Trabalho de Otimização Inteira

SME0110 - Programação Matemática

Grupo 9

- **Ana Vitória Freitas (anavitoria_gouvea@usp.br) - 1370196**
- **Eduardo Rodrigues Amaral (eduardoroamaral@usp.br) - 11735021**

- **Fernando Henrique Paes Generich (fernando_gene@usp.br) - 11795342**
- **Guilherme Rios (guilhermerios@usp.br) - 11222839**
- **Pedro Augusto Ribeiro Gomes (parg07@usp.br) - 11819125**

Enunciado

O problema de localização de facilidades é um problema clássico de otimização que pode ser abordado de várias formas. Abaixo, apresentamos duas abordagens clássicas. A primeira visa atender a demanda dos clientes minimizando a soma dos custos fixos de instalação das facilidades e dos custos de transportes. Na segunda, o objetivo é maximizar o lucro gerado pelas facilidades abertas.

Minimizando Custos

O problema de localização de facilidades que visa atender a demanda dos clientes com mínimo custo pode ser descrito como um problema de otimização linear inteira dado por $(1) - (5)$:

$$minimize\ z = \Sigma_{i=1}^{n} f_i y_i + \Sigma_{i=1}^{n}\Sigma_{j=1}^{m} c_{ij} x_{ij} \quad (1)$$

$$sujeito\ a$$

$$\Sigma_{i=1}^{n} x_{ij} = 1 \qquad j = 1, ..., m \qquad (2)$$

$$\Sigma_{j=1}^{m} d_j x_{ij} \leq Cap_i\ y_i \qquad i = 1, ..., n \qquad (3)$$

$$y_i \in \{0, 1\} \qquad i = 1, ..., n \qquad (4)$$

$$0 \leq x_{ij} \leq 1 \quad i = 1, ..., n; j = 1, ..., m \qquad (5)$$

em que:

n: é o número de locais onde podem ser instaladas as facilidades;

m: é o número de clientes que devem ser atendidos;

f_i: é o custo fixo da facilidade $i \in \{1, ..., n\}$ se ela for aberta;

c_{ij}: é o custo de transporte da facilidade $i (i \in \{1, ...n\})$ para o cliente $j (j \in \{1, ..., m\})$;

d_j: é a demanda do cliente $j (j \in \{1, ..., m\})$;

Cap_i: é a capacidade da facilidade $i (i \in \{1, ...n\})$;

y_i: é uma variável binária que assume valor 1 se a facilidade $i (i \in \{1, ...n\})$ é aberta, e assume o valor 0 caso contrário;

x_{ij}: é uma variável contínua que corresponde a fração da demanda do cliente $j (j \in \{1, ..., m\})$ atendida pela facilidade $i (i \in \{1, ...n\})$ $(0 \leq x_{ij} \leq 1)$.

A função objetivo (1) visa minimizar o custo total de distribuição, ou seja, a soma dos custos fixos das facilidades abertas mais a soma dos custos de transporte das facilidades para os clientes. As restrições (2) garantem que todos os clientes tenham sua demanda atendida. As restrições (3) asseguram que a capacidade de cada uma das facilidades é respeitada. O domínio das variáveis é definido em (4) e (5).

Maximizando Lucros

O problema de localização de facilidades também pode ter como objetivo a maximização dos lucros. Neste caso, o problema consiste em dado um conjunto de facilidades e um conjunto de clientes, decidir quais facilidades instalar e quais clientes atender de forma a maximizar os lucros. Este problema pode ser modelado como descrito a seguir:

$$maximize\ z = \Sigma_{i=1}^{n}\Sigma_{j=1}^{m} L_{ij} x_{ij} - \Sigma_{i=1}^{n} f_i y_i \qquad (6)$$

$$sujeito\ a$$

$$\Sigma_{i=1}^{n} x_{ij} \leq 1 \qquad j = 1, ..., m \qquad (7)$$

$$\Sigma_{j=1}^{m} d_j x_{ij} \leq Cap_i\ y_i \qquad i = 1, ..., n \qquad (8)$$

$$y_i \in \{0, 1\} \qquad i = 1, ..., n \qquad (9)$$

$$0 \leq x_{ij} \leq 1 \quad i = 1, ..., n; j = 1, ..., m \qquad (10)$$

em que:

n: é o número de locais onde podem ser instaladas as facilidades;

m: é o número de clientes que devem ser atendidos;

f_i: é o custo fixo da facilidade $i \in \{1, ..., n\}$ se ela for aberta;

L_{ij}: é o lucro obtido se a facilidade $i (i \in \{1, ...n\})$ atender ao cliente $j (j \in \{1, ..., m\})$;

d_j: é a demanda do cliente $j (j \in \{1, ..., m\})$;

Cap_i: é a capacidade da facilidade $i (i \in \{1, ...n\})$;

y_i: é uma variável binária que assume valor 1 se a facilidade $i (i \in \{1, ...n\})$ é aberta, e assume o valor 0 caso contrário;

x_{ij}: é uma variável contínua que corresponde a fração da demanda do cliente $j (j \in \{1, ..., m\})$ atendida pela facilidade $i (i \in \{1, ...n\})$ $(0 \leq x_{ij} \leq 1)$.

A função objetivo (6) visa maximizar os lucros, ou seja, a soma do lucro obtido pelas facilidades menos os custos fixos associados a elas. As restrições (7) asseguram que a demanda máxima dos clientes é respeitada. As restrições (8) asseguram que a capacidade de cada uma das facilidades é respeitada. O domínio das variáveis é definido em (10) e (9)

Tarefa 1

Escreva o modelo de localização de facilidades que minimiza os custos em linguagem de modelagem.

Modelo GUROBI

```
# n ->  número de locais onde facilidades podem ser instaladas
# m -> número de clientes que devem ser atendidos
# capacities -> capacidade de cada facilidade
# demands -> demanda de cada cliente
# fixedCosts -> custo fixo de cada facilidade
# transportCosts -> custo de transporte de cada facilidade para cada cliente

def getMinCostModelGurobi(data):
    n, m = data['dimensions']
    capacities = data['capacities']
    demands = data['demands']
    fixedCosts = data['fixedCosts']
    transportCosts = data['transportCosts']

    model = gp.Model(env=gurobiEnv)

    # Definindo variáveis para o modelo
    # Facilidades abertas
    open = model.addVars(n,
                         vtype=GRB.BINARY,
                         obj=fixedCosts,
                         name="open")

    # Fração da demanda do cliente j atendida pela facilidade i
    x = model.addVars(n,
                      m,
                      vtype=GRB.CONTINUOUS,
                      lb=0,
                      ub=1,
                      name='coveredClients')

    # Definindo as restrições do problema
    # Garantindo que todos os clientes tenham sua demanda atendida (2)
    model.addConstrs((gp.quicksum(x[(i,j)] for i in range(n)) == 1 for j in range(m)), name='clientDemand')

    # Garantindo que a demanda total atendida pelas facilidades abertas seja menor ou igual à capacidade dessas facilidades (3)
    for i in range(n):
        model.addConstr(gp.quicksum(demands[j] * x[i,j] for j in range(m)) <= capacities[i] * open[i], name=f'capacityConstr_{i}')

    # Definindo função objetivo (1)
    model.setObjective(
        (gp.quicksum(fixedCosts[i] * open[i] for i in range(n))
        + gp.quicksum(transportCosts[i][j] * x[i,j] for i in range(n) for j in range(m)) ), GRB.MINIMIZE)

    return model
```

Modelo SCIP

```
# n ->  número de locais onde facilidades podem ser instaladas
# m -> número de clientes que devem ser atendidos
# capacities -> capacidade de cada facilidade
# demands -> demanda de cada cliente
# fixedCosts -> custo fixo de cada facilidade
# transportCosts -> custo de transporte de cada facilidade para cada cliente

def getMinCostModelSCIP(data):
    n, m = data['dimensions']
    capacities = data['capacities']
    demands = data['demands']
    fixedCosts = data['fixedCosts']
    transportCosts = data['transportCosts']

    model = scip.Model()

    # Definindo variáveis para o modelo
```

```
    # Facilidades abertas
    open = {}
    for i in range(n):
        open[i] = model.addVar(vtype='BINARY', obj=fixedCosts[i], name=f"open_{i}")

    # Fração da demanda do cliente j atendida pela facilidade i
    x = {}
    for i in range(n):
        for j in range(m):
            x[i, j] = model.addVar(vtype='CONTINUOUS', lb=0, ub=1, name=f'coveredClients_{i}_{j}')

    # Definindo as restrições do problema
    # Garantindo que todos os clientes tenham sua demanda atendida (2)
    for j in range(m):
        model.addCons(scip.quicksum(x[(i, j)] for i in range(n)) == 1, name=f'clientDemand_{j}')

    # Garantindo que a demanda total atendida pelas facilidades abertas seja menor ou igual à capacidade dessas facilidades (3)
    for i in range(n):
        model.addCons(scip.quicksum(demands[j] * x[i, j] for j in range(m)) <= capacities[i] * open[i], name=f'capacityConstr_{i}')

    # Definindo função objetivo (1)
    model.setObjective(
        scip.quicksum(fixedCosts[i] * open[i] for i in range(n))
        + scip.quicksum(transportCosts[i][j] * x[i, j] for i in range(n) for j in range(m)), "minimize")

    model.data = {
        'open': open,
        'x': x
    }

    return model
```

Tarefa 2

Sabemos que as restrições (3) garantem que as variáveis y_i assumam valores 0 ou 1 em todas as soluções factíveis. No entanto, quando é resolvida a relaxação linear do problema, estas variáveis podem assumir valores reais. O limitante dual do problema poderia ser melhorado se novas restrições fossem adicionadas ao problema, por exemplo:

$$x_{ij} \leq y_i$$

Explique brevemente por que essas restrições poderiam trazer melhorias. Para as instâncias disponibilizadas resolva o problema linearmente relaxado considerando o modelo (1) - (5). Em seguida, resolva novamente as instâncias adicionando as restrições (11) e reescrevendo as restrições (3) como:

$$\sum_{j=1}^{m} d_j x_{ij} \leq Cap_i \qquad i = 1, ..., n.$$

Código

Observe que em ambos os modelos as alterações requeridas no final do enunciado acima, são realizadas através do parâmetro `extraRestrictions`

Modelo Relaxado e Customizado - GUROBI

```
# n ->  número de locais onde facilidades podem ser instaladas
# m -> número de clientes que devem ser atendidos
# capacities -> capacidade de cada facilidade
# demands -> demanda de cada cliente
# fixedCosts -> custo fixo de cada facilidade
# transportCosts -> custo de transporte de cada facilidade para cada cliente

def getMinCostModelRelaxedGurobi(data, extraRestrictions):
    n, m = data['dimensions']
    capacities = data['capacities']
    demands = data['demands']
    fixedCosts = data['fixedCosts']
```

```
    transportCosts = data['transportCosts']

    model = gp.Model(env=gurobiEnv)

    # Definindo variáveis para o modelo
    # Facilidades abertas (variável relaxada)
    open = model.addVars(n,
                         vtype=GRB.CONTINUOUS,
                         lb=0,
                         ub=1,
                         obj=fixedCosts,
                         name="open")

    # Fração da demanda do cliente j atendida pela facilidade i
    x = model.addVars(n,
                      m,
                      vtype=GRB.CONTINUOUS,
                      lb=0,
                      ub=1,
                      name='coveredClients')

    # Definindo as restrições do problema
    # Garantindo que todos os clientes tenham sua demanda atendida (2)
    model.addConstrs((gp.quicksum(x[(i,j)] for i in range(n)) == 1 for j in range(m)), name='clientDemand')

    # Garantindo que a demanda total atendida pelas facilidades abertas seja menor ou igual à capacidade dessas facilidades (3)
    if not extraRestrictions:
        for i in range(n):
            model.addConstr(gp.quicksum(demands[j] * x[i,j] for j in range(m)) <= capacities[i] * open[i], name=f'capacityConstr_{i}')
    else:
        # Restrição 3 reescrita
        for i in range(n):
            model.addConstr(gp.quicksum(demands[j] * x[i,j] for j in range(m)) <= capacities[i], name=f'customCapacityConstr_{i}')

    if extraRestrictions:
        # Adicionando a restrição sugerida (11)
        for i in range(n):
            for j in range(m):
                model.addConstr(x[i,j] <= open[i], name=f'newConstr_{i}_{j}')

    # Definindo função objetivo (1)
    model.setObjective(
        (gp.quicksum(fixedCosts[i] * open[i] for i in range(n))
        + gp.quicksum(transportCosts[i][j] * x[i,j] for i in range(n) for j in range(m)) ), GRB.MINIMIZE)

    return model
```

Modelo Relaxado e Customizado - SCIP

```
# n ->  número de locais onde facilidades podem ser instaladas
# m -> número de clientes que devem ser atendidos
# capacities -> capacidade de cada facilidade
# demands -> demanda de cada cliente
# fixedCosts -> custo fixo de cada facilidade
# transportCosts -> custo de transporte de cada facilidade para cada cliente

def getMinCostModelRelaxedSCIP(data, extraRestrictions):
    n, m = data['dimensions']
    capacities = data['capacities']
    demands = data['demands']
    fixedCosts = data['fixedCosts']
    transportCosts = data['transportCosts']

    model = scip.Model()

    # Definindo variáveis para o modelo
    # Facilidades abertas (variável relaxada)
    open = {}
    for i in range(n):
        open[i] = model.addVar(vtype='CONTINUOUS', lb=0, ub=1, obj=fixedCosts[i], name=f"open_{i}")

    # Fração da demanda do cliente j atendida pela facilidade i
    x = {}
    for i in range(n):
```

```
        for j in range(m):
            x[i, j] = model.addVar(vtype='CONTINUOUS', lb=0, ub=1, name=f'coveredClients_{i}_{j}')

    # Definindo as restrições do problema
    # Garantindo que todos os clientes tenham sua demanda atendida (2)
    for j in range(m):
        model.addCons(scip.quicksum(x[(i, j)] for i in range(n)) == 1, name=f'clientDemand_{j}')

    # Garantindo que a demanda total atendida pelas facilidades abertas seja menor ou igual à capacidade dessas facilidades (3)
    if not extraRestrictions:
      for i in range(n):
          model.addCons(scip.quicksum(demands[j] * x[i, j] for j in range(m)) <= capacities[i] * open[i], name=f'capacityConstr_{i}')
    else:
      # Restrição 3 reescrita
      for i in range(n):
          model.addCons(scip.quicksum(demands[j] * x[i, j] for j in range(m)) <= capacities[i], name=f'customCapacityConstr_{i}')

    if extraRestrictions:
        # Adicionando a restrição sugerida (11)
        for i in range(n):
            for j in range(m):
                model.addCons(x[i,j] <= open[i], name=f'newConstr_{i}_{j}')

    # Definindo função objetivo (1)
    model.setObjective(
        scip.quicksum(fixedCosts[i] * open[i] for i in range(n))
        + scip.quicksum(transportCosts[i][j] * x[i, j] for i in range(n) for j in range(m)), "minimize")

    model.data = {
        'open': open,
        'x': x
    }

    return model
```

Resultados

- **SCIP vs Gurobi - Modelo Relaxado**

  Obs.: A tabela abaixo foi divida em 3 partes, prezando pela formatação do pdf

Instância	Limitante Dual SCIP	Limitante Dual Gurobi	Diferença Limitante Dual	Limitante Primal SCIP	Limitante Primal Gurobi
1	6.90E+04	6.90E+04	1.16E-10	6.90E+04	6.90E+04
2	7.58E+04	7.58E+04	8.73E-11	7.58E+04	7.58E+04
3	1.15E+05	1.15E+05	0.00E+00	1.15E+05	1.15E+05
4	1.35E+05	1.35E+05	0.00E+00	1.35E+05	1.35E+05
5	1.62E+05	1.62E+05	8.73E-11	1.62E+05	1.62E+05

Instância	Diferença Limitante Primal	Alcançou Tempo Limite SCIP	Alcançou Tempo Limite Gurobi	Gap SCIP	Gap Gurobi
1	1.16E-10	Não	Não	0.00%	0.00%
2	8.73E-11	Não	Não	0.00%	0.00%
3	0.00E+00	Não	Não	0.00%	0.00%
4	0.00E+00	Não	Não	0.00%	0.00%
5	8.73E-11	Não	Não	0.00%	0.00%

Instância	Tempo de Execução SCIP	Tempo de Execução Gurobi	Diferença Tempo de Execução	Otimalidade SCIP	Otimalidade Gurobi
1	00:00:18.27	00:00:00.88	00:00:17.40	Sim	Sim
2	00:00:29.93	00:00:01.30	00:00:28.63	Sim	Sim
3	00:01:12.81	00:00:02.74	00:01:10.07	Sim	Sim
4	00:04:04.98	00:00:03.90	00:04:01.08	Sim	Sim

5	00:13:10.66	00:00:05.85	00:13:04.81	Sim	Sim

- **SCIP vs Gurobi - Modelo Relaxado com Restrições Customizadas**

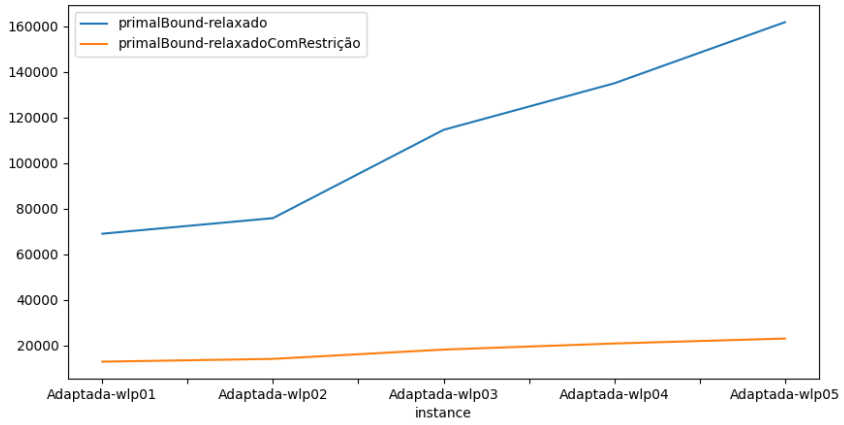
  Obs.: A tabela abaixo foi divida em 3 partes, prezando pela formatação do pdf

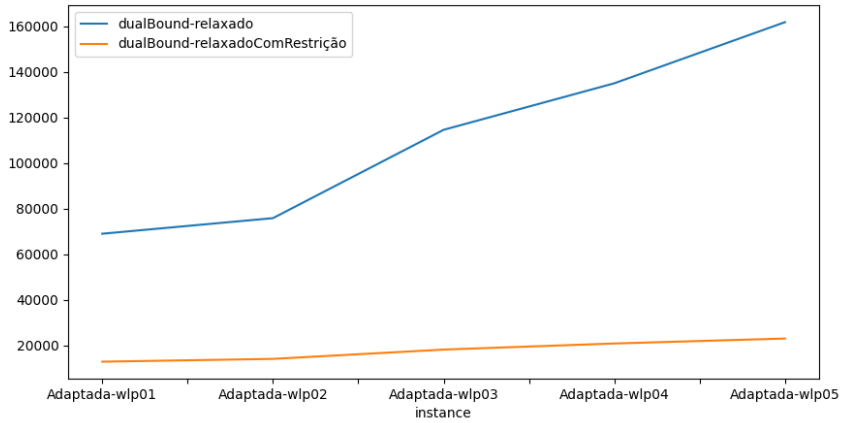
Instância	Limitante Dual SCIP	Limitante Dual Gurobi	Diferença Limitante Dual	Limitante Primal SCIP	Limitante Primal Gurobi
1	1.28E+04	1.28E+04	4.73E-11	1.28E+04	1.28E+04
2	1.40E+04	1.40E+04	2.47E-10	1.40E+04	1.40E+04
3	1.81E+04	1.81E+04	9.46E-11	1.81E+04	1.81E+04
4	TLE	2.08E+04	TLE	TLE	2.08E+04
5	TLE	2.30E+04	TLE	TLE	2.30E+04

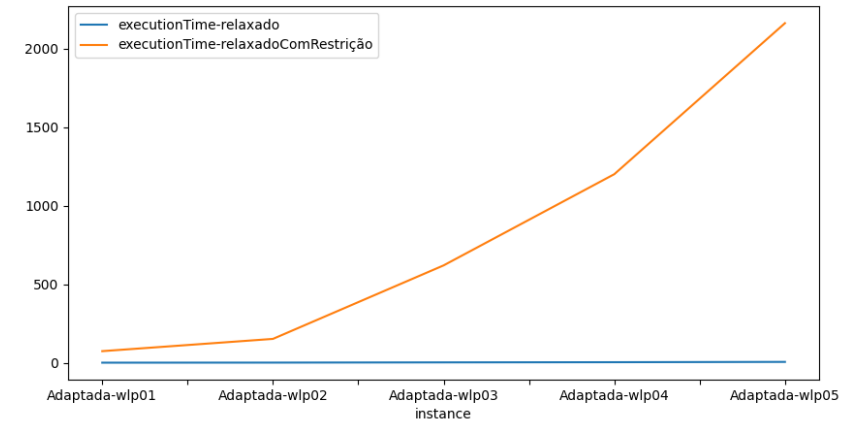
Instância	Diferença Limitante Primal	Alcançou Tempo Limite SCIP	Alcançou Tempo Limite Gurobi	Gap SCIP	Gap Gurobi
1	4.73E-11	Não	Não	0.00%	TLE
2	2.47E-10	Não	Não	0.00%	TLE
3	9.46E-11	Não	Não	0.00%	TLE
4	TLE	TLE	Não	TLE	TLE
5	TLE	TLE	Não	TLE	TLE

Instância	Tempo de Execução SCIP	Tempo de Execução Gurobi	Diferença Tempo de Execução	Otimalidade SCIP	Otimalidade Gurobi
1	00:10:10.11	00:01:14.20	00:08:55.91	Sim	Sim
2	00:23:43.64	00:02:32.19	00:21:11.45	Sim	Sim
3	01:57:24.46	00:10:19.81	01:47:04.65	Sim	Sim
4	TLE	00:20:00.04	TLE	TLE	Sim
5	TLE	00:36:00.86	TLE	TLE	Sim

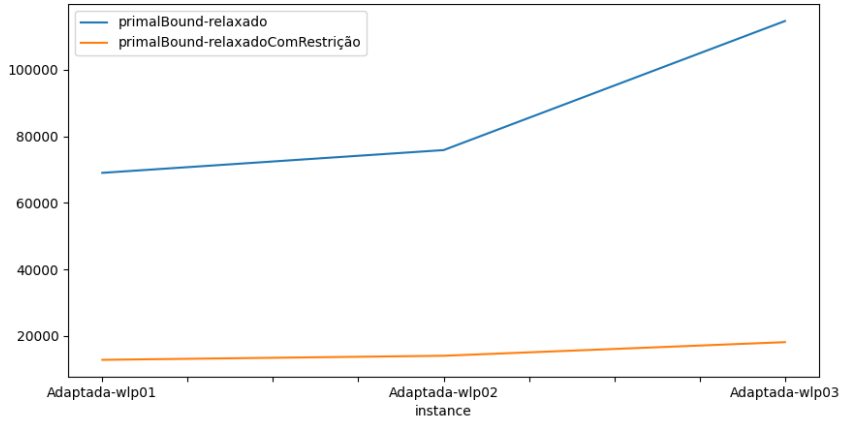
- **GUROBI Comparativo:**

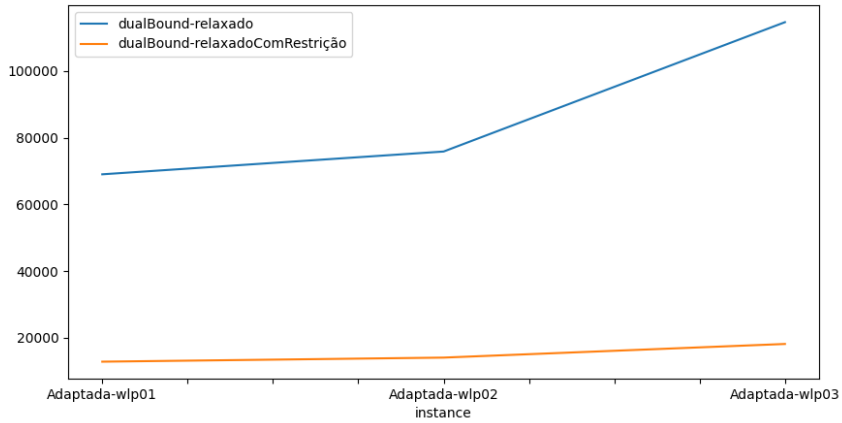


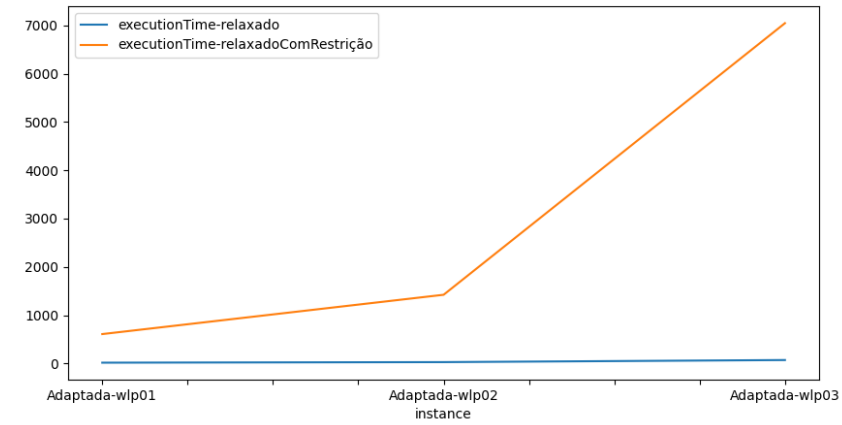




- **SCIP Comparativo:**







Análise

Os gráficos e tabelas abaixo ilustram as seguintes conclusões:

SCIP x Gurobi

- **tempo limite:** O **Gurobi** se mostrou mais eficiente em todas as instâncias, obtendo um resultado ótimo mais rápido que o **SCIP** em todos os casos
- **prova de otimalidade:** Todos os *solvers* foram capazes de obter um resultado ótimo em todas as instâncias. Vale ressaltar que o **SCIP** não foi capaz de obter resultados para as instâncias 4 e 5 em tempo hábil para a realização do trabalho (< 4h+).
- **gap:** Todas as instâncias em todos os *solvers* tiveram gap nulo

Relaxado x Relaxado com Restrições Customizadas

- **tempo limite:** O problema relaxado foi resolvido mais rapidamente que o com restrições customizadas em todas instâncias e todos *solvers*
- **prova de otimalidade:** Em todos os casos foi provado a otimalidade. Vale ressaltar que os resultados de limitante primal e dual obtidos para o problema relaxado diferem consideravelmente dos resultados obtidos para o problema com restrições customizadas

- **gap:** Todas as instâncias em todos os *solvers* tiveram gap nulo em ambos os modelos

Tarefas 3 e 4

(3) Resolva as instâncias disponibilizadas no site da disciplina utilizando o *software* não-comercial de otimização SCIP, com tempo limite de **5 minutos** para cada instância.

(4) Resolver as instâncias disponibilizadas no site da disciplina utilizando o *software* comercial de otimização GUROBI com tempo limite de **5 minutos** para cada instância.

Código

Modelos

Ambos os modelos foram apresentados na desse relatório

Resultados

SCIP (Tarefa 03) vs Gurobi (Tarefa 04)

Obs.: A tabela abaixo foi divida em 3 partes, prezando pela formatação do pdf

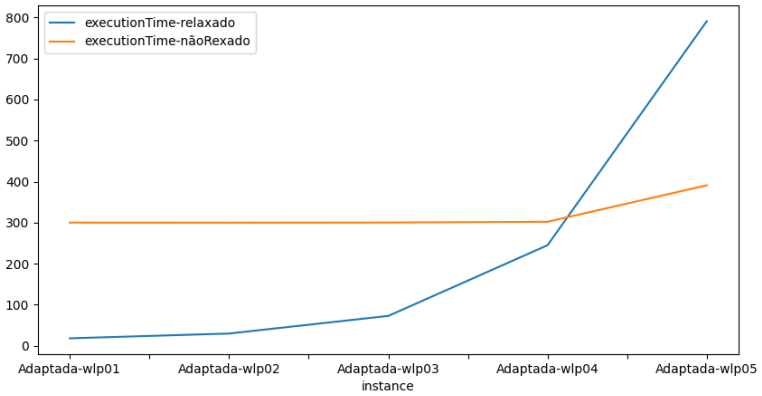
Obs.: A tabela abaixo foi divida em 3 partes, prezando pela formatação do pdf

Instância	Limitante Dual SCIP	Limitante Dual Gurobi	Diferença Limitante Dual	Limitante Primal SCIP	Limitante Primal Gurobi
1	6.90E+04	6.90E+04	6.67E+00	6.93E+04	6.91E+04
2	7.59E+04	7.59E+04	2.33E+01	7.59E+04	7.59E+04
3	1.15E+05	1.15E+05	4.93E+00	1.15E+05	1.15E+05
4	1.35E+05	1.35E+05	7.12E+00	1.41E+05	1.35E+05
5	-1.00E+20	1.62E+05	1.00E+20	1.00E+20	1.62E+05

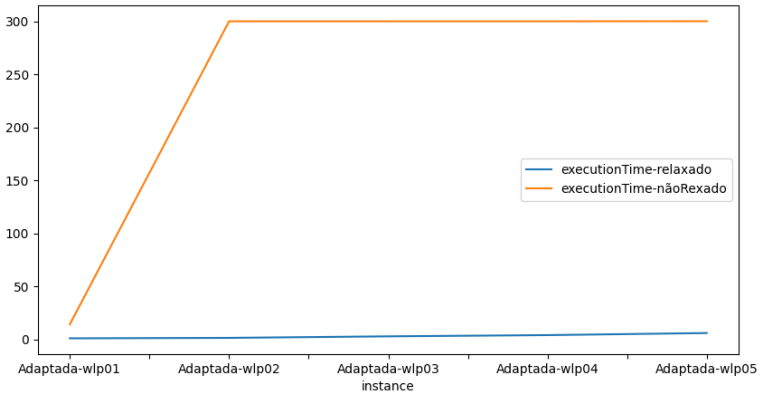
Instância	Diferença Limitante Primal	Alcançou Tempo Limite SCIP	Alcançou Tempo Limite Gurobi	Gap SCIP	Gap Gurobi	Diferença Gap
1	2.80E+02	Sim	Não	0.42%	0.01%	0.42%
2	2.91E-11	Sim	Sim	0.08%	0.05%	0.03%
3	1.57E+02	Sim	Sim	0.19%	0.05%	0.14%
4	5.70E+03	Sim	Sim	4.25%	0.03%	4.23%
5	1.00E+20	Sim	Sim	10^{22}.00%	0.04%	10^{22}.00%

Instância	Tempo de Execução SCIP	Tempo de Execução Gurobi	Diferença Tempo de Execução	Otimalidade SCIP	Otimalidade Gurobi
1	00:05:00.16	00:00:14.11	00:04:46.05	Não	Sim
2	00:05:00.05	00:05:00.03	00:00:00.02	Não	Não
3	00:05:00.33	00:05:00.04	00:00:00.30	Não	Não
4	00:05:02.14	00:05:00.05	00:00:02.09	Não	Não
5	00:06:30.95	00:05:00.11	00:01:30.84	Não	Não

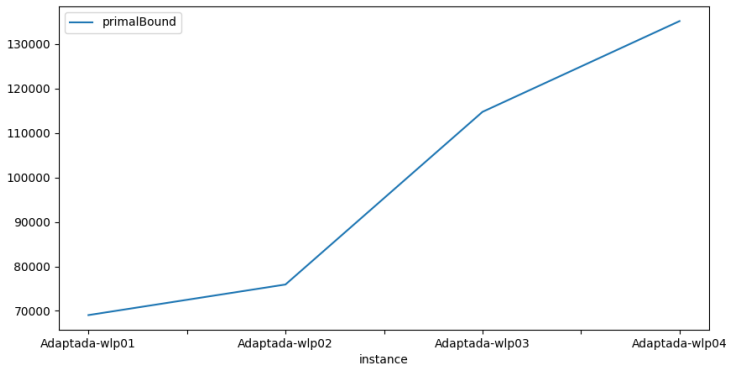
- **Comparativo sobre relaxamento**
  - **SCIP:**

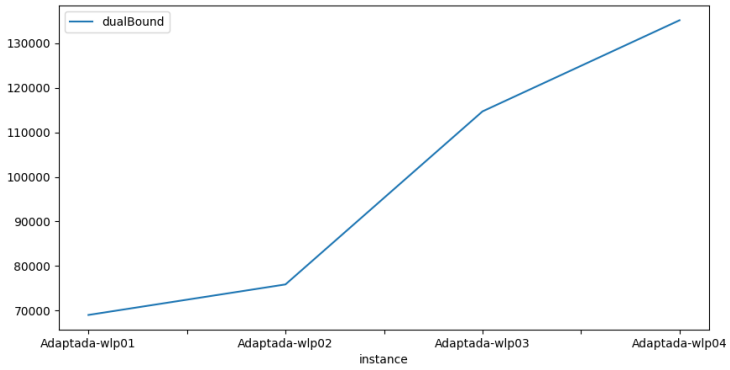


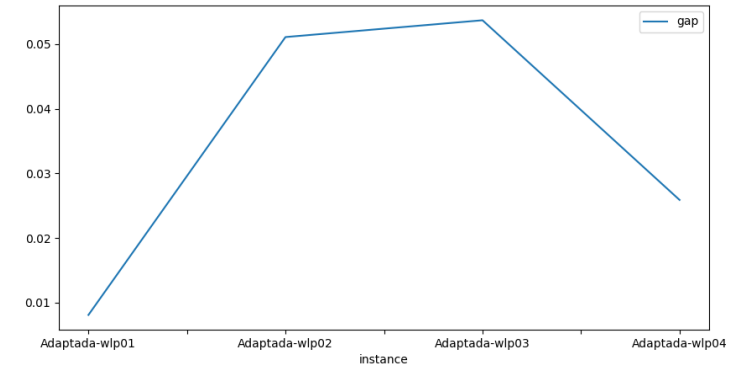
- **GORUBI:**

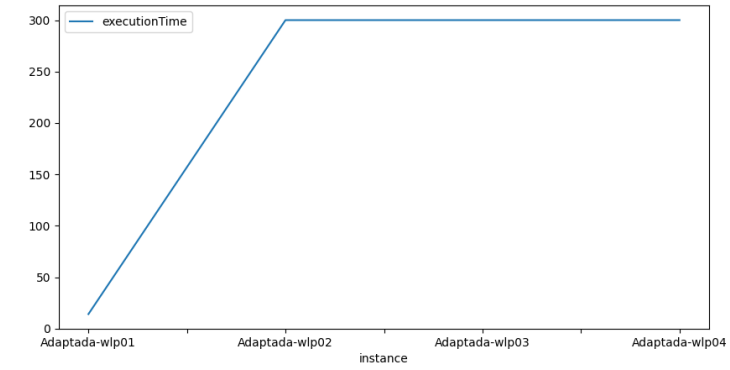


- **GUROBI:**

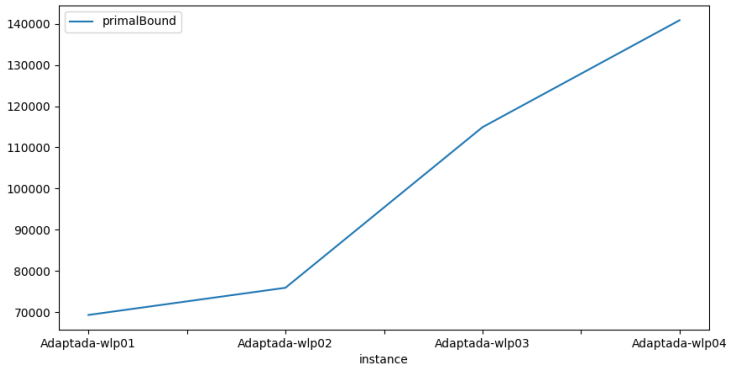


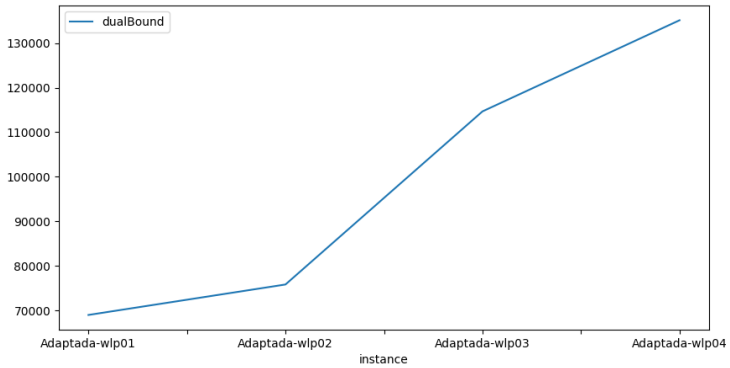

dualBound
130000
120000
110000
100000
90000
80000
70000
Adaptada-wlp01
Adaptada-wlp02
Adaptada-wlp03
Adaptada-wlp04
instance

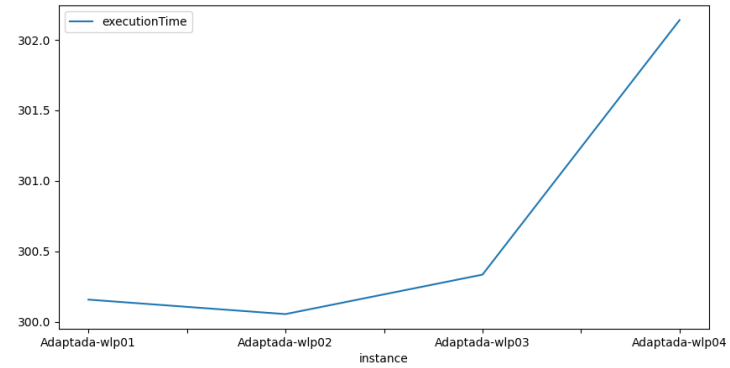

gap
0.05
0.04
0.03
0.02
0.01
Adaptada-wlp01
Adaptada-wlp02
Adaptada-wlp03
Adaptada-wlp04
instance

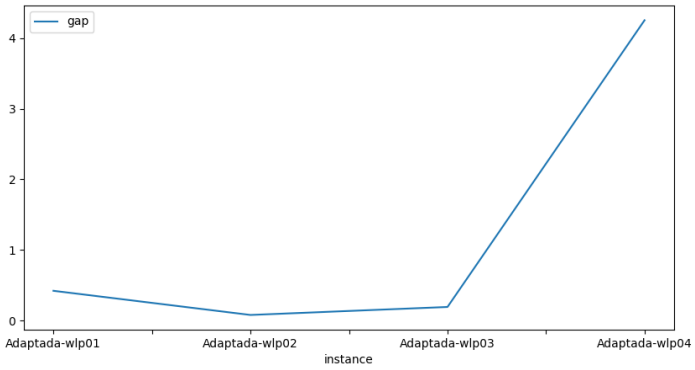

executionTime
300
250
200
150
100
50
0
Adaptada-wlp01
Adaptada-wlp02
Adaptada-wlp03
Adaptada-wlp04
instance

- **SCIP:**

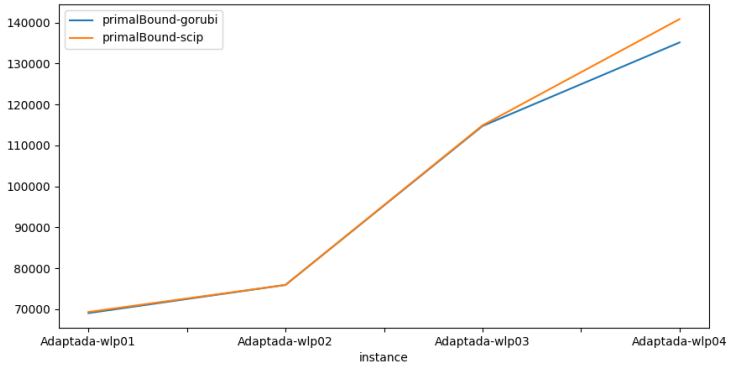

primalBound
140000
130000
120000
110000
100000
90000
80000
70000
Adaptada-wlp01
Adaptada-wlp02
Adaptada-wlp03
Adaptada-wlp04
instance

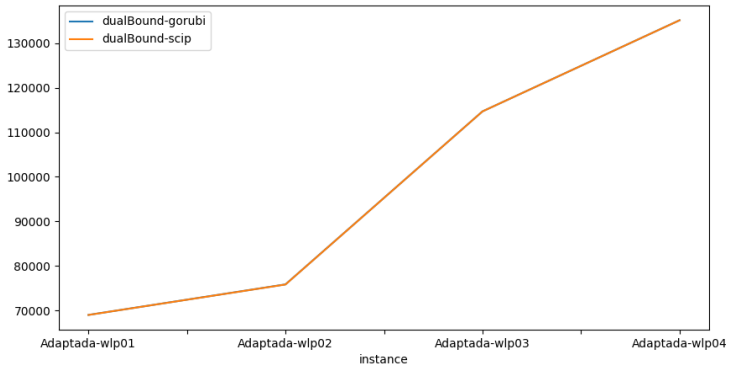

dualBound
130000
120000
110000
100000
90000
80000
70000
Adaptada-wlp01
Adaptada-wlp02
Adaptada-wlp03
Adaptada-wlp04
instance

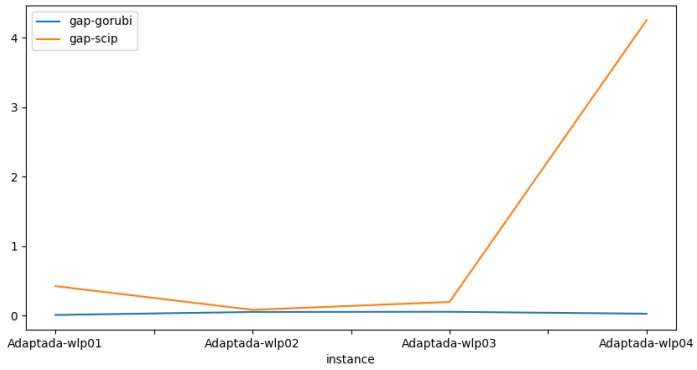

executionTime
302.0
301.5
301.0
300.5
300.0
Adaptada-wlp01
Adaptada-wlp02
Adaptada-wlp03
Adaptada-wlp04
instance

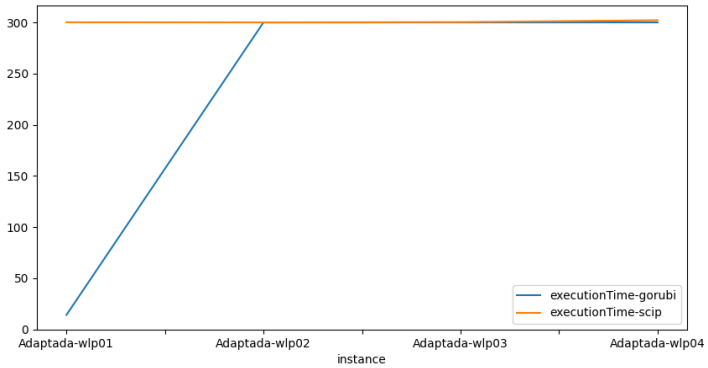


- **GORUBI X SCIP:**









Análise

Os gráficos e tabelas abaixo ilustram as seguintes conclusões:

- **tempo limite:** Ambos os *solvers* atingem o tempo limite de execução, salvo pela instância 1, a qual pôde ser resolvida em tempo recorde pelo **Gurobi**.
- **prova de otimalidade:** A partir da instância 1, a qual foi provada otimalidade, o **Gurobi** estourou o tempo limite de 5 minutos, impedindo a prova de otimalidade. Já o **SCIP**, desde a primeira instância o tempo limite definido não foi o suficiente para provar a otimalidade.
- **gap:** Apesar de estourarem o tempo limite nas instâncias subsequentes, o **Gurobi** se mostrou ser mais promissor com gap relativamente menor que o **SCIP**. Por fim, na instância 5 o **SCIP** foi incapaz de selecionar um bom valor inicial no tempo estipulado, o que fez com o que o gap obtido fosse exorbitante.

Tarefa 5 - Aplicação do Problema de Facilidades

Pesquise uma aplicação do problema de localização de facilidades. Descreva a aplicação, explique quais são os parâmetros, suas variáveis, a função objetivo e suas restrições. Em resumo, deixe clara sua aplicação e justifique sua utilidade. Obs. Não podem ser utilizados os exemplos apresentados em aula.

Ideia Geral do Problema

Imaginemos uma situação em que uma empresa de logística precisa decidir onde construir centros de distribuição para minimizar o custo operacional e maximizar a cobertura de clientes.

Foram identificados vários locais candidatos para os centros de distribuição, mas devem ser tomadas decisões com relação a quantos armazéns abrir e em quais localidades.

Abrir muitos armazéns seria vantajoso, pois aumentaria a cobertura de clientes. No entanto, a abertura de um centro de distribuição tem um custo fixo associado e o transporte de mercadorias até o cliente também tem um custo variável de acordo com o centro de distribuição e o cliente.

Neste exemplo, nosso objetivo é encontrar o equilíbrio ideal entre a cobertura oferecida pelos centros de distribuições abertos e os custos de construção e operação dos mesmos.

Como temos mais de um objetivo neste problema (maximizar cobertura e minimizar custos) utilizaremos a estratégia de Agregação Ponderada, na função objetivo, de modo a atribuir diferentes pesos a cada objetivo. Caso seja desejado priorizar um objetivo a outro, basta atribuir-se um peso maior ao objetivo principal.

Formulação do Problema

Conjuntos e Índices

$i \in I$: Índice e conjunto dos centros de distribuição (ou facilidades).

$j \in J$: Índice e conjunto dos clientes.

m: Número de clientes

Parâmetros

$f_i \in \mathbb{R}^+$: Custo fixo associado a construção do centro de distribuição $i \in I$.

$d_j \in \mathbb{R}^+$: Demanda máxima do cliente $j \in J$.

$Cap_i \in \mathbb{R}^+$: Capacidade do centro de distribuição $i \in I$.

$c_{i,j} \in \mathbb{R}^+$: Custo de transporte entre o centro de distribuição candidato $i \in I$ e a localidade do cliente $j \in J$.

$w_{custo} \in \mathbb{R}^+$: Peso do objetivo de minimizar os custos totais.

$w_{cobertura} \in \mathbb{R}^+$: Peso do objetivo de maximizar cobertura total.

Variáveis de decisão

$select_i \in \{0, 1\}$: Variável é 1 se o centro de distribuição é construído na localidade candidata $i \in I$; e 0 caso contrário.

$0 \leq assign_{i,j} \leq 1$: Variável contínua, não negativa que determina a fração de cobertura do cliente $j \in J$ pelo centro de distribuição $i \in I$.

Função Objetivo

- **Custos Totais e Cobertura Total:** Queremos minimizar os custos totais de abrir e operar os centros de distribuição e ao mesmo tempo maximizar a cobertura de clientes. Para isso utilizaremos uma função objetivo agregada ponderada.
  - Para os custos totais, temos a soma do custo fixo de abrir centros de distribuição e o custo relacionado ao transporte entre centros de distribuição e clientes dividida pela soma de todos custos fixos e variáveis (normalizada).
  - Para a cobertura total temos a soma das coberturas de cada cliente por cada centro de distribuição dividida pelo número de clientes (normalizada). Como o objetivo é aumentar a cobertura e a função objetivo é de minimização, utiliza-se o oposto do valor encontrado.
  - Por fim calcula-se a soma ponderada dos objetivos com seus respectivos pesos dividida pela soma total dos pesos.

$$\text{Min} \quad Z = \left(\frac{\sum_{i \in I} f_i \cdot select_i + \sum_{i \in I} \sum_{j \in J} c_{i,j} \cdot assign_{i,j}}{\sum_{i \in I} f_i + \sum_{i \in I} \sum_{j \in J} c_{i,j}}\right) \cdot w_{custo} + \left(-\frac{\sum_{i \in I} \sum_{j \in J} assign_{i,j}}{m}\right) \cdot w_{cobertura} \quad (0)$$

Restrições

- **Demanda:** Para cada cliente $i \in I$ assegura-se que sua demanda máxima é respeitada. Sendo assim, a soma da fração recebida de cada facilidade para cada cliente deve ser menor ou igual a 1.

$$\sum_{i \in I} assign_{i,j} \leq 1 \quad \forall j \in J \tag{1}$$

- **Capacidade:** É preciso garantir que a facilidade $i \in I$, lide no máximo com sua capacidade máxima, caso ela tenha sido construída.

$$\sum_{j \in J} d_j \cdot assign_{i,j} \leq Cap_i \cdot select_i \quad \forall i \in I \tag{2}$$

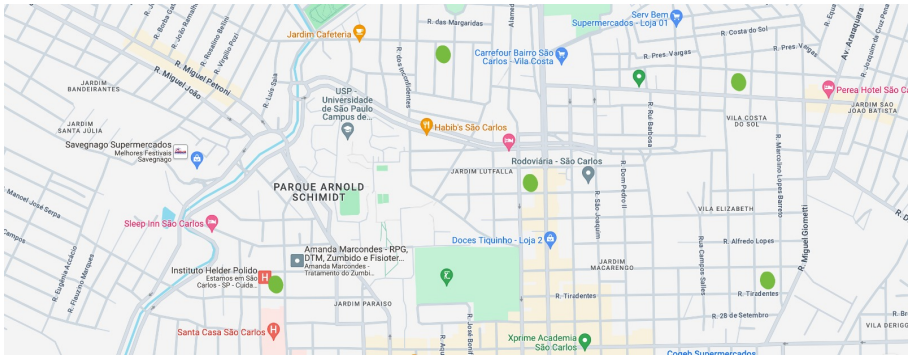
Tarefa 6 - *Toy Problem*

Elabore um problema exemplo (*toy problem*), descreva-o matematicamente utilizando o modelo e resolva-o utilizando o *solver* não-comercial **SCIP**.
Este exemplo deve ser pequeno em relação a sua dimensão. Ele deve ser utilizado para facilitar a compreensão do problema, do modelo e de suas restrições. Ele deve ser um exemplo para a aplicação escolhida na Tarefa 5.

Exemplificação do Problema de Facilidades

A campanha do agasalho em São Carlos, possui o seguinte lema: "*Doe agasalhos: aqueça a noite e o coração de alguém*". Para que a campanha seja um sucesso de arrecadações, é preciso pensar nas seguintes estratégias: onde instalar um ponto de coleta na cidade? Quais são as localidades candidatas? Quais os custos de transporte dos pontos de coleta até as instituições que irão receber essas peças de roupa? Qual o cuso fixo de construção de um ponto de coleta? Como aumentar a cobertura da demanda das instituições que irão receber essas peças de roupa?

Para reduzir o conjunto das possíveis localidades, a prefeitura de São Carlos nos mostrou um mapa com alguns locais estratégicos para construção dos pontos de coleta:



As localidadades candidatas podem ser identificadas pelos pontos verdes no mapa.

Para arrecadar ainda mais agasalhos durante a campanha, a prefeitura pensou em construir pontos de coletas utilizando contâiners metálicos para preservar as peças de roupa da chuva e poeira, além de permitir que as pessoas apenas depositem as roupas, impedindo que alguém sem a chave as retirem do ponto de arrecação. Dessa forma, cada ponto de coleta terá um custo fixo associado na sua construção e manutenção.

Além disso, a prefeitura deseja enviar os agasalhos arrecadados não apenas para as instituições de São Carlos, mas também para as cidades vizinhas como: Água vermelha, Ibaté, Descalvado, Riberão Bonito, etc. Essas instituições também possuem as devidas demandas de agasalho conforme as necessidades da sua comunidade.

Portanto, é preciso minimizar os custos de construir os contâiners metálicos de arrecadação e os custos de transporte das peças de roupas até as instituições dentro e fora de São Carlos e ao mesmo tempo maximizar a cobertura da demanda das instituições participantes pelos pontos de coleta construídos.

Formulação do Problema

Conjuntos e Índices

$i \in I$: Índice e conjunto das localidades candidatas dos pontos de coleta.

$j \in J$: Índice e conjunto das localidades das instituições participantes.

m: Número de instituições participantes

Parâmetros

$f_i \in \mathbb{R}^+$: Custo fixo associado a construção dos pontos de coleta $i \in I$.

$d_j \in \mathbb{R}^+$: Demanda máxima da instituição participante $j \in J$.

$Cap_i \in \mathbb{R}^+$: Capacidade do ponto de coleta $i \in I$.

$c_{i,j} \in \mathbb{R}^+$: Custo de transporte entre o ponto de coleta candidato $i \in I$ e a localidade da instituição $j \in J$.

$w_{custo} \in \mathbb{R}^+$: Peso do objetivo de minimizar os custos totais.

$w_{cobertura} \in \mathbb{R}^+$: Peso do objetivo de maximizar cobertura total.

Variáveis

$select_i \in \{0, 1\}$: Variável é 1 se o ponto de coleta é construído na localidade candidata i I; e 0 caso contrário.

$0 \leq assign_{i,j} \leq 1$: Variável contínua, não negativa que determina a fração da demanda de peças de roupa recebida pela instituição $j \in J$ do ponto de coleta$i \in I$.

Restrições

- **Demanda:** Para cada instituição participante $j \in J$ assegura-se que sua demanda máxima é respeitada. Sendo assim, a soma da fração recebida de cada ponto de coleta para cada instituição deve ser menor ou igual a 1:

$$\sum_{i \in I} assign_{i,j} \leq 1 \quad \forall j \in J \tag{1}$$

- **Capacidade:** É preciso garantir que o ponto de coleta $i \in I$, lide no máximo com sua capacidade máxima, caso ele tenha sido construída.

$$\sum_{j \in J} d_j \cdot assign_{i,j} \leq Cap_i \cdot select_i \quad \forall i \in I \tag{2}$$

Função Objetivo

- **Custos Totais e Cobertura Total:** Queremos minimizar os custos totais de abrir e operar os pontos de coleta e ao mesmo tempo maximizar a cobertura da demanda das instituições participantes. Para isso utilizaremos uma função objetivo agregada ponderada.
  - Para os custos totais, temos a soma do custo fixo de abrir pontos de coleta e o custo relacionado ao transporte entre pontos de coleta e as instituições participantes dividida pela soma de todos custos fixos e variáveis (normalizada).
  - Para a cobertura total temos a soma das coberturas de cada instituição participante por cada ponto de coleta dividida pelo número de instituições participantes (normalizada). Como o objetivo é aumentar a cobertura e a função objetivo é de minimização, utiliza-se o oposto do valor encontrado.
  - Por fim calcula-se a soma ponderada dos objetivos com seus respectivos pesos dividida pela soma total dos pesos.

$$\text{Min} \quad Z = \left(\frac{\sum_{i \in I} f_i \cdot select_i + \sum_{i \in I} \sum_{j \in J} c_{i,j} \cdot assign_{i,j}}{\sum_{i \in I} f_i + \sum_{i \in I} \sum_{j \in J} c_{i,j}}\right) \cdot w_{custo} \left(-\frac{\sum_{i \in I} \sum_{j \in J} assign_{i,j}}{m}\right) \cdot w_{cobertura} \tag{0}$$

Dados

- **Dimensões:** (5, 4)

- **Capacidades:**

Ponto 1	Ponto 2	Ponto 3	Ponto 4	Ponto 5
20	22	17	19	18

- **Custos Fixados:**

Ponto 1	Ponto 2	Ponto 3	Ponto 4	Ponto 5
12000	15000	17000	13000	2200

- **Demandas:**

Instituição 1	Instituição 2	Instituição 3	Instituição 4
15	18	14	20

- **Custos de Transport:**

	Instituição 1	Instituição 2	Instituição 3	Instituição 4
Ponto 1	4000	2500	1200	2200
Ponto 2	2000	2600	1800	2600
Ponto 3	3000	3400	2600	3100
Ponto 4	2500	3000	4100	3700
Ponto 5	4500	4000	3000	3200

Modelo SCIP

```
# n ->  número de locais onde pontos de coleta podem ser instalados
# m -> número de instituições participantes que devem ser atendidas
# capacities -> capacidade de cada ponto de coleta
# demands -> demanda de cada instituição participante
# fixedCosts -> custo fixo de cada ponto de coleta
# transportCosts -> custo de transporte de cada ponto de coleta para cada instituição participante

def getToyProblemModelSCIP(data, wCost = 1, wCoverage = 1):
    n, m = data['dimensions']
    capacities = data['capacities']
    demands = data['demands']
    fixedCosts = data['fixedCosts']
    transportCosts = data['transportCosts']

    model = scip.Model()

    # Definindo variáveis para o modelo
    # Pontos de coleta abertos
    select = {}
    for i in range(n):
        select[i] = model.addVar(vtype='BINARY', obj=fixedCosts[i], name=f"select_{i}")

    # Fração da demanda da instituição participante j atendida pelo ponto de coleta i
    assign = {}
    for i in range(n):
        for j in range(m):
            assign[i, j] = model.addVar(vtype='CONTINUOUS', lb=0, ub=1, name=f'coveredClients_{i}_{j}')

    # Definindo as restrições do problema
    # Garantindo que todos as instituições participantes tenham sua demanda máxima respeitada (1)
    for j in range(m):
        model.addCons(scip.quicksum(assign[(i, j)] for i in range(n)) <= 1, name=f'clientDemand_{j}')

    # Garantindo que a demanda total atendida pelos pontos de coleta abertos seja menor ou igual à capacidade desses pontos de coleta (2)
    for i in range(n):
        model.addCons(scip.quicksum(demands[j] * assign[i, j] for j in range(m)) <= capacities[i] * select[i], name=f'capacityConstr_{i}')


    # Definindo função objetivo (0)
```

```
    sumOfCosts = (sum(fixedCosts[i] for i in range(n))
        +sum(transportCosts[i][j] for i in range(n) for j in range(m)))

    totalCost = (scip.quicksum(fixedCosts[i] * select[i] for i in range(n))
        + scip.quicksum(transportCosts[i][j] * assign[i, j] for i in range(n) for j in range(m))) / sumOfCosts

    totalCoverage = scip.quicksum(assign[i, j] for i in range(n) for j in range(m)) / m

    model.setObjective(
        totalCost * wCost - totalCoverage * wCoverage,
        "minimize"
    )

    model.data = {
        'select': select,
        'assign': assign

    }

    return model
```

Resultados

SCIP

Instância	Limitante Dual	Limitante Primal	Tempo Limite	Gap	Tempo de execução	Otimalidade
Toy Problem	0.5619	-0.5619	Não	0.0%	0.0024	Sim

Análise

O *solver* **SCIP** é capaz de encontrar um resultado ótimo, com *gap* de 0%, rapidamente. A partir do modelo otimizado é possível obter então os valores ideais das variáveis *assign* e *select* e desta forma descobrir quais pontos de coleta abrir ou não e a porcentagem de cobertura das demandas das instituições participantes que deve ser atendida por cada ponto de coleta de modo a economizar nos gastos e maximizar a cobertura total.

Como executar o trabalho (README)

Junto a esse relatório será disponibilizado um arquivo Notebook Python (*.ipynb*). Basta fazer *upload* do mesmo para a plataforma Google Colab e executar todas as células.

Para fazer *download* dos resultados obtidos, basta acessar o repositório git em: https://github.com/rolimans/sme0110-t1.