Integer Linear Programming, Min-Max, Max-Min, Energy Barriers, and Enumeration of Solutions in a Chemical Reaction Network

Daniel Merkle, Christoph Flamm

July 2, 2024

Contents

1 Introduction

In this set of exercises, we will explore min-max optimization problems, which are crucial in various fields such as operations research, computer science, and of course when considering chemical reaction networks. Min-max problems involve finding the minimum of the maximum values or the maximum of the minimum values under certain constraints. These problems can often be challenging and, in some cases, are NP-complete (we will see that ont of the following problems will allow us to solve the subset sum problem).

We will start with relatively simple problems that could for sure be solved without integer linear programming (ILP) tools. As we progress, we will tackle more complex problems that inherently solve NP-complete problems as a side effect. In later exercises, we will apply the techniques learned here to real-world scenarios, such as optimizing chemical reaction networks.

To illustrate how to use Gurobi with Python for solving optimization problems, we will first solve a classical problem known as the Knapsack Problem.

All example code, code templates, and solutions (after the summer school) can be found here: https://tacsy-school-2024.algochem.techfak.de/

1.1 Knapsack Problem Example

Problem Statement: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

Example: Consider the following items with their weights and values:

- Item 1: weight = 2, value = 3
- Item 2: weight = 3, value = 6
- Item 3: weight = 4, value = 5
- Item 4: weight = 5, value = 8

The weight limit of the knapsack is 5.

Decision Variables

$$x_i \quad \text{for } i = 1, 2, \ldots, n \quad (x_i \in \{0, 1\})$$

Here, x_i is a binary variable indicating whether item i is included in the knapsack (1) or not (0).

Objective Function

$$\text{Maximize} \sum_{i=1}^{n} v_i \cdot x_i$$

Where v_i is the value of item i.

Constraints

$$\sum_{i=1}^{n} w_i \cdot x_i \leq W$$

Where w_i is the weight of item i, and W is the maximum weight capacity of the knapsack.

Solution to the Example

To solve this example, we use the following decision variables:

$$x_1 = 1, \quad x_2 = 1, \quad x_3 = 0, \quad x_4 = 0$$

This means we include item 1 and item 2 in the knapsack.

- Total weight $= 2 \cdot 1 + 3 \cdot 1 + 4 \cdot 0 + 5 \cdot 0 = 5$
- Total value $= 3 \cdot 1 + 6 \cdot 1 + 5 \cdot 0 + 8 \cdot 0 = 9$

Python Code using Gurobi:

```python
import gurobipy as gp
from gurobipy import GRB

def solve_knapsack(weights, values, capacity):
    # Number of items
    n = len(weights)

    # Create a new model
    model = gp.Model("knapsack")

    # Create variables
    x = model.addVars(n, vtype=GRB.BINARY, name="x")

    # Set objective: maximize total value
    model.setObjective(gp.quicksum(values[i] * x[i] for i in range(n)),
        GRB.MAXIMIZE)

    # Add constraint: total weight must be less than or equal to capacity
    model.addConstr(gp.quicksum(weights[i] * x[i] for i in range(n)) <=
        capacity, "capacity")

    # Optimize the model
    model.optimize()

    # Print the results
    if model.status == GRB.OPTIMAL:
        print("Optimal solution found")
        selected_items = [i for i in range(n) if x[i].x > 0.5]
        print(f"Selected items: {selected_items}")
        print(f"Total value: {model.objVal}")
    else:
        print("No optimal solution found")

if __name__ == "__main__":
    # Example data
    weights = [2, 3, 4, 5]
    values = [3, 6, 5, 8]
    capacity = 5

    solve_knapsack(weights, values, capacity)
```

2 Min-Max, Max-Min, Exercises

Each exercise in this document is divided into three parts: a) an example and solving it, b) the mathematical formulation, which includes defining decision variables, the objective function, and the constraints, and c) the implementation using Python and Gurobi.

2.1 Exercise 1

Problem Statement: Consider i variables x_j for $j = 0, 1, \ldots, i-1$ such that $m \leq x_j \leq M$. You need to select which variables x_j are chosen for the sum, such that the sum of the chosen variables equals a given target sum. Additionally, you want to minimize the maximum value among the chosen variables.

a) Example

For $i = 10$, $m = 0$, $M = 20$, and the target sum $= 100$, solve the problem with pen, paper, and brain.

b) Mathematical Formulation

Define the decision variables, the objective function, and the constraints.

c) Implementation

Implement a solution to the problem using Python and Gurobi.

2.2 Exercise 2

Problem Statement: Consider i variables x_j for $j = 0, 1, \ldots, i-1$ such that $m \leq x_j \leq M$. You need to select which variables x_j are chosen for the sum, such that the sum of the chosen variables equals a given target sum. Additionally, you want to maximize the minimum value among the chosen variables.

a) Example

For $i = 10$, $m = 0$, $M = 20$, and the target sum = 100, solve the problem with pen, paper, and brain.

b) Mathematical Formulation

Define the decision variables, the objective function, and the constraints.

c) Implementation

Implement a solution to the problem using Python and Gurobi.

2.3 Exercise 3

Problem Statement: Consider the numbers $5, 8, 15, 21, 22, 25, 26, 27, 36, 50$. For each x_j, let x_j be one of the given numbers. Select which variables x_j are chosen for the sum, such that the sum of the chosen variables equals 100. Additionally, you want to minimize the maximum value among the chosen variables.

a) Example

For the given numbers and the target sum = 100, solve the problem with pen, paper, and brain.

b) Mathematical Formulation

Define the decision variables, the objective function, and the constraints.

c) Implementation

Implement a solution to the problem using Python and Gurobi.

2.4 Exercise 4

Problem Statement: Consider the numbers $5, 8, 15, 21, 22, 25, 26, 27, 36, 50$. For each x_j, let x_j be one of the given numbers. Select which variables x_j are chosen for the sum, such that the sum of the chosen variables equals 100. Additionally, you want to maximize the minimum value among the chosen variables.

a) Example

For the given numbers and the target sum = 100, solve the problem with pen, paper, and brain.

b) Mathematical Formulation

Define the decision variables, the objective function, and the constraints.

c) Implementation

Implement a solution to the problem using Python and Gurobi.

3 Energy Barriers, Min-Max, and Enumeration of Solutions in CRNs

3.1 Enumeration - Introduction and Exercise

The following code provides all hyperedges of a chemical reaction network underlying the non-oxidative pentose phosphate pathway (PPP) and should be considered as an example only. Details can be found in the additional material. Below is the python/Gurobi code to find an optimal solution using the sum of flows as an objective. In addition, we enforce, among other flows, the inflow of Fructose-6-Phosphate and the outflow of Fructose-6-Phosphate. Here is the code to find an optimal solution, using balance constraints:

```
hyperedges = {
    3: (['Ribulose-5-Phosphate'], ['p_{0,0}']),
    6: (['Ribulose-5-Phosphate', 'p_{0,0}'], ['p_{0,1}', 'p_{0,2}']),
    9: (['p_{0,1}', 'p_{0,2}'], ['Fructose-6-Phosphate', 'p_{0,3}']),
    11: (['p_{0,0}', 'p_{0,1}'], ['p_{0,3}', 'p_{0,4}']),
    13: (['Ribulose-5-Phosphate', 'p_{0,2}'], ['Fructose-6-Phosphate',
        'p_{0,5}']),
    14: (['Fructose-6-Phosphate', 'p_{0,3}'], ['Fructose-6-Phosphate',
        'p_{0,3}']),
    16: (['Fructose-6-Phosphate', 'p_{0,5}'], ['p_{0,3}', 'p_{0,6}']),
    17: (['Fructose-6-Phosphate', 'p_{0,2}'], ['Ribulose-5-Phosphate',
        'p_{0,3}']),
    18: (['Fructose-6-Phosphate', 'p_{0,0}'], ['p_{0,1}', 'p_{0,3}']),
    20: (['p_{0,3}', 'p_{0,4}'], ['Fructose-6-Phosphate', 'p_{0,7}']),
    21: (['Ribulose-5-Phosphate', 'p_{0,3}'], ['Fructose-6-Phosphate',
        'p_{0,2}']),
    22: (['p_{0,1}', 'p_{0,3}'], ['Fructose-6-Phosphate', 'p_{0,0}']),
    23: (['p_{0,4}', 'p_{0,5}'], ['p_{0,6}', 'p_{0,7}']),
    24: (['p_{0,2}', 'p_{0,4}'], ['Ribulose-5-Phosphate', 'p_{0,7}']),
    25: (['p_{0,0}', 'p_{0,4}'], ['p_{0,1}', 'p_{0,7}']),
    26: (['Ribulose-5-Phosphate', 'p_{0,5}'], ['p_{0,2}', 'p_{0,6}']),
    27: (['p_{0,1}', 'p_{0,5}'], ['p_{0,0}', 'p_{0,6}']),
    29: (['p_{0,2}'], ['p_{0,8}']),
    31: (['p_{0,0}', 'p_{0,8}'], ['p_{0,9}']),
    33: (['p_{0,2}', 'p_{0,8}'], ['p_{0,10}']),
    36: (['H2O', 'p_{0,9}'], ['p_{0,11}', 'p_{0,12}']),
    37: (['H2O', 'p_{0,9}'], ['p_{0,4}', 'p_{0,11}']),
    39: (['H2O', 'p_{0,10}'], ['p_{0,11}', 'p_{0,13}']),
    40: (['H2O', 'p_{0,10}'], ['Fructose-6-Phosphate', 'p_{0,11}']),
    41: ([], ['H2O']),
    42: ([], ['Ribulose-5-Phosphate']),
    43: (['Fructose-6-Phosphate'], []),
    44: (['p_{0,11}'], [])
}

# Create the model
model = Model('HypergraphFlow')

# Create variables
x = {}
for e_id in hyperedges:
    x[e_id] = model.addVar(vtype=GRB.INTEGER, name=f'x_{e_id}')
```

```python

# Update model to integrate new variables
model.update()

# Flow conservation constraints
vertices = set(v for e in hyperedges.values() for v in e[0] + e[1])

for v in vertices:
    inflow = quicksum(x[e_id] for e_id, e in hyperedges.items() if v in e[1])
    outflow = quicksum(x[e_id] for e_id, e in hyperedges.items() if v in
        e[0])
    model.addConstr(inflow == outflow, name=f'flow_conservation_{v}')

# Specific inflow and outflow constraints
model.addConstr(x[41] == 1, name='inflow_H2O')
model.addConstr(x[42] == 6, name='inflow_Ribulose_5_Phosphate')
model.addConstr(x[43] == 5, name='outflow_Fructose_6_Phosphate')
model.addConstr(x[44] == 1, name='outflow_p_0_11')

# Objective function: Minimize the sum of flows
model.setObjective(quicksum(x[e_id] for e_id in hyperedges), GRB.MINIMIZE)

# Optimize the model
model.optimize()

# Get the optimal solution
optimal_solution = {e_id: x[e_id].X for e_id in x if x[e_id].X > 0}

# Output the results for the optimal solution
print("Optimal Solution:")
for e_id, flow in optimal_solution.items():
    tails, heads = hyperedges[e_id]
    print(f'Hyperedge {e_id}: Flow = {flow}, Heads = {heads}, Tails =
        {tails}')
```

Exercise 5

Your task is to find a (second best) solution, for which you shall guarantee that not the same hyperedges as in the best found solution are used. Please first formulate the additional decision variables and the additional constraints (keep the objective function unmodified).

3.2 Energy barriers - Introduction and Exercise

Imagine you would know a value for each hyperedge that indicates how likely or unlikely a reaction is to happen (think about the height of an energy barrier height). For this, we introduce a value $r_{e_{id}}$ per hyperedge, which should be considered as predefined constants per hyperedge for the following exercises.

Exercise 6

We want to find pathways which minimize the energy barrier height of all hyperedges used. Your goal is to find solutions which minimize the max of all $r_{e_{id}}$ for all hyperedges in a solution.

a)

As usual, introduce decision variables, constraints, and the objective function.

b)

Similar to exercises solved earlier, find a second solution, which uses a different set of hyperedges compared to the optimal solution. Provide additional decision variables and constraints.