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.

Solutions

Solution to Exercise 1

- **Decision Variables:**

$$x_j \quad \text{for } j = 0, 1, \ldots, i-1$$

$$b_j \quad \text{binary variables indicating whether } x_j \text{ is chosen}$$

$$T \quad \text{maximum value among the chosen variables } x_j$$

- **Objective Function:**

$$\text{Minimize } T$$

- **Constraints:**

$$\sum_{j=0}^{i-1} x_j = \text{target sum}$$

$$x_j \leq T \quad \forall j$$

$$x_j \leq M \cdot b_j \quad \forall j$$

$$x_j \geq m \cdot b_j \quad \forall j$$

```
python3.11 ./max-min-gen.py 10 0 20 100

Optimale Lösung gefunden:
Verwendete Zahlen: [10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0]
Maximaler Wert (T): 10.0
```

```python
import argparse
from gurobipy import Model, GRB, quicksum

def minimize_max_value(i, m, M, target_sum):
    model = Model("minimize_max_value")

    x = model.addVars(i, vtype=GRB.INTEGER, name="x")
    b = model.addVars(i, vtype=GRB.BINARY, name="b")
    T = model.addVar(vtype=GRB.INTEGER, name="T")

    model.setObjective(T, GRB.MINIMIZE)

    model.addConstr(quicksum(x[j] for j in range(i)) == target_sum,
        name="sum_constraint")
    for j in range(i):
        model.addConstr(x[j] <= T, name=f"max_constraint_{j}")
        model.addConstr(x[j] <= M * b[j], name=f"binary_max_constraint_{j}")
        model.addConstr(x[j] >= m * b[j], name=f"binary_min_constraint_{j}")

    model.optimize()

    if model.status == GRB.OPTIMAL:
```

```
        solution = [x[j].x for j in range(i) if b[j].x > 0.5]
        print("Optimal solution found:")
        print(f"Selected values: {solution}")
        print(f"Maximum value (T): {T.x}")
    else:
        print("No optimal solution found")

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Minimize the maximum value
         of a subset that sums to a target.')
    parser.add_argument('i', type=int, help='number of variables')
    parser.add_argument('m', type=int, help='minimum value for variables')
    parser.add_argument('M', type=int, help='maximum value for variables')
    parser.add_argument('target_sum', type=int, help='the target sum')

    args = parser.parse_args()
    minimize_max_value(args.i, args.m, args.M, args.target_sum)
```

Solution to Exercise 2

- **Decision Variables:**

$$x_j \quad \text{for } j = 0, 1, \ldots, i-1$$

$$b_j \quad \text{binary variables indicating whether } x_j \text{ is chosen}$$

$$L \quad \text{minimum value among the chosen variables } x_j$$

- **Objective Function:**

$$\text{Maximize } L$$

- **Constraints:**

$$\sum_{j=0}^{i-1} x_j = \text{target sum}$$

$$x_j \geq L \cdot b_j \quad \forall j$$

$$x_j \leq M \cdot b_j \quad \forall j$$

$$x_j \geq m \cdot b_j \quad \forall j$$

```
python3.11 ./min-max-gen.py 10 0 20 100

Optimale Lösung gefunden:
Verwendete Zahlen: [20.0, 20.0, 20.0, 20.0, 20.0]
Minimaler Wert (L): 20.0
```

```python
import argparse
from gurobipy import Model, GRB, quicksum

def maximize_min_value(i, m, M, target_sum):
    model = Model("maximize_min_value")

    x = model.addVars(i, vtype=GRB.INTEGER, name="x")
    b = model.addVars(i, vtype=GRB.BINARY, name="b")
    L = model.addVar(vtype=GRB.INTEGER, name="L")

    model.setObjective(L, GRB.MAXIMIZE)

    model.addConstr(quicksum(x[j] for j in range(i)) == target_sum,
         name="sum_constraint")
    for j in range(i):
        model.addConstr(x[j] >= L * b[j], name=f"min_constraint_{j}")
        model.addConstr(x[j] <= M * b[j], name=f"max_constraint_{j}")
        model.addConstr(x[j] >= m * b[j], name=f"max_constraint_{j}")


    model.optimize()

    if model.status == GRB.OPTIMAL:
        solution = [x[j].x for j in range(i) if b[j].x > 0.5]
        print("Optimal solution found:")
```

```
        print(f"Selected values: {solution}")
        print(f"Minimum value (L): {L.x}")
    else:
        print("No optimal solution found")

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Maximize the minimum value
         of a subset that sums to a target.')
    parser.add_argument('i', type=int, help='number of variables')
    parser.add_argument('m', type=int, help='minimum value for variables')
    parser.add_argument('M', type=int, help='maximum value for variables')
    parser.add_argument('target_sum', type=int, help='the target sum')

    args = parser.parse_args()
    maximize_min_value(args.i, args.m, args.M, args.target_sum)
```

Solution to Exercise 3

- **Decision Variables:**

$$x_j \quad \text{for } j = 0, 1, \ldots, 9 \quad (x_j \in \{5, 8, 15, 21, 22, 25, 26, 27, 36, 50\})$$

$$b_j \quad \text{binary variables indicating whether } x_j \text{ is chosen}$$

$$T \quad \text{maximum value among the chosen variables } x_j$$

- **Objective Function:**

$$\text{Minimize } T$$

- **Constraints:**

$$\sum_{j=0}^{9} x_j = \text{target sum}$$

$$x_j \leq T \quad \forall j$$

$$x_j \leq M \cdot b_j \quad \forall j$$

```
python3.11 min-max-spe.py 5 8 15 21 22 25 26 27 36 50 100

Optimale Lösung gefunden:
Verwendete Zahlen: [5, 8, 15, 21, 25, 26]
Maximaler Wert (T): 26.0
```

```python
import argparse
from gurobipy import Model, GRB, quicksum

def minimize_max_value_with_fixed_numbers(fixed_numbers, target_sum):
    n = len(fixed_numbers)

    model = Model("minimize_max_value_with_fixed_numbers")

    b = model.addVars(n, vtype=GRB.BINARY, name="b")
    T = model.addVar(vtype=GRB.INTEGER, name="T")

    model.setObjective(T, GRB.MINIMIZE)

    model.addConstr(quicksum(fixed_numbers[i] * b[i] for i in range(n)) ==
         target_sum, name="sum_constraint")
    for i in range(n):
        model.addConstr(fixed_numbers[i] * b[i] <= T,
            name=f"max_constraint_{i}")

    model.optimize()

    if model.status == GRB.OPTIMAL:
        solution = [fixed_numbers[i] for i in range(n) if b[i].x > 0.5]
        print("Optimal solution found:")
        print(f"Selected values: {solution}")
        print(f"Maximum value (T): {T.x}")
    else:
```

```
        print("No optimal solution found")

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Minimize the maximum value of a subset that sums to a target with fixed numbers.')
    parser.add_argument('fixed_numbers', metavar='N', type=int, nargs='+', help='an integer for the list of fixed numbers')
    parser.add_argument('target_sum', type=int, help='the target sum')

    args = parser.parse_args()
    minimize_max_value_with_fixed_numbers(args.fixed_numbers, args.target_sum)
```

Solution to Exercise 4

- **Decision Variables:**

$$x_j \quad \text{for } j = 0, 1, \ldots, 9 \quad (x_j \in \{5, 8, 15, 21, 22, 25, 26, 27, 36, 50\})$$

$$b_j \quad \text{binary variables indicating whether } x_j \text{ is chosen}$$

$$L \quad \text{minimum value among the chosen variables } x_j$$

- **Objective Function:**

$$\text{Maximize } L$$

- **Constraints:**

$$\sum_{j=0}^{9} x_j = \text{target sum}$$

$$x_j \geq L \cdot b_j \quad \forall j$$

```
python3.11 max-min-spe.py 5 8 15 21 22 25 26 27 36 50 100

Optimale Lösung gefunden:
Verwendete Zahlen: [22, 25, 26, 27]
```

```python
import argparse
from gurobipy import Model, GRB, quicksum

def maximize_min_value_with_fixed_numbers(fixed_numbers, target_sum):
    n = len(fixed_numbers)

    model = Model("maximize_min_value_with_fixed_numbers")

    b = model.addVars(n, vtype=GRB.BINARY, name="b")
    L = model.addVar(vtype=GRB.INTEGER, name="L")

    model.setObjective(L, GRB.MAXIMIZE)

    model.addConstr(quicksum(fixed_numbers[i] * b[i] for i in range(n)) ==
         target_sum, name="sum_constraint")
    for i in range(n):
        model.addConstr(fixed_numbers[i] * b[i] >= L,
             name=f"min_constraint_{i}")

    model.optimize()

    if model.status == GRB.OPTIMAL:
        solution = [fixed_numbers[i] for i in range(n) if b[i].x > 0.5]
        print("Optimal solution found:")
        print(f"Selected values: {solution}")
        print(f"Minimum value (L): {L.x}")
    else:
        print("No optimal solution found")

if __name__ == "__main__":
```

```
29    parser = argparse.ArgumentParser(description='Maximize the minimum value
          of a subset that sums to a target with fixed numbers.')
30    parser.add.argument('fixed_numbers', metavar='N', type=int, nargs='+',
          help='an integer for the list of fixed numbers')
31    parser.add.argument('target_sum', type=int, help='the target sum')
32
33    args = parser.parse_args()
34    maximize_min_value_with_fixed_numbers(args.fixed_numbers,
          args.target_sum)
```

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).

Solution

To find the second best solution, we introduce new decision variables and constraints as follows:

New Decision Variables

- $x2_{e_{id}}$: Flow variable for each hyperedge e_{id} in the second model.
- $b_{e_{id}}$: Binary variable indicating whether each hyperedge e_{id} is used in the second model (1 if used, 0 otherwise).

New Constraints

- **Linking Flow and Binary Variables:** For each hyperedge e_{id}, we introduce the following constraints to link the flow variables to the binary variables:

$$x2_{e_{id}} \leq b_{e_{id}} \cdot M \tag{1}$$

$$x2_{e_{id}} \geq b_{e_{id}} \tag{2}$$

  where M is a sufficiently large constant.

- **Different Hyperedges Constraint:** Define the set OE as all the hyperedges in the optimal solution with non-zero flow. To ensure the second solution does not use the same set of hyperedges as the optimal solution, we add the following constraint:

$$\sum_{e_{id} \in OE} b_{e_{id}} \leq |OE| - 1 \tag{3}$$

The objective function remains the same: Minimize the sum of flows in the second model independently. Here is the modified code:

```
# Create a new model for finding the second best solution
second_model = Model('SecondBestHypergraphFlow')

# Create variables
x2 = {}
for e_id in hyperedges:
    x2[e_id] = second_model.addVar(vtype=GRB.INTEGER, name=f'x2_{e_id}')

# Create binary variables to indicate whether each hyperedge is used
b = {}
for e_id in hyperedges:
    b[e_id] = second_model.addVar(vtype=GRB.BINARY, name=f'b_{e_id}')

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

# Flow conservation constraints for the second model
for v in vertices:
    inflow = quicksum(x2[e_id] for e_id, e in hyperedges.items() if v in
        e[1])
```

```python
    outflow = quicksum(x2[e_id] for e_id, e in hyperedges.items() if v in e[0])
    second_model.addConstr(inflow == outflow, name=f'flow_conservation_{v}')

# Specific inflow and outflow constraints for the second model
second_model.addConstr(x2[41] == 1, name='inflow_H2O')
second_model.addConstr(x2[42] == 6, name='inflow_Ribulose_5_Phosphate')
second_model.addConstr(x2[43] == 5, name='outflow_Fructose_6_Phosphate')
second_model.addConstr(x2[44] == 1, name='outflow_p_0_11')

# Big-M constraint to link binary variables to the flow variables
M = 1e6 # A sufficiently large constant
for e_id in hyperedges:
    second_model.addConstr(x2[e_id] <= b[e_id] * M)
    second_model.addConstr(x2[e_id] >= b[e_id])

# Ensure the second solution does not use the same set of hyperedges
second_model.addConstr(quicksum(b[e_id] for e_id in optimal_solution.keys()) <= len(optimal_solution) - 1, name='different_hyperedges')

# Objective function: Minimize the sum of flows in the second model independently
second_model.setObjective(quicksum(x2[e_id] for e_id in hyperedges), GRB.MINIMIZE)

# Optimize again for the second best solution
second_model.optimize()

# Get the second best solution
if second_model.status == GRB.Status.OPTIMAL:
    second_best_solution = {e_id: x2[e_id].X for e_id in x2 if x2[e_id].X > 0}
    binary_solution = {e_id: b[e_id].X for e_id in b if b[e_id].X > 0}

    # Output the results for the second best solution
    print("\nSecond Best Solution:")
    for e_id, flow in second_best_solution.items():
        tails, heads = hyperedges[e_id]
        print(f'Hyperedge {e_id}: Flow = {flow}, Tails = {tails}, Heads = {heads}')

    # Output the binary variables for the second best solution
    print("\nBinary Variables:")
    for e_id, val in binary_solution.items():
        print(f'Binary Variable b_{e_id} = {val}')
else:
    print('No optimal solution found for the second best model.')
```

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.

Solution: Finding Optimal Pathways with Minimal Energy Barriers

a)

Decision Variables

- $x_{e_{id}}$: Flow variable for each hyperedge e_{id}.
- $b_{e_{id}}$: Binary variable indicating whether each hyperedge e_{id} is used in the solution (1 if used, 0 otherwise).
- z: Auxiliary variable representing the maximum $r_{e_{id}}$ value among the used hyperedges.

Constraints

- **Flow Conservation:** For each vertex v:

$$\sum_{e_{id}: v \in \text{heads of } e_{id}} x_{e_{id}} = \sum_{e_{id}: v \in \text{tails of } e_{id}} x_{e_{id}} \tag{4}$$

- **Specific Inflow and Outflow Constraints:**

$$x_{41} = 1, \quad x_{42} = 6, \quad x_{43} = 5, \quad x_{44} = 1 \tag{5}$$

- **Linking Flow and Binary Variables:**

$$x_{e_{id}} \leq b_{e_{id}} \cdot M, \quad \forall e_{id} \in E \tag{6}$$

$$x_{e_{id}} \geq b_{e_{id}}, \quad \forall e_{id} \in E \tag{7}$$

- **Maximum Energy Barrier:**

$$z \geq r_{e_{id}} \cdot b_{e_{id}}, \quad \forall e_{id} \in E \tag{8}$$

Objective Function

Minimize the maximum $r_{e_{id}}$ value among the used hyperedges:

$$\text{minimize} \quad z \tag{9}$$

b) and code

Decision Variables

- $x2_{e_{id}}$: Flow variable for each hyperedge e_{id} in the second model.
- $b2_{e_{id}}$: Binary variable indicating whether each hyperedge e_{id} is used in the second model (1 if used, 0 otherwise).
- $z2$: Auxiliary variable representing the maximum $r_{e_{id}}$ value among the used hyperedges in the second model.

Constraints

- **Flow Conservation:** For each vertex v:

$$\sum_{e_{id}: v \in \text{heads of } e_{id}} x2_{e_{id}} = \sum_{e_{id}: v \in \text{tails of } e_{id}} x2_{e_{id}} \tag{10}$$

- **Specific Inflow and Outflow Constraints:**

$$x2_{41} = 1, \quad x2_{42} = 6, \quad x2_{43} = 5, \quad x2_{44} = 1 \tag{11}$$

- **Linking Flow and Binary Variables:**

$$x2_{e_{id}} \leq b2_{e_{id}} \cdot M, \quad \forall e_{id} \in E \tag{12}$$

$$x2_{e_{id}} \geq b2_{e_{id}}, \quad \forall e_{id} \in E \tag{13}$$

- **Maximum Energy Barrier:**

$$z2 \geq r_{e_{id}} \cdot b2_{e_{id}}, \quad \forall e_{id} \in E \tag{14}$$

- **Different Hyperedges Constraint:** Define the set OE as all the hyperedges in the optimal solution with non-zero flow:

$$\sum_{e_{id} \in OE} b2_{e_{id}} \leq |OE| - 1 \tag{15}$$

Objective Function

Minimize the maximum $r_{e_{id}}$ value among the used hyperedges in the second model:

$$\text{minimize} \quad z2 \tag{16}$$

Solution Code

```python
import random
from gurobipy import Model, GRB, quicksum
import sys

# Function to set the random seed
def set_random_seed(seed=None):
    if seed is not None:
        random.seed(seed)
        print(f"Random seed set to: {seed}")
    else:
        random.seed()
        print("Random seed set to system default.")

# Set the random seed if provided as a command line argument
if len(sys.argv) > 1:
    seed = int(sys.argv[1])
    set_random_seed(seed)
else:
    set_random_seed(42) # Default seed for reproducibility

# Define the hyperedges as given
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}'], [])
}
```

```
52
53 # Assign r_values, with edges 41, 42, 43, and 44 having r_{e_{id}} = 0
54 r_values = {e_id: random.random() if e_id not in [41, 42, 43, 44] else 0.0
       for e_id in hyperedges}
55
56 # Create the model
57 model = Model('HypergraphFlow')
58
59 # Create variables
60 x = {}
61 for e_id in hyperedges:
62     x[e_id] = model.addVar(vtype=GRB.INTEGER, name=f'x_{e_id}')
63
64 # Create binary variables to indicate whether each hyperedge is used
65 b = {}
66 for e_id in hyperedges:
67     b[e_id] = model.addVar(vtype=GRB.BINARY, name=f'b_{e_id}')
68
69 # Auxiliary variable for the max r_{e_{id}} value among used hyperedges
70 z = model.addVar(vtype=GRB.CONTINUOUS, name='z')
71
72 # Update model to integrate new variables
73 model.update()
74
75 # Flow conservation constraints
76 vertices = set(v for e in hyperedges.values() for v in e[0] + e[1])
77
78 for v in vertices:
79     inflow = quicksum(x[e_id] for e_id, e in hyperedges.items() if v in e[1])
80     outflow = quicksum(x[e_id] for e_id, e in hyperedges.items() if v in
           e[0])
81     model.addConstr(inflow == outflow, name=f'flow_conservation_{v}')
82
83 # Specific inflow and outflow constraints
84 model.addConstr(x[41] == 1, name='inflow_H2O')
85 model.addConstr(x[42] == 6, name='inflow_Ribulose_5_Phosphate')
86 model.addConstr(x[43] == 5, name='outflow_Fructose_6_Phosphate')
87 model.addConstr(x[44] == 1, name='outflow_p_0_11')
88
89 # Big-M constraint to link binary variables to the flow variables
90 M = 1e6 # A sufficiently large constant
91 for e_id in hyperedges:
92     model.addConstr(x[e_id] <= b[e_id] * M)
93     model.addConstr(x[e_id] >= b[e_id])
94     model.addConstr(z >= r_values[e_id] * b[e_id])
95
96 # Objective function: Minimize the max r_{e_{id}} value among used hyperedges
97 model.setObjective(z, GRB.MINIMIZE)
98
99 # Optimize the model
100 model.optimize()
101
102 # Get the optimal solution
103 optimal_solution = {e_id: x[e_id].X for e_id in x if x[e_id].X > 0}
104 binary_solution = {e_id: b[e_id].X for e_id in b if b[e_id].X > 0}
105
106 # Output the results for the optimal solution
```

```
print("Optimal Solution:")
for e_id, flow in optimal_solution.items():
    heads, tails = hyperedges[e_id]
    print(f'Hyperedge {e_id}: Flow = {flow}, Heads = {heads}, Tails =
        {tails}, r_value = {r_values[e_id]}')

# Output the binary variables for the solution
print("\nBinary Variables:")
for e_id, val in binary_solution.items():
    print(f'Binary Variable b_{e_id} = {val}')

# Output the max r_{e_id} value
print(f'\nMax r_{e_id} value used: {z.X}')

# Create a new model for finding the second best solution
second_model = Model('SecondBestHypergraphFlow')

# Create variables
x2 = {}
for e_id in hyperedges:
    x2[e_id] = second_model.addVar(vtype=GRB.INTEGER, name=f'x2_{e_id}')

# Create binary variables to indicate whether each hyperedge is used
b2 = {}
for e_id in hyperedges:
    b2[e_id] = second_model.addVar(vtype=GRB.BINARY, name=f'b2_{e_id}')

# Auxiliary variable for the max r_{e_{id}} value among used hyperedges
z2 = second_model.addVar(vtype=GRB.CONTINUOUS, name='z2')

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

# Flow conservation constraints for the second model
for v in vertices:
    inflow = quicksum(x2[e_id] for e_id, e in hyperedges.items() if v in
        e[1])
    outflow = quicksum(x2[e_id] for e_id, e in hyperedges.items() if v in
        e[0])
    second_model.addConstr(inflow == outflow,
        name=f'flow_conservation_{v}_second')

# Specific inflow and outflow constraints for the second model
second_model.addConstr(x2[41] == 1, name='inflow_H2O_second')
second_model.addConstr(x2[42] == 6,
     name='inflow_Ribulose_5_Phosphate_second')
second_model.addConstr(x2[43] == 5,
     name='outflow_Fructose_6_Phosphate_second')
second_model.addConstr(x2[44] == 1, name='outflow_p_0_11_second')

# Big-M constraint to link binary variables to the flow variables for the
     second model
for e_id in hyperedges:
    second_model.addConstr(x2[e_id] <= b2[e_id] * M)
    second_model.addConstr(x2[e_id] >= b2[e_id])
    second_model.addConstr(z2 >= r_values[e_id] * b2[e_id])

```

```python
# Ensure the second solution does not use the same set of hyperedges
used_hyperedges = [e_id for e_id in optimal_solution.keys()]
second_model.addConstr(quicksum(b2[e_id] for e_id in used_hyperedges) <= len(used_hyperedges) - 1, name='different_hyperedges_second')

# Objective function: Minimize the max r_{e_{id}} value among used hyperedges in the second model
second_model.setObjective(z2, GRB.MINIMIZE)

# Optimize the second model
second_model.optimize()

# Get the second best solution
if second_model.status == GRB.Status.OPTIMAL:
    second_best_solution = {e_id: x2[e_id].X for e_id in x2 if x2[e_id].X > 0}
    binary_solution_second = {e_id: b2[e_id].X for e_id in b2 if b2[e_id].X > 0}

    # Output the results for the second best solution
    print("\nSecond Best Solution:")
    for e_id, flow in second_best_solution.items():
        heads, tails = hyperedges[e_id]
        print(f'Hyperedge {e_id}: Flow = {flow}, Heads = {heads}, Tails = {tails}, r_value = {r_values[e_id]}')

    # Output the binary variables for the second best solution
    print("\nBinary Variables (Second Best Solution):")
    for e_id, val in binary_solution_second.items():
        print(f'Binary Variable b2_{e_id} = {val}')

    # Output the max r_{e_id} value for the second best solution
    print(f'\nMax r_{e_id} value used in the second best solution: {z2.X}')
else:
    print('No optimal solution found for the second best model.')
```