Flow Assignment Report

Yang Zhang 37594777

Problem Formulation and Explanation

1. Decision Variables:

x_{ikj}: The amount of flow starts from source node S_i to tansits node T_k then arrives to destination node D_j .

c_{ik}: $The\ capacity\ from$ source node S_i to tansits node T_k.

d_{kj}: $The\ capacity\ from$ tansits node T_k to destination node D_j.

b_{ikj}: Binary variable means whether the path through the transit nodeT_k from source S_i to destination D_j is used (1 if used, 0 otherwise).

2. Objective Function: Minimize the maximum load on T_k to balance loads for all transit nodes to implement traffic load balancing (ie: there are no any nodes with too much load), which is beneficial and efficient and reliable to prevent overload issues.

$$Minimize\ max_k\ (\sum_{i=1}^{X}\sum_{j=1}^{Z} x_{ikj}):$$

max_k: Calculate load for all tansits node T_k then choose the max load.

$\sum_{i=1}^{X}\sum_{j=1}^{Z} x_{ikj}$: Calculate the all flow x_{ikj} from source node S_i to to destination node D_j .

3. Constrains:

(1) Flow Constraints: ensure each amount of flow from source node to destination node via transit node to be assigned to two paths.

$$\sum_{k=1}^{Y} x_{ikj} = h_{ij}, \forall i \in \{1, 2, \dots, X\}, \forall \in \{1, 2, \dots, Z\}$$

where $h_{ij} = i + j\, is\ the\ demand\ from\ source\ S_i\ to\ destination\ D_j$.

(2) Capacity Constraints:

$$x_{ikj} \leq c_{ik}, \forall i, \forall k$$

$$x_{ikj} \leq d_{kj}, \forall k, \forall j$$

(3) Two Paths:

Ensures two paths should be used for each demand:

$$\sum_{k=1}^{Y} b_{ikj} = 2, \forall i, \forall j$$

(4) Link the binary variables b_{ikj} to the flow:

$$x_{ikj} \leq M * b_{ikj}, \forall i, \forall j$$

If x_{ikj} > 0 then b_{ikj} = 1, if x_{ikj} < 0 then b_{ikj} = 0. M is a big constant that is used in LP to effectively make a constraint non-binding when a condition is met.

4. Bounds:

$$x_{ikj} \geq 0, b_{ikj} \in \{0, 1\}$$

Results for the CPLEX Execution Time

Y	3	4	5	6	7
CPLEX Execution Time	0.27s	0.07s	0.05s	0.09s	0.08s
Transit Load Nodes	130.67	98.0	78.4	65.33	56.0
Highest Capacity	4.67	3.5	2.8	2.33	14

Results of X = 7, Z= 7 and Y ∈ {3, 4, 5, 6, 7}

1. CPLEX execution time: The execution time decreases as the 'Y' value increases. This may be related to the complexity of the linear programming problem. In general, the larger the value of 'Y', the fewer constraints and variables involved, which reduces the problem size and increases the computation speed.

2. Transit load nodes: The table shows that the maximum transmission load decreases as the value of 'Y' increases. This may mean that the resources and loads or constraints assigned to each node become more balanced as 'Y' increases. The solution of the problem reduces the amount of data that needs to be processed.

3. Maximum capacity: Large variability in maximum capacity. This indicates that there are significant differences in the calculation methods of maximum capacity and variables included in the models corresponding to different 'Y' values. In particular, the maximum capacity at 'Y=7' is abnormally high (14), which may be an extreme value due to certain constraints or variable settings.

Overall, as the value of 'Y' increases, CPLEX runs more efficiently and the problem size and complexity decrease. These observed trends and results can help you understand and evaluate the process of solving optimization problems using different parameter settings.

Appendix

Source codes:

```
import subprocess
import sys



def create_aim_obj(file):
    """写入目标函数部分。"""
    file.write("Minimize\n")
    file.write("obj: max_load\n")  # 定义目标函数以最小化最大负载



def create_constraints(file, X, Y, Z):
    """编写约束以确保适当的流量和容量管理。"""
    file.write("Subject To\n")

    # 每个传输节点的最大负载约束
    for k in range(1, Y + 1):
        transit_loads = " + ".join(f"x{i}{k}{j}" for i in range(1, X + 1) for j in range(1, Z + 1))
        file.write(f"transit_load_{k}: {transit_loads} - max_load <= 0\n")

    # 确保满足需求并且恰好分配在两条路径上
```

```
    for i in range(1, X + 1):
        for j in range(1, Z + 1):
            hj = i + j
            file.write(f"demand_{i}_{j}: " + " + ".join(f"x{i}{k}{j}" for k in range(1, Y + 1)) + f" = {hj}\n")
            file.write(f"paths_{i}_{j}: " + " + ".join(f"b{i}{k}{j}" for k in range(1, Y + 1)) + " = 2\n")
            for k in range(1, Y + 1):
                file.write(f"link_flow_{i}{k}{j}: x{i}{k}{j} <= {hj} b{i}{k}{j}\n")  # 避免乘法运算


def create_bounds(file, X, Y, Z):
    """定义流变量的界限并声明二进制变量。"""
    file.write("Bounds\n")
    for i in range(1, X + 1):
        for j in range(1, Z + 1):
            for k in range(1, Y + 1):
                file.write(f" 0 <= x{i}{k}{j} <= {i + j}\n")  # 设置流变量的界限
    file.write("Generals\n")
    for i in range(1, X + 1):
        for j in range(1, Z + 1):
            for k in range(1, Y + 1):
                file.write(f" b{i}{k}{j}\n")  # 将二进制变量声明为一般整数
```

```python
    file.write("End\n")


def run_cplex(lp_filename):
    """运行CPLEX并捕获输出。"""
    command = f'/usr/bin/time -p cplex -c "read {lp_filename}" "optimize"
"display solution variables -"'
    result = subprocess.run(command, shell=True, text=True, capture_output=True)
    print("CPLEX Runtime:", result.stderr)
    print("CPLEX Output:", result.stdout)
    return result.stdout


def main():
    """主函数处理命令行输入。"""
    if len(sys.argv) != 5:
        print("Usage: python generate_lp.py X Y Z filename.lp")
        return
    X, Y, Z = int(sys.argv[1]), int(sys.argv[2]), int(sys.argv[3])
    filename = sys.argv[4]
    with open(filename, "w") as file:
        create_aim_obj(file)
        create_constraints(file, X, Y, Z)
```

```
        create_bounds(file, X, Y, Z)

    cplex_result = run_cplex(filename)




if __name__ == "__main__":

    main()
```

323.lp:

```
Minimize

obj: max_load

Subject To

transit_load_1: x111 + x112 + x113 + x211 + x212 + x213 + x311 + x312 + x313 -
max_load <= 0

transit_load_2: x121 + x122 + x123 + x221 + x222 + x223 + x321 + x322 + x323 -
max_load <= 0

demand_1_1: x111 + x121 = 2

paths_1_1: b111 + b121 = 2

link_flow_111: x111 <= 2 b111

link_flow_121: x121 <= 2 b121

demand_1_2: x112 + x122 = 3

paths_1_2: b112 + b122 = 2

link_flow_112: x112 <= 3 b112

link_flow_122: x122 <= 3 b122
```

```
demand_1_3: x113 + x123 = 4

paths_1_3: b113 + b123 = 2

link_flow_113: x113 <= 4 b113

link_flow_123: x123 <= 4 b123

demand_2_1: x211 + x221 = 3

paths_2_1: b211 + b221 = 2

link_flow_211: x211 <= 3 b211

link_flow_221: x221 <= 3 b221

demand_2_2: x212 + x222 = 4

paths_2_2: b212 + b222 = 2

link_flow_212: x212 <= 4 b212

link_flow_222: x222 <= 4 b222

demand_2_3: x213 + x223 = 5

paths_2_3: b213 + b223 = 2

link_flow_213: x213 <= 5 b213

link_flow_223: x223 <= 5 b223

demand_3_1: x311 + x321 = 4

paths_3_1: b311 + b321 = 2

link_flow_311: x311 <= 4 b311

link_flow_321: x321 <= 4 b321

demand_3_2: x312 + x322 = 5

paths_3_2: b312 + b322 = 2

link_flow_312: x312 <= 5 b312
```

```
link_flow_322: x322 <= 5 b322

demand_3_3: x313 + x323 = 6

paths_3_3: b313 + b323 = 2

link_flow_313: x313 <= 6 b313

link_flow_323: x323 <= 6 b323

Bounds

0 <= x111 <= 2

0 <= x121 <= 2

0 <= x112 <= 3

0 <= x122 <= 3

0 <= x113 <= 4

0 <= x123 <= 4

0 <= x211 <= 3

0 <= x221 <= 3

0 <= x212 <= 4

0 <= x222 <= 4

0 <= x213 <= 5

0 <= x223 <= 5

0 <= x311 <= 4

0 <= x321 <= 4

0 <= x312 <= 5

0 <= x322 <= 5

0 <= x313 <= 6
```

```
0 <= x323 <= 6

Generals

b111

b121

b112

b122

b113

b123

b211

b221

b212

b222

b213

b223

b311

b321

b312

b322

b313

b323

End
```