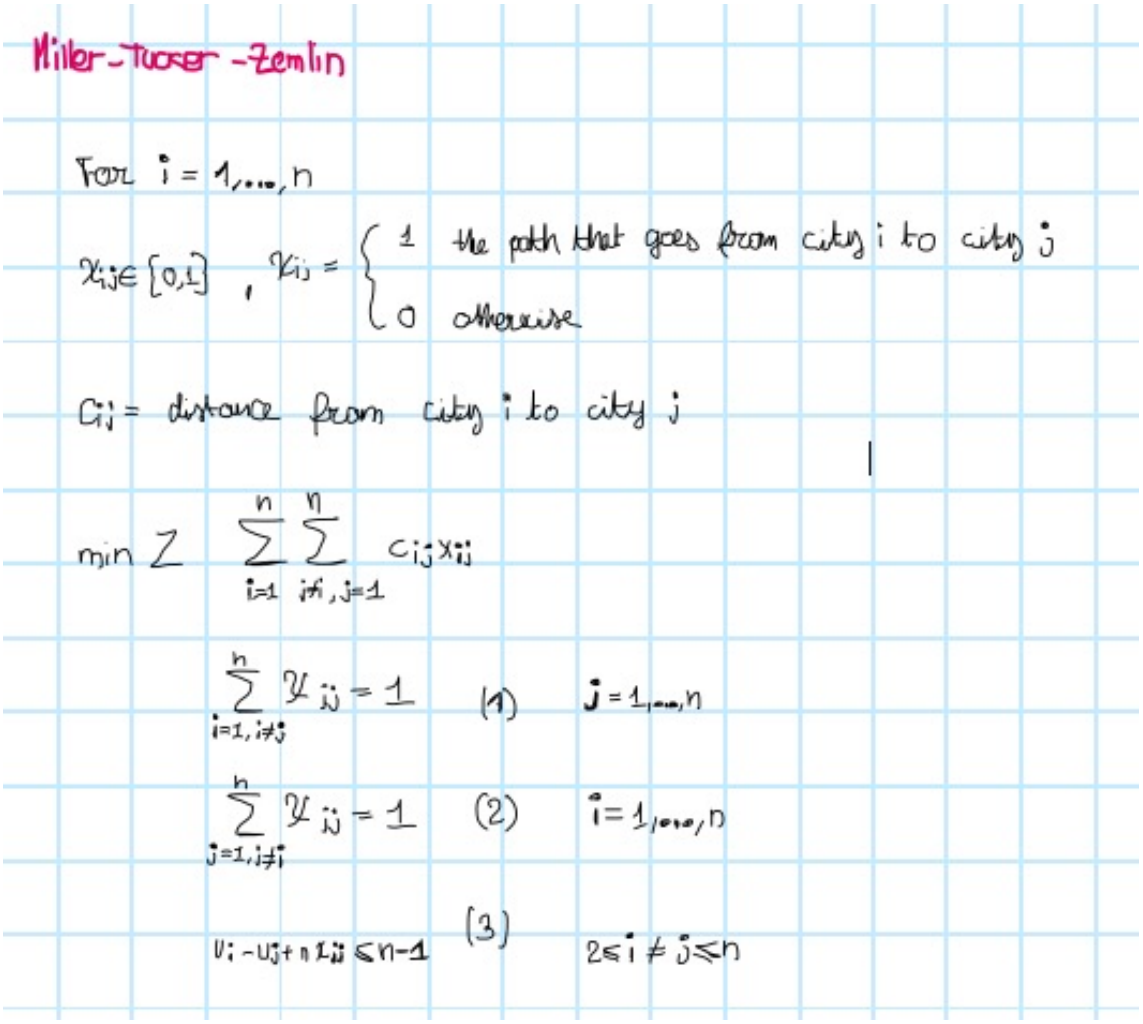

Figure 2.1.1: Miller–Tucker–Zemlin formulation.

2.2 Implement a TSP solver in Java/JOpt

The same Java/Jopt formulation method that worked in the previous exercise failed here. When compiling the code the console raised an exception when it was instructed to add the variables of the multidimensional array PATH. In particular, the PATH.get(i).get(j) called an object that apparently did not exist even if it was previously created. Reference to the code can be found in the box below

```
        this.numberOfCities = cities.length;
        ArrayList<ArrayList<Variable> > PATH = new ArrayList<>();

        for (int i=0; i<numberOfCities; i++) {
            PATH.add(new ArrayList<>());
            for (int j=0; j<numberOfCities; j++) {
                if (i!=j) {PATH.get(i).add(new Variable("x"+i+"_"+j, VarType.INT, 0, 1));}
                else {PATH.get(i).add(new Variable("0", VarType.INT, 0, 0));}
                mip.add(PATH.get(i).get(j));
            }
        }
```

Finally, the objective function and the constraints were formulated as follows. The dummy

variables ui and uj, presented in the formulation, are not defined in the implementation of the third constraint.

```
        //objective function
        mip.setObjectiveMax(false);
        for (int i=0; i<numberOfCities; i++) {
            for (int j=0; j<numberOfCities; j++) {mip.addObjectiveTerm(distances[i][j], PATH.get(i).get(j));}}

        //first constraint
        for (int i=0; i<numberOfCities; i++) {
            Constraint c1 = new Constraint(CompareType.EQ, 1);
            for (int j=0; j<numberOfCities; j++) {c1.addTerm(1, PATH.get(i).get(j));}
            mip.add(c1);}

        //second constraint
        for (int i=0; i<numberOfCities; i++) {
            Constraint c2 = new Constraint(CompareType.EQ, 1);
            for (int j=0; j<numberOfCities; j++) {c2.addTerm(1, PATH.get(j).get(i));}
            mip.add(c2);}

        //third constraint
        for (int i=0; i<numberOfCities; i++) {
            for (int j=0; j<numberOfCities; j++) {
                Constraint c3 = new Constraint(CompareType.LEQ, numberOfCities-1);
                c3.addTerm(numberOfCities, PATH.get(j).get(i));
                mip.add(c3); } }
```

To conclude, the print statement could not be properly tested due to the problem encountered early in the problem. The intuition was to create a for loop with the dummy variables ui and uj.

```
            for (int i = 0; i < numberOfCities; i++) {
                for (int j = 0; j < numberOfCities; j++) {
                    returnString += "From City "  + "to City \n";}
            }
```