Modelagem e Otimização de Fluxo Máximo:

Transformação de PFM em PFCM

Matheus Batista Honório, Victória Monteiro Pontes e Lucas Guedes da Silva

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2020

Matheus Batista Honório, Victória Monteiro Pontes e Lucas Guedes da Silva

Modelagem e Otimização de Fluxo Máximo

Relatório apresentado à disciplina Pesquisa Operacional do curso Engenharia de Computação do Centro de Informática, da Universidade Federal da Paraíba.

Professor: Teobaldo Leite Bulhões Junior

Julho de 2020

AGRADECIMENTOS

Agradecimento à Itamar com seu github sobre Pesquisa Operacional, muito organizado.

RESUMO

Existem vários modelos de Problema do Fluxo Máximo e formas de modelá-los com o intuito de passá-los para um solver e então resolvê-los. O projeto consiste em modelar um Problema de Fluxo Máximo como um Problema de Fluxo de Custo Mínimo que é um subtópico de modelagem em redes. Seguindo os passos que mostraremos a seguir, é possível modelar um PFM em PFCM e achar o seu valor ótimo. O trabalho também contará com a resolução do exercício 9.4-3 do livro [1].

Palavras-chave: <Fluxo>, <Custo>, <Mínimo>, <Máximo>.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

UFPB	Universidade Federal da Paraíba
PO	Pesquisa Operacional
PFM	Problema do Fluxo Máximo
PFCM	Problema do Fluxo de Custo Mínimo

Sumário

1 INTRODUÇÃO

A Pesquisa Operacional é uma ciência aplicada voltada para a resolução de problemas reais. Tendo como foco a tomada de decisões, aplica conceitos e métodos de várias áreas científicas na concepção, planejamento ou operação de sistemas.

A Pesquisa Operacional é usada para avaliar linhas de ação alternativas e encontrar as soluções que melhor servem aos objetivos dos indivíduos ou organizações.

Um modelo de programação matemática é definido por um sistema de equações/inequações.

- As **variáveis** representam as decisões a serem tomadas;
- As **equações/inequações** representam as restrições que existem sobre essas decisões, refletindo as características do sistema real.
- Uma **função objetivo** indica qual dentre as possíveis decisões é a mais desejável (solução ótima).

Segundo [1], O Problema de Fluxo Máximo que consiste em determinar o valor do maior fluxo possível que pode ser enviado de um nó a outro da rede.

1.1 Definição do Problema

O projeto é constituído de três partes: introdução, modelagem de PFM para PFCM e o exercício 9.4-3 da página 395 do livro Introdução à Pesquisa Operacional, 9º edição.

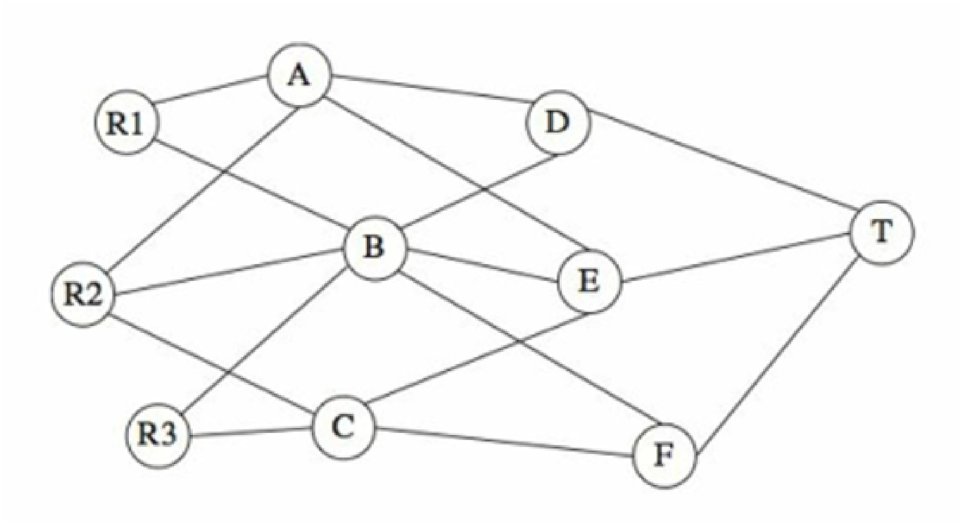

Figura 1: *Grafo que representa o sistema de arquedutos do Exercício 9.4-3*

Com objetivo de realizar um plano de fluxo que vai maximizar o fluxo de agua para a cidade.

De / Para	A	B	C
R1	130	115	–
R2	70	90	110
R3	–	140	120

De / Para	D	E	F
A	110	85	–
B	130	95	85
C	–	130	160

De / Para	T
D	220
E	330
F	240

Figura 2: *Tabela que representa a entrada dos arcos do sistema de arquedutos do Exercício 9.4-3*

2 METODOLOGIA

A modelagem foi feita primeiramente de forma manual, utilizando-se da linguagem matemática para a sua conclusão. Após a modelagem manual, a linguagem matemática foi transcrita para a linguagem de programação Python, utilizando o CPLEX e a sua biblioteca, tanto para a transcrição quanto para a resolução dos problemas propostos.

Para a transformação do PFM em PFCM foi utilizado o seguinte método:

- Deve ser atribuída a todos os nós, sejam de transbordo, origem ou escoadouro, demanda de valor zero;
- O custo da passagem de fluxo para cada arco do grafo deve ser zero;
- Um novo arco do escoadouro para o nó de origem deve ser criado;
- Este novo arco deve ter custo unitário de (-1) e capacidade infinita;

Esse método garante que a quantidade de fluxo que passa pelo arco (escoadouro, origem) seja o máximo de fluxo viável possível e portanto, a solução do problema. Isso ocorre porque o custo unitário de passar por esse arco é negativo, enquanto os outros arcos possuem custos iguais a zero. Como a função objetivo procura minimizar os custos, o arco (escoadouro, origem) se torna a opção mais favorável. Devido as restrições de conservação de fluxo no arco (já que a demanda em cada nó é zero), a quantidade de fluxo que passa pelo arco (escoadouro, origem) é igual a soma das quantidades de fluxo que passam por todos os outros arcos. Dessa forma, a função objetivo tenta maximizar o fluxo dos arcos que estão com custo zero para que possa passar o máximo de fluxo possível pelo arco (escoadouro, origem) que possui custo unitário (-1). Logo, o fluxo máximo que pode ir da origem ao escoadouro é a mesma quantidade de fluxo que passa pelo arco (escoadouro, origem).

3 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

Os códigos dispõem de uma interface básica produzida em python pela biblioteca Tkinter para a seleção e compilação das instâncias.

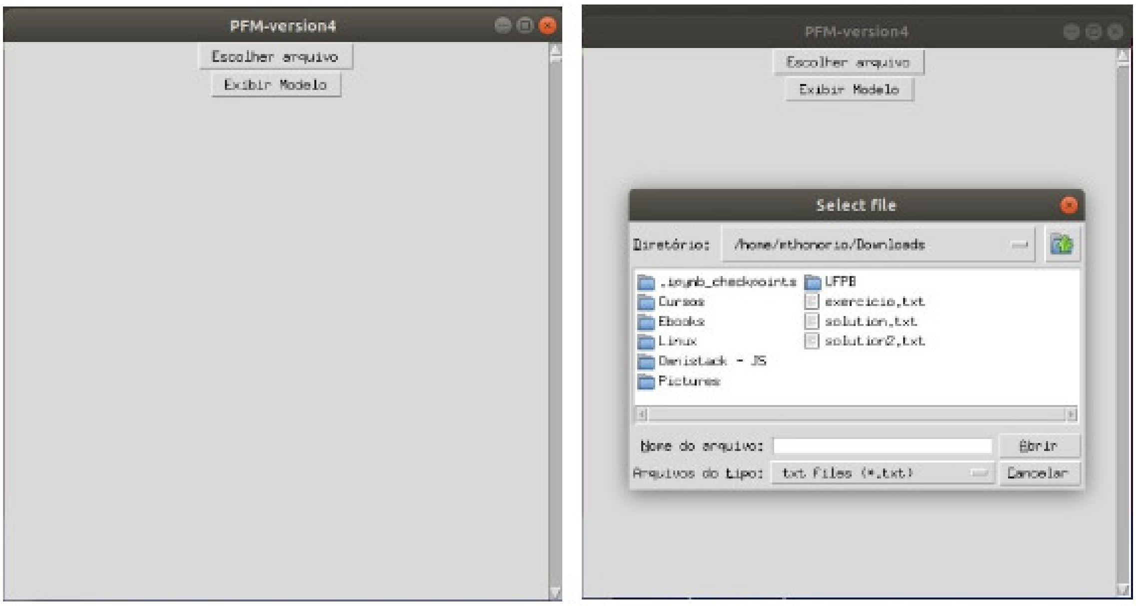

Figura 3: *Interface de inicio das Modelagens em Python*

3.1 Problema de Fluxo de Custo Mínimo

3.1.1 Fórmula geral de um PFCM

Min

$$\sum_{(i,j)\in A} c_{ij} x_{ij}$$

S.a

$$\sum_{(j,i)\in A} x_{ji} - \sum_{(i,j)\in A} x_{ij} = b_i \qquad \forall i \in N$$

$$0 \leqslant x_{ij} \leqslant u_{ij} \qquad \forall (i,j) \in A$$

3.1.2 Transformando um problema de fluxo máximo para um problema de fluxo de custo mínimo

```
#Leitura do arquivo e armazenando os dados
import pylab as pb
import sys
```

```python
from tkinter import *
from tkinter import filedialog
import os
import io
import sys

def showModel():
    result = []
    if(root.filename != ''):
        #Armazenando os dados do numero de vertices, arcos e indices dos nos de origem e escoadouro
        instance_info = pb.loadtxt(root.filename, dtype = int, max_rows = 4)

        #Armazenando os dados dos arcos
        dataSet = pb.loadtxt(root.filename,dtype = int, skiprows = 4)

        #Criando um range para facilitar as operacoes com vertices
        vertices=[]
        verticesRange = range(instance_info[0])
        for i in verticesRange:
            vertices.insert(i, i + 1)
        print(instance_info)
        result.extend((str(instance_info).split('\n')))
        print(dataSet)
        result.extend((str(dataSet).split('\n')))
        print(vertices)
        result.extend((str(vertices).split('\n')))

        #Modelagem
        from docplex.mp.model import Model
        mdl = Model("Problema do Fluxo de Custo Minimo")
        #Criando ranges
        dataRange = range(instance_info[0] + 1)
        source = vertices[instance_info[2]] #No de origem
        sink = vertices[instance_info[3]] #Escoadouro

        edge_capacity = {}
        costUnit = {}
        for i in range(len(dataSet)):
            edge_capacity[(dataSet[i][0], dataSet[i][1])] = dataSet[i][2]
```

```
            costUnit[(dataSet[i][0], dataSet[i][1])] = 0

        #Agora adicionamos um novo arco (origem, escoadouro) que
           possui um custo maior ao dicionario
        #Tambem adicionaremos o novo arco ao edge_capacity com uma
           capacidade "infinita", que na verdade o apenas um numero
           muito grande

        smallCost = -1
        bigNumber = sys.float_info.max
        costUnit[(sink, source)] = smallCost
        edge_capacity[(sink, source)] = bigNumber

        #Criamos um dicionario demanda, que relaciona cada no a sua
           demanda(bi = 0)
        demand = {}
        for i in vertices:
            demand[i] = 0

        print(demand)
        result.extend((str(demand).split('\n')))

        #Modelagem
        #Criando variaveis Xij para cada arco (i, j)
        x = {(i, j): mdl.continuous_var(name="x_{0}_{1}".format(i, j))
            for i in vertices for j in vertices if (i, j) in
           edge_capacity}
        mdl.print_information()
        print(x)

        result.extend((str(x).split('\n')))

        #Adicionando restricoes de conservacao de fluxo (no pfcm,
           restricoes que envolvem demanda)
        for i in vertices:
            mdl.add_constraint(mdl.sum(x[j, i] for j in vertices if (j
               , i) in edge_capacity) -
                                mdl.sum(x[i, j] for j in vertices if (i, j
                                   ) in edge_capacity) == demand[i])

        minimumFlow = {}
        for i in vertices:
```

```
            for j in vertices:
                if (i, j) in edge_capacity:
                    minimumFlow[i, j] = 0
        print(minimumFlow)

        #Adicionando restricoes de fluxo maximo e fluxo minimo
        for i in vertices:
            for j in vertices:
                if (i, j) in edge_capacity:
                    mdl.add_constraint(x[i, j] >= minimumFlow[i, j])
                    mdl.add_constraint(x[i, j] <= edge_capacity[i, j])

        #Criando a funcao objetivo
        mdl.minimize(mdl.sum(costUnit[i, j] * x[i, j] for i in
           vertices for j in vertices if (i, j) in costUnit))

        print(mdl.export_to_string())

        result.extend((str(mdl.export_to_string()).split('\n')))

        solution = mdl.solve(log_output=True)

        old_target = sys.stdout
        sys.stdout = open('solution2.txt', 'w')

        solution.display()

        sys.stdout.close()
        sys.stdout = old_target

        r = open('solution2.txt','r')

        contentSolution = str(r.read()).split('\n')


        contentSolutionString= '\n'.join(contentSolution)

        print(contentSolutionString)



        result.extend(contentSolution)
```

```
        mylist = Listbox(root, yscrollcommand = scrollbar.set,width
           =150,height=200)

        for line in result:
            mylist.insert(END, line)

        mylist.pack()

def chooseFileName():
    root.filename =  filedialog.askopenfilename(initialdir =
       currentDirectory,title = "Select file",filetypes = (("txt files
       ","*.txt"),("all files","*.*")))
    print (root.filename)



root = Tk()

root.title('PfmToPfcm-version2')

scrollbar = Scrollbar(root)
scrollbar.pack( side = RIGHT, fill = Y )
root.filename = ''


root.geometry("500x500") #You want the size of the app to be 500x500
#root.resizable(0, 0) #Don't allow resizing in the x or y direction


chooseButton = Button(root,text="Escolher arquivo",command =
   chooseFileName)
solveButton = Button(root,text="Exibir Modelo",command = showModel)




chooseButton.pack()
solveButton.pack()


currentDirectory = os.getcwd()
```

```


root.mainloop()
```

3.1.3 Resultados:

Printando os vértices:

```
[11 34  0 10]
[[      1       2     245]
 [      1       3     270]
 [      1       4     260]
 [      2       5     130]
 [      2       6     115]
 [      3       5      70]
 [      3       6      90]
 [      3       7     110]
 [      4       6     140]
 [      4       7     120]
 [      5       8     110]
 [      5       9      85]
 [      6       8     130]
 [      6       9      95]
 [      6      10      85]
 [      7       9     130]
 [      7      10     160]
 [      8      11     220]
 [      9      11     330]
 [     10      11     240]
 [      5       2     130]
 [      5       3      70]
 [      6       2     115]
 [      6       3      90]
 [      6       4     140]
 [      7       3     110]
 [      7       4     120]
 [      8       5     100]
 [      8       6     130]
 [      9       5     330]
 [      9       6      95]
 [      9       7     130]
 [     10       6      85]
```

```
 [      10        7      160]]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Printando os arcos:

1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0

Printando o Modelo:

Model: Problema do Fluxo de Custo Minimo - number of variables: 35 - binary=0, integer=0, continuous=35 - number of constraints: 0 - linear=0 - parameters: defaults - objective: none - problem type is: LP (1, 2): docplex.mp.Var(type=C,name='x1$_2'$), $(1,3) : docplex.mp.Var(type = C, name =' x1'_3), (1,4) : docplex.mp.Var(type = C, name =' x1'_4), (2,5) : docplex.mp.Var(type = C, name =' x2'_5), (2,6) : docplex.mp.Var(type = C, name =' x2'_6), (3,5) : docplex.mp.Var(type = C, name =' x3'_5), (3,6) : docplex.mp.Var(type = C, name =' x3'_6), (3,7) : docplex.mp.Var(type = C, name =' x3'_7), (4,6) : docplex.mp.Var(type = C, name =' x4'_6), (4,7) : docplex.mp.Var(type = C, name =' x4'_7), (5,2) : docplex.mp.Var(type = C, name =' x5'_2), (5,3) : docplex.mp.Var(type = C, name =' x5'_3), (5,8) : docplex.mp.Var(type = C, name =' x5'_8), (5,9) : docplex.mp.Var(type = C, name =' x5'_9), (6,2) : docplex.mp.Var(type = C, name =' x6'_2), (6,3) : docplex.mp.Var(type = C, name =' x6'_3), (6,4) : docplex.mp.Var(type = C, name =' x6'_4), (6,8) : docplex.mp.Var(type = C, name =' x6'_8), (6,9) : docplex.mp.Var(type = C, name =' x6'_9), (6,10) : docplex.mp.Var(type = C, name =' x6_10'), (7,3) : docplex.mp.Var(type = C, name =' x7'_3), (7,4) : docplex.mp.Var(type = C, name =' x7'_4), (7,9) : docplex.mp.Var(type = C, name =' x7'_9), (7,10) : docplex.mp.Var(type = C, name =' x7_10'), (8,5) : docplex.mp.Var(type = C, name =' x8'_5), (8,6) : docplex.mp.Var(type = C, name =' x8'_6), (8,11) : docplex.mp.Var(type = C, name =' x8_11'), (9,5) : docplex.mp.Var(type = C, name =' x9'_5), (9,6) : docplex.mp.Var(type = C, name =' x9'_6), (9,7) : docplex.mp.Var(type = C, name =' x9'_7), (9,11) : docplex.mp.Var(type = C, name =' x9_11'), (10,6) : docplex.mp.Var(type = C, name =' x_10'_6), (10,7) : docplex.mp.Var(type = C, name =' x_10'_7), (10,11) : docplex.mp.Var(type = C, name =' x_10_11'), (11,1) : docplex.mp.Var(type = C, name =' x_11'_1)$

Printando as demandas dos arcos:

```
{(1, 2): 0, (1, 3): 0, (1, 4): 0, (2, 5): 0, (2, 6): 0, (3, 5): 0, (3,
   6): 0, (3, 7): 0, (4, 6): 0, (4, 7): 0, (5, 2): 0, (5, 3): 0, (5,
  8): 0, (5, 9): 0, (6, 2): 0, (6, 3): 0, (6, 4): 0, (6, 8): 0, (6,
  9): 0, (6, 10): 0, (7, 3): 0, (7, 4): 0, (7, 9): 0, (7, 10): 0, (8,
   5): 0, (8, 6): 0, (8, 11): 0, (9, 5): 0, (9, 6): 0, (9, 7): 0, (9,
   11): 0, (10, 6): 0, (10, 7): 0, (10, 11): 0}
```

Printando a solução da modelagem pelo solver:

solution for: Problema do Fluxo de Custo Minimo objective: -715.000 $x_{12} = 245.000 x_{13} = 210.000 x_{14} = 260.000 x_{25} = 130.000 x_{26} = 115.000 x_{35} = 65.000 x_{36} = 35.000 x_{37} = 110.000 x_{46} = 140.000 x_{47} = 120.000 x_{58} = 110.000 x_{59} = 85.000 x_{68} = 110.000 x_{69} = 95.000 x_{61}0 = 85.000 x_{79} = 130.000 x_{71}0 = 100.000 x_{81}1 = 220.000 x_{91}1 = 310.000 x_{1}0_{1}1 = 185.000 x_{1}1_{1} = 715.000$

3.2 Exercício 9.4-3

3.2.1 Considerações e conceitos iniciais

O enunciado do exercício apresenta ao leitor um sistema de arquedutos que se origina de três rios, R1, R2 e R3 e passa por vários nós de transbordo até chegar a uma cidade T que, neste exercício, será o escoadouro. Para ser possível a modelagem desse problema, é necessário criar um nó fictício de origem que se liga a esses três rios (R1, R2, R3) e assim, esse rios se tornam nós de transbordo e o nó fictício passa a ser a origem. A capacidade de cada arco que liga o nó fictício a cada rio é igual à soma das capacidades dos arcos que saem do rio para outros vértices.

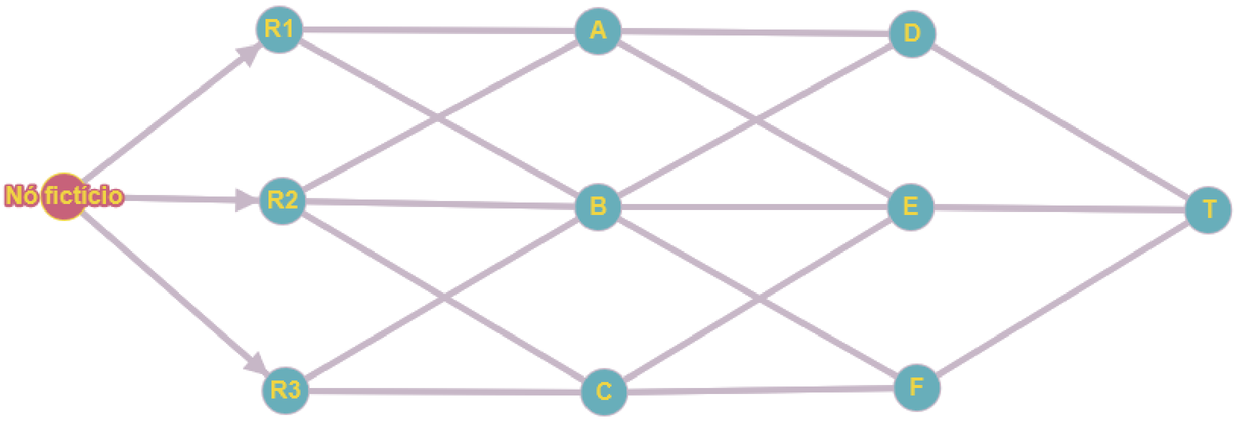

Figura 4: *Adição do nó fictício no grafo*

3.2.2 Formula Geral do PFM

Dado o Grafo G = (v,a)

- *v -> vértices v = {1,...,11}*
- *a -> arcos*
- *Sendo s a origem e t o escoadouro*

Max:

$$\sum_{(s,j)\in A} x_{sj}$$

S.a:

$$\sum_{(j,i)\in A} x_{ji} - \sum_{(i,j)\in A} x_{ij} = 0; \qquad \forall i \in V/\{s,t\}$$

$$0 \leqslant x_{ij} \leqslant m_{ij}; \qquad \forall (i,j) \in A$$

Sendo m_{ij} a capacidade máxima do arco $(_{i},_{j})$

3.2.3 Código de Modelagem do PFM

```
#Leitura e armazenamento dos dados do arquivo
import pylab as pb
from tkinter import *
from tkinter import filedialog
import os
import sys
import io


def chooseFileName():
    root.filename =  filedialog.askopenfilename(initialdir = currentDirectory,title = "Select file",filetypes = (("txt files","*.txt"),("all files","*.*")))
    print (root.filename)

def showModel():
    result = []
    if(root.filename != ''):
        #Armazenando os dados do numero de vertices, arcos e indices dos nos de origem e escoadouro
        instance_info = pb.loadtxt(root.filename, dtype = int, max_rows = 4)

        #Armazenando os dados dos arcos
        dataSet = pb.loadtxt(root.filename,dtype = int, skiprows = 4)

        #Criando um range para facilitar as operacoes com vertices
        vertices=[]
        verticesRange = range(instance_info[0])
        for i in verticesRange:
            vertices.insert(i, i + 1)
        print(instance_info)
        result.append(str(instance_info))
        print(dataSet)
        result.append(str(dataSet))
        print(vertices)
        result.append(str(vertices))

        #Criando um dicionario que organiza melhor o dataSet
```

```python
        #edge_capacity serve para relacionar os arcos com suas respectivas capacidades
        edge_capacity = {}
        for i in range(len(dataSet)):
            edge_capacity[(dataSet[i][0], dataSet[i][1])] = dataSet[i][2]

        #Modelagem
        from docplex.mp.model import Model
        mdl = Model("Problema do Fluxo Maximo")
        #Criando ranges
        dataRange = range(instance_info[0] + 1)
        source = vertices[instance_info[2]] #No de origem
        sink = vertices[instance_info[3]] #Escoadouro

        #Criamos as variaveis Xij
        x = {(i, j): mdl.continuous_var(name="x_{0}_{1}".format(i, j)) for i in vertices for j in vertices if (i, j) in edge_capacity}

        mdl.print_information()

        res = str(mdl.get_statistics()).split('\n')

        result.extend(res)



        #Adicionando restricoes de capacidade para cada variavel Xij
        for i in vertices:
            for j in vertices:
                if (i, j) in edge_capacity:
                    mdl.add_constraint(x[i, j] <= edge_capacity[i, j])
                    print(x[i, j])
                    result.append(str(x[i, j]))
                    print(edge_capacity[i, j]) #Observe no print como as variaveis condizem com seus arcos e suas capacidades maximas
                                                # de acordo com o dict edge_capacity na terceira celula de codigo
                    result.append(str(edge_capacity[i, j]))
```

```

    #Criando um array de nos de transbordo apenas:
    nodes = []
    j = 0
    for i in vertices:
        if i != source and i!=sink:
            nodes.insert(j, i)
            j += 1

    #Adicionando restricoes de conservacao de fluxo
    for i in nodes:
        mdl.add_constraint(mdl.sum(x[j, i] for j in vertices if (j, i) in edge_capacity) -
                        mdl.sum(x[i, j] for j in vertices if (i, j) in edge_capacity) == 0)

    #Criando a funcao objetivo
    mdl.maximize(mdl.sum(x[source, j] for j in vertices if (source, j) in edge_capacity))

    solution = mdl.solve(log_output=True)


    old_target = sys.stdout


    sys.stdout = open('solution.txt','w')


    solution.display() #printando no arquivo (que posteriormente e lido e jogado na janela do tkinter)

    sys.stdout.close()

    sys.stdout = old_target

    solution.display() #printando novamente, dessa vez no terminal


    r = open('solution.txt', 'r')

    # print("reeepeee: ")
```

```
        # print(r.read())
        # print("end r")

        #Printando o modelo
        print(mdl.export_to_string())

        info = (mdl.export_to_string()).split('\n')

        result.extend(info)

        res = str(mdl.get_statistics()).split('\n')

        result.extend(res)

        mdl.print_information()



        s = str(r.read()).split('\n')

        result.extend(s)

        mylist = Listbox(root, yscrollcommand = scrollbar.set,width
           =150,height=200)

        for line in result:
            mylist.insert(END, line)

        mylist.pack()




    else:
        print("O caminho fornecido e invalido")

root = Tk()

root.title('PFM-version4')

scrollbar = Scrollbar(root)
scrollbar.pack( side = RIGHT, fill = Y )
```

```
root.filename = ''


root.geometry("500x500") #You want the size of the app to be 500x500
#root.resizable(0, 0) #Don't allow resizing in the x or y direction


chooseButton = Button(root,text="Escolher arquivo",command =
    chooseFileName)
solveButton = Button(root,text="Exibir Modelo",command = showModel)




chooseButton.pack()
solveButton.pack()


currentDirectory = os.getcwd()



root.mainloop()
```

3.2.4 Resultados:

Printando os vértices:

```
[11 34  0 10]
[[  1   2 245]
 [  1   3 270]
 [  1   4 260]
 [  2   5 130]
 [  2   6 115]
 [  3   5  70]
 [  3   6  90]
 [  3   7 110]
 [  4   6 140]
 [  4   7 120]
 [  5   8 110]
 [  5   9  85]
 [  6   8 130]
```

```
 [  6   9  95]
 [  6  10  85]
 [  7   9 130]
 [  7  10 160]
 [  8  11 220]
 [  9  11 330]
 [ 10  11 240]
 [  5   2 130]
 [  5   3  70]
 [  6   2 115]
 [  6   3  90]
 [  6   4 140]
 [  7   3 110]
 [  7   4 120]
 [  8   5 100]
 [  8   6 130]
 [  9   5 330]
 [  9   6  95]
 [  9   7 130]
 [ 10   6  85]
 [ 10   7 160]]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Printando o Modelo:

Model: Problema do Fluxo Maximo

- number of variables: 34

- binary=0, integer=0, continuous=34

- number of constraints: 0

- linear=0

- parameters: defaults

- objective: none

- problem type is: LP

Printando as váriaveis $\boldsymbol{X}_{ij}$:

$\boldsymbol{x}_{12}$

245

x_{13}

270

x_{14}

260

x_{25}

130

x_{26}
115
x_{35}
70
x_{36}
90
x_{37}
110
x_{46}
140
x_{47}
120
x_{52}
130
x_{53}
70
x_{58}
110
x_{59}
85
x_{62}
115
x_{63}
90
x_{64}
140
x_{68}
130
x_{69}
95
x_{610}
85
x_{73}
110
x_{74}
120
x_{79}
130
x_{710}

160

x_{85}

100

x_{86}

130

x_{811}

220

x_{95}

330

x_{96}

95

x_{97}

130

x_{911}

330

x_{106}

85

x_{107}

160

x_{1011}

240

Printando as informações do solver:

CPXPARAM_Read_DataCheck 1

Tried aggregator 1 time.

LP Presolve eliminated 34 rows and 15 columns.

Reduced LP has 9 rows, 19 columns, and 33 nonzeros.

Presolve time = 0.02 sec. (0.03 ticks)

Initializing dual steep norms . . .

Iteration log . . .

Iteration: 1 Dual objective = 775.000000

Printando a solução da modelagem pelo solver:

solution for: Problema do Fluxo Maximo

objective: 715.000

$x_{12} = 245.000$

$x_{13} = 265.000$

$x_{14} = 205.000$

$x_{25} = 130.000$

$x_{26} = 115.000$

$x_{35} = 65.000$

$x_{36} = 90.000$

$x_{37} = 110.000$

$x_{46} = 85.000$

$x_{47} = 120.000$

$x_{58} = 110.000$

$x_{59} = 85.000$

$x_{68} = 110.000$

$x_{69} = 95.000$

$x_{610} = 85.000$

$x_{79} = 75.000$

$x_{710} = 155.000$

$x_{811} = 220.000$

$x_{911} = 255.000$

$x_1 0_{11} = 240.000$

Printando o modelo:

This file has been generated by DOcplex

ENCODING=ISO-8859-1

Problem name: Problema do Fluxo Maximo

Maximize

obj: $x_{12} + x_{13} + x_{14}$

$SubjectTo$

$c1 : x_{12} <= 245$

$c2 : x_{13} <= 270$

$c3 : x_{14} <= 260$

$c4 : x_{25} <= 130$

$c5 : x_{26} <= 115$

$c6 : x_{35} <= 70$

$c7 : x_{36} <= 90$

$c8 : x_{37} <= 110$

$c9 : x_{46} <= 140$

$c10 : x_{47} <= 120$

$c11 : x_{52} <= 130$

$c12 : x_{53} <= 70$

$c13 : x_{58} <= 110$

$c14 : x_{59} <= 85$

$c15 : x_{62} <= 115$

$c16 : x_{63} <= 90$

$c17 : x_{64} <= 140$

$c18 : x_{68} <= 130$

$c19 : x_{69} <= 95$

$c20 : x_{610} <= 85$

$c21 : x_{73} <= 110$

$c22 : x_{74} <= 120$

$c23 : x_{79} <= 130$

$c24 : x_{71}0 <= 160$

$c25 : x_{85} <= 100$

$c26 : x_{86} <= 130$

$c27 : x_{81}1 <= 220$

$c28 : x_{95} <= 330$

$c29 : x_{96} <= 95$

$c30 : x_{97} <= 130$

$c31 : x_{91}1 <= 330$

$c32 : x_{1}0_{6} <= 85$

$c33 : x_{1}0_{7} <= 160$

$c34 : x_{1}0_{1}1 <= 240$

$c35 : x_{12} - x_{25} - x_{26} + x_{52} + x_{62} = 0$

$c36 : x_{13} - x_{35} - x_{36} - x_{37} + x_{53} + x_{63} + x_{73} = 0$

$c37 : x_{14} - x_{46} - x_{47} + x_{64} + x_{74} = 0$

$c38 : x_{25} + x_{35} - x_{52} - x_{53} - x_{58} - x_{59} + x_{85} + x_{95} = 0$

$c39 : x_{26} + x_{36} + x_{46} - x_{62} - x_{63} - x_{64} - x_{68} - x_{69} - x_{61}0 + x_{86} + x_{96} + x_{1}0_{6} = 0$

$c40 : x_{37} + x_{47} - x_{73} - x_{74} - x_{79} - x_{71}0 + x_{97} + x_{1}0_{7} = 0$

$c41 : x_{58} + x_{68} - x_{85} - x_{86} - x_{81}1 = 0$

$c42 : x_{59} + x_{69} + x_{79} - x_{95} - x_{96} - x_{97} - x_{91}1 = 0$

$c43 : x_{61}0 + x_{71}0 - x_{1}0_{6} - x_{1}0_{7} - x_{1}0_{1}1 = 0$

$Bounds$

End

Printando informações do modelo:

Model: Problema do Fluxo Maximo

- number of variables: 34

- binary=0, integer=0, continuous=34

- number of constraints: 43

- linear=43

- parameters: defaults

- objective: maximize

- problem type is: LP

Plot do resultado da rede:

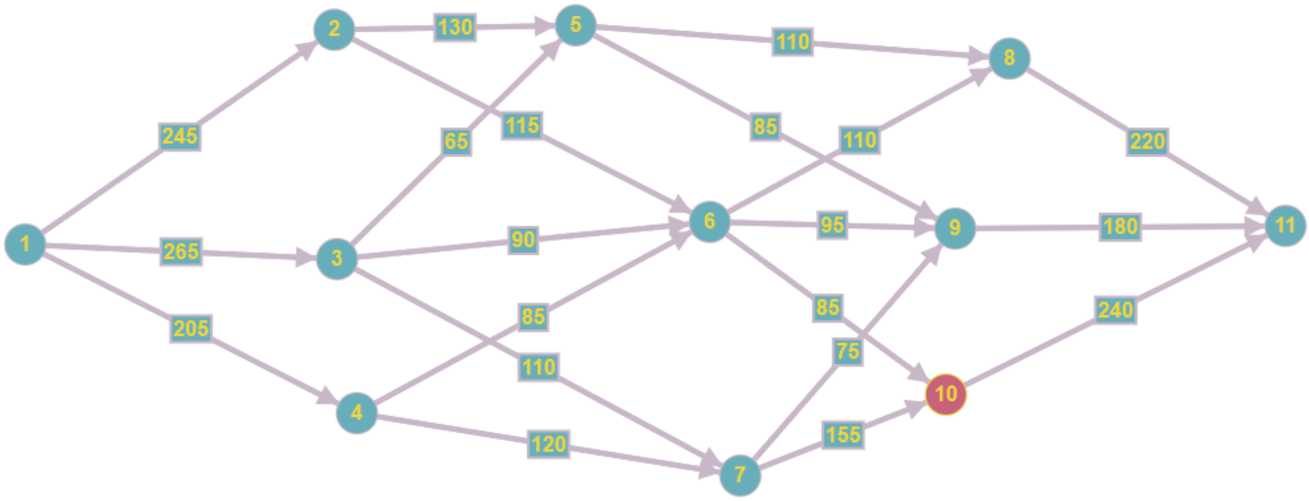

Figura 5: *Grafo com os fluxos que passam em cada arco do Exercicio 9.4-3*

4 CONCLUSÕES E TRABALHOS FUTUROS

Destarte, o problema imposto, tanto a modelagem como a transformação de um Problema de Fluxo Máximo(PFM) em um Problema de Fluxo de Custo Mínimo(PFCM) foram um sucesso, o resultado ótimo da função objetivo foram possíveis e encontrados utilizando o Cplex, dado o problema proposto modelado como PFM resultou em 715 e o resultado do PFCM -715.

REFERÊNCIAS

[1] J. H. Frederick e G. S. LIEBERMAN, *Introdução à Pesquisa Operacional*. [Mc Graw Will e Bookman]; 1/2013. Vol. Grupo A, 9788580551198. endereço: https://integrada.minhabiblioteca.com.br/#/books/9788580551198/.