Projeto de Graduação

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

19/12/2016

OTIMIZAÇÃO LINEAR: PROBLEMA DE FLUXO MÁXIMO

Rodrigo Gomide Antunes Dias

Renan Neumann Duriez Mendes

PUC RIO

Projeto de Graduação

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

OTIMIZAÇÃO LINEAR: PROBLEMA DE FLUXO MÁXIMO

Aluno(s): Rodrigo Gomide Antunes Dias

Renan Neumann Duriez Mendes

Orientador: Marco Antônio Grivet Mattoso Maia

Trabalho apresentado com requisito parcial à conclusão do curso de Engenharia Elétrica na Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil.

Agradecimentos

Gostaria de agradecer, primeiramente, aos que até aqui foram o maior motivo de meu esforço. Aqueles que estiveram em todos os momentos da minha formação acadêmica e moral. Pai e mãe, obrigado por tudo. Agradeço também aos professores que de alguma forma contribuíram à minha escalada. Cito alguns: Prof.a Ana Maria Beltrani Pavani; Prof. Eduardo; Prof. Gláucio Lima Siqueira. Agradeço especialmente ao Orientador desse projeto que teve início em março de 2016, Prof. Marco Grivet, obrigado pela atenção, cuidado, pelos ensinamentos e boas risadas, obrigado também por prontamente se mostrar solícito conosco e todos os alunos. Agradeço aos amigos de graduação, todos devem saber o quanto eles são não só companheiros de jornada, mas também, grandes motivadores em todos os momentos. A graduação possivelmente levaria mais tempo sem vocês, obrigado. Agradeço, finalmente, aos familiares, amigos e afetos, que seguraram a barra, a ausência, que deram colo e me impulsionaram à conclusão desse ciclo, obrigado pelo carinho.

Resumo

O projeto envolveu a elaboração de um código que cobre o tema de otimização em redes de telecomunicações. Ele irá apresentar uma solução para a questão clássica de fluxo máximo. O problema consiste em otimizar a utilização dos enlaces de forma a distribuir o tráfego de dados e, dessa forma, fazer com que a rede trabalhe com folga em relação ao seu fator de utilização máximo. Para a solução do problema estudado utilizamos a ferramenta *Matlab*, e o script foi baseado no uso da função *linprog.* Os dados de rede utilizados nesse trabalho foram coletados do site http://www.sndlib.zib.de/ em formato .txt e posteriormente lidos e convertidos para formato .mat. Os dados tratados na resolução do projeto foram: capacidade em cada enlace; tráfego total de um nó para outro; número de nós; números de enlaces; conexões existentes entre os nós. De forma geral nosso objetivo é minimizar a variável Z, como segue:

Palavras-chave: Otimização Linear; programação linear; técnicas de decomposição; tráfego de dados; redes de telecomunicações;

DEPARTAMENTO
DE ENGENHARIA
ELÉTRICA

Optimization programming in Telecommunication Networks

Abstract

The project's main goal involves writing a programming script that optimizes telecommunication networks. It will lead us to a solution for the classical Optimal Effectiveness problem. The problem consists in optimize the utilization factor of links in a way that the traffic of data becomes well distributed between them. By doing that we guarantee a loose utilization factor. For the solution of the problem we used the software MATLAB, the script was based on the employ of linprog function. The network data used for this paper was collected from http://www.sndlib.zib.de/ website in .txt format and later read and converted to .mat format. The data handled in the resolution of our project was: capacity of each link; total traffic from a node to another; quantity of nodes; quantity of links; existing links between nodes. In a general scenario our objective is to minimize variable Z, as it follows:

Keywords: Linear optimization; decomposition techniques in mathematical programming; linear programming; flow network; traffic flow.

Sumário

Lista de figuras

Lista de Tabelas

1. Introdução

Na disciplina "Seminário I" do departamento de matemática foi proposto um trabalho envolvendo algum tema da área. O orientador do projeto foi o Dr. Marco Antônio Grivet Mattoso Maia que havia sido professor dos autores em algumas disciplinas da engenharia elétrica. O tema sugerido foi "Otimização de redes de telecomunicações" e é de grande interesse dos alunos, uma vez que eles estagiam na área. Foi através daí que houve a motivação para continuar a pesquisa e utilizá-la como Projeto de Final de Curso.

Dependendo do tipo das variáveis, bem como a natureza da função objetivo e das restrições, os problemas são classificados de diferentes maneiras. Se as variáveis envolvidas são contínuas e tanto a função objetivo quanto as restrições são lineares, o problema é denominado "problema de programação linear." Se alguma das variáveis envolvidas é número inteiro ou binário, enquanto as restrições e a função objetivo são ambos lineares, o problema é denominado "inteiro-misto de programação linear. " Analogamente, se a função objetivo ou qualquer restrição é não-linear e todas as variáveis são contínuas, o problema é denominado "problema de programação não-linear". Se, além disso, qualquer variável é inteira, o problema correspondente é denominado "inteiro-misto de programação não-linear." [1]

O assunto de otimização linear surge com a necessidade de maximizar ou minimizar uma solução, dado um sistema linear de equações e inequações relacionadas a uma situação real. Tais situações são diversas, como por exemplo, uma fábrica que quer maximizar seu lucro minimizando o uso de matéria-prima, ou uma cantina que pretende variar o cardápio para uma dieta mais saudável considerando mais nutrientes e menos desperdício e, ainda, minimizando o custo da refeição.

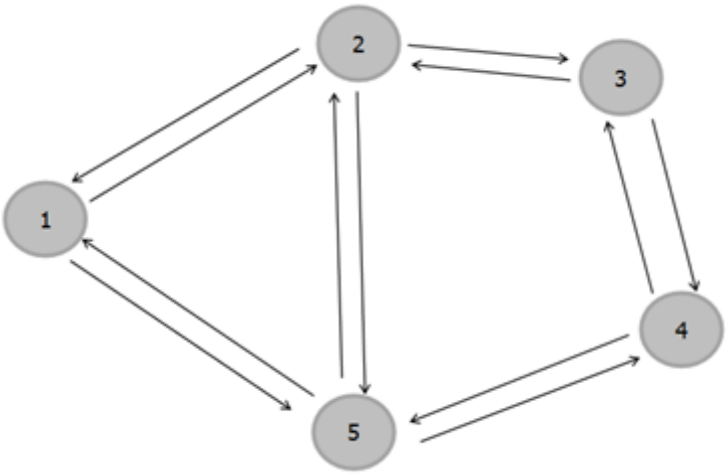
As aplicações são inúmeras, mas aqui focaremos no caso particular da otimização em Redes de Telecomunicações em programação linear, onde pretende-se minimizar uma função de fluxo de dados. A topologia de uma rede, que relaciona todas os dados envolvidos no problema, pode ser representada em um plano, o que facilita a compreensão do mesmo. Nesse, as restrições delimitam uma área e as soluções ótimas se encontram nos vértices dessa área.

É importante mencionar que mesmo para uma rede real, nem sempre será possível representá-la como um problema de Otimização. Isso porque, para alguns sistemas pode não existir solução que satisfaça todas as restrições (problema insolúvel - infeasible) – não há ponto da função objetivo dentro da área delimitada pelas restrições. A discussão mais avançada sobre infeasibility ultrapassa o escopo desse trabalho.

Há um problema clássico em empresas gestoras de redes, que é o balanceamento do tráfego de dados utilizados por cada enlace e por cada equipamento integrante do sistema. Esse problema ocorre por diversos motivos: variação de usuários e pacote de dados utilizados; eventos esporádicos; falhas no sistema; erro técnico de balanceamento; e outros similares.

Em resumo, redes de telecomunicações funcionam a partir do conceito matemático de grafo orientado: "consiste de um conjunto de nós (também chamado vértices) e ramos direcionais entre os nós, qualquer matriz quadrada tem um grafo associado, e qualquer grafo tem um padrão de matriz quadrada associada" [2]. Os nós representam as antenas/roteadores e as arestas seriam os enlaces. Cada nó recebe e transmite informações continuamente, sendo essas provenientes de outros ou de si mesmo. As arestas são os caminhos por onde os dados trafegam. Entre vértices interligados há dois enlaces, ou seja, o caminho é bidirecional.

Para esse projeto, foi analisado o problema de fluxo máximo. Esse é baseado na distribuição do tráfego de dados de forma a otimizar a utilização de cada enlace. Fazendo assim, com que cada um tenha a menor taxa de operação. Esse balanceamento serve para dar mais segurança à rede e, dessa forma, atuará com a maior folga possível. Assim, há maior probabilidade de o sistema manter-se estável para casos extraordinários.



Exemplo de rede

2. Teoria

Todo problema de otimização linear segue um conjunto de restrições apresentadas na forma de equações e inequações que fazem com que o sistema obedeça as leis de preservação de fluxo. Além dessas, há também a função a ser minimizada ou maximizada, chamada função objetivo.

A formulação base de um problema de otimização tem a seguinte forma:

$$A.x = b$$

$$B.x \leq C_e$$

$$x \geq 0$$

A primeira equação mostrada acima é dita uma condição de igualdade do sistema. Ela relaciona a direção do fluxo em cada enlace com o tráfego de um nó para outro. A segunda é dita uma condição de desigualdade e estabelece um limite para a transferência de dados. Esse limite é definido pela capacidade que cada enlace suporta. Tráfegos acima desse valor significam perda de dados. A terceira equação nos diz que a variável x é necessariamente um número real positivo, uma vez que essa variável significa o fluxo de dados nos enlaces.

O problema estudado apresenta variáveis e parâmetros que serão necessárias para o desenvolvimento do código. Esses são apresentados e definidos a seguir:

Notação

$\delta^{-}(v) = conjunto\ de\ enlaces\ com\ direção\ ao\ nó\ v$

$\delta^{+}(v) = conjunto\ de\ enlaces\ provenientes\ do\ nó\ v$

nV = número de vértices da rede

nE = número de enlaces da rede

Variáveis

Variáveis	Definição	Quantidade	Natureza
x_{et}	Fluxo ECMP no link e destinado ao nó t induzido pelo sistema w	nE.nV	$\in \mathbb{R}^{+} \cup \{0\}$
z_{st}	Valor comum ao fluxo ECMP destinado ao nó t atribuído à todos outlinks do nó s pertencentes aos menores caminhos do nó s ao nó t	nV.nV	$\in \mathbb{R}^{+} \cup \{0\}$
r_{st}	Distância do menor caminho(w) do nó s destinado ao nó t	nV.nV	$r_{st} \geq, r_{ss} = 0$
w_e	Peso atribuído ao link e	nE	$\in \{1,2,..,K_0\}$
Z	Máximo fator de utilização sobre link	1	$\in (0,1]$
u_{et}	Variável indicando se o link e está no menor caminho destinado ao nó t	nE.nV	$\in \{0,1\}$

Tabela de variáveis

Nota: Há uma diferença importante a se observar entre menor caminho e o que aqui chamamos menor caminho(w). Menor caminho é o conjunto de links consecutivos interligados através de nós que juntos formam o caminho físico mais curto. O que nomeamos menor caminho(w) é o conjunto de links consecutivos interligados por nós que consideram os pesos w_e, e assim, formam o caminho mais eficiente (custo e distância) para se passar informações.

Parâmetros

$d_{vt} - Tráfego\ total\ que\ sai\ do\ nó\ v\ destinado\ ao\ nó\ t$

$D_t - Tráfego\ total\ destinado\ ao\ t \qquad D_t = \sum_{v=1}^{nE} d_{vt}$

$C_e - Capacidade\ do\ enlace\ e$

Em posse dessas informações podemos então definir as matrizes que farão parte da solução computacional:

- Matrizes A: estão relacionadas ao sentido do fluxo que sai de cada nó. Tem dimensão nV.nE e obedece a seguinte regra de formação:

$$A_t(i,j) = \begin{cases} -1 & i \neq t, j \in \delta^+(t) \\ +1 & j \in \delta^-(t) \\ 0 & caso\ contrário \end{cases}$$

  Onde t é o índice da matriz A e corresponde ao nó de interesse.

- Vetor a: é o vetor que contém as condições de igualdade do sistema. Tem dimensão nV e obedece a seguinte regra de formação:

$$a_t(k) = \begin{cases} D_t & k = t \\ d_{kt} & k \neq t \end{cases}$$

- Matrizes G: Tem dimensão nE.nE e obedece a seguinte regra de formação:

$$G_t(i,j) = \begin{cases} 1 & i = j, t \in V \backslash a(i) \\ 0 & caso\ contrário \end{cases}$$

- Vetor K: Tem dimensão nE.1 e é negativo. Ele é igual ao negativo da capacidade de cada enlace.

- Vetor X*et*: Tem dimensão nE.nV e será utilizado para a obtenção de Z. É aquele que contém as informações do fluxo no enlace *e* destinado ao nó *t*.

3. Desenvolvimento

A preparação para o trabalho envolveu a leitura de alguns livros, dentre eles um do dr. Conejo; alguns capítulos de modelagem computacional também foram estudados; além de várias reuniões e conversas com o orientador do projeto.

Sabemos que problemas de programação linear podem ser muito grandes, uma vez que consideram inúmeras variáveis e restrições. Para facilitar a resolução de alguns problemas usa-se técnicas de decomposição por blocos. Para que o uso de tais práticas seja possível, o problema deve ter uma estrutura apropriada. Aqui, estudaremos apenas redes que seguem essa 'facilidade'.

No problema estudado iremos otimizar uma rede de telecomunicações levando em consideração as variables and constraints ECMP e w-System. O que faremos é dividir as variáveis e restrições em blocos de acordo com seu uso. As matrizes obedecem o seguinte modelo que posteriormente será usado na otimização da variável Z anteriormente descrita. P

Através dessa elucidação foi possível o início do projeto: através da ferramenta computacional MatLab, foram introduzidas as variáveis de acordo as leis de formação abaixo demonstradas. Primeiramente, foi utilizado a função "fscanf" para ler os dados do arquivo .txt, assim passou-se todos os dados necessários para o roteiro do código. E partir daí iniciou-se o desenvolvimento do projeto.

$$\min z = cT \cdot \underline{x} + 0T \cdot \underline{z} + 0T \cdot \underline{r} + 0T \cdot \underline{w} + 0T \cdot \underline{u} \quad sujeito\ a$$

$$\begin{aligned} A \cdot \underline{x} &= r0 \\ R \cdot \underline{x} &\leq 0 \\ B \cdot \underline{x} + C \cdot \underline{z} - D \cdot \underline{u} &\leq s0 \\ F \cdot \underline{r} + H \cdot \underline{w} + E \cdot \underline{u} &\leq t0 \end{aligned}$$

$$\underline{x}, \underline{z} \geq 0$$
$$\underline{x}(1) \leq 1$$
$$\underline{r}, \underline{w} \geq 1$$
$$\underline{u}\ vetor\ binário$$

Restrições e igualdades 1

Vetor x (da forma que aplicado no programa):

$$x = \begin{pmatrix} Z \\ Xe1 \\ Xe2 \\ \vdots \\ Xev \end{pmatrix}$$

O passo seguinte foi definir as matrizes A, Dvt, G, que seguem abaixo:

- Organização da matriz A:

$$A = \begin{pmatrix} 0 & \boxed{A1} & & \\ 0 & & \boxed{A2} & \\ 0 & & & \ddots \\ \vdots & & & \\ 0 & & & \boxed{AnV} \end{pmatrix}$$

- Matriz R (envolvendo G e K):

$$R = \left(\begin{array}{c|c|c|c} -C1 & G1 & \ldots & Gv \\ -C2 & & & \\ \vdots & & & \\ -Ce & & & \end{array}\right)$$

Ela tem dimensão nV.nE+1 e seu primeiro elemento é o único não nulo. A função linprog foi usada com seus respectivos parâmetros com a finalidade de obter os resultados: o agrupamento e vetores x, e o fator Z de utilização dos enlaces.

Em seguido temos as matrizes B, C e D, que pertencem ao conjunto de inequações do vetor b

- Matriz B

$$B = \begin{pmatrix} B1 & 0 & 0 & 0 \\ 0 & B2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & BnV \end{pmatrix}$$

Todo elemento da matriz B pertence ao conjunto {-1,0,1}

- Matriz C

$$C = \begin{pmatrix} C1 & 0 & 0 & 0 \\ 0 & C2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & CnV \end{pmatrix}$$

Todo elemento da matriz B pertence ao conjunto {-1,0,1}

- Matriz D

$$D = \begin{pmatrix} D1 & 0 & 0 & 0 \\ 0 & D2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & DnV \end{pmatrix}$$

Todo elemento da matriz D pertence ao conjunto dos Reais

- Vetor b (envolvendo a):

$$b = \begin{pmatrix} a1 \\ a2 \\ \vdots \\ av \end{pmatrix}$$

Por fim, temos o último bloco de matrizes E, F e G que correspondem ao conjunto de inequações do vetor c

- Matriz E

$$E = \begin{pmatrix} E1 & 0 & 0 & 0 \\ 0 & E2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & EnV \end{pmatrix}$$

Todo elemento de D pertence ao conjunto {-1,0,M}

- Matriz F

$$F = \begin{pmatrix} F1 & 0 & 0 & 0 \\ 0 & F2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & FnV \end{pmatrix}$$

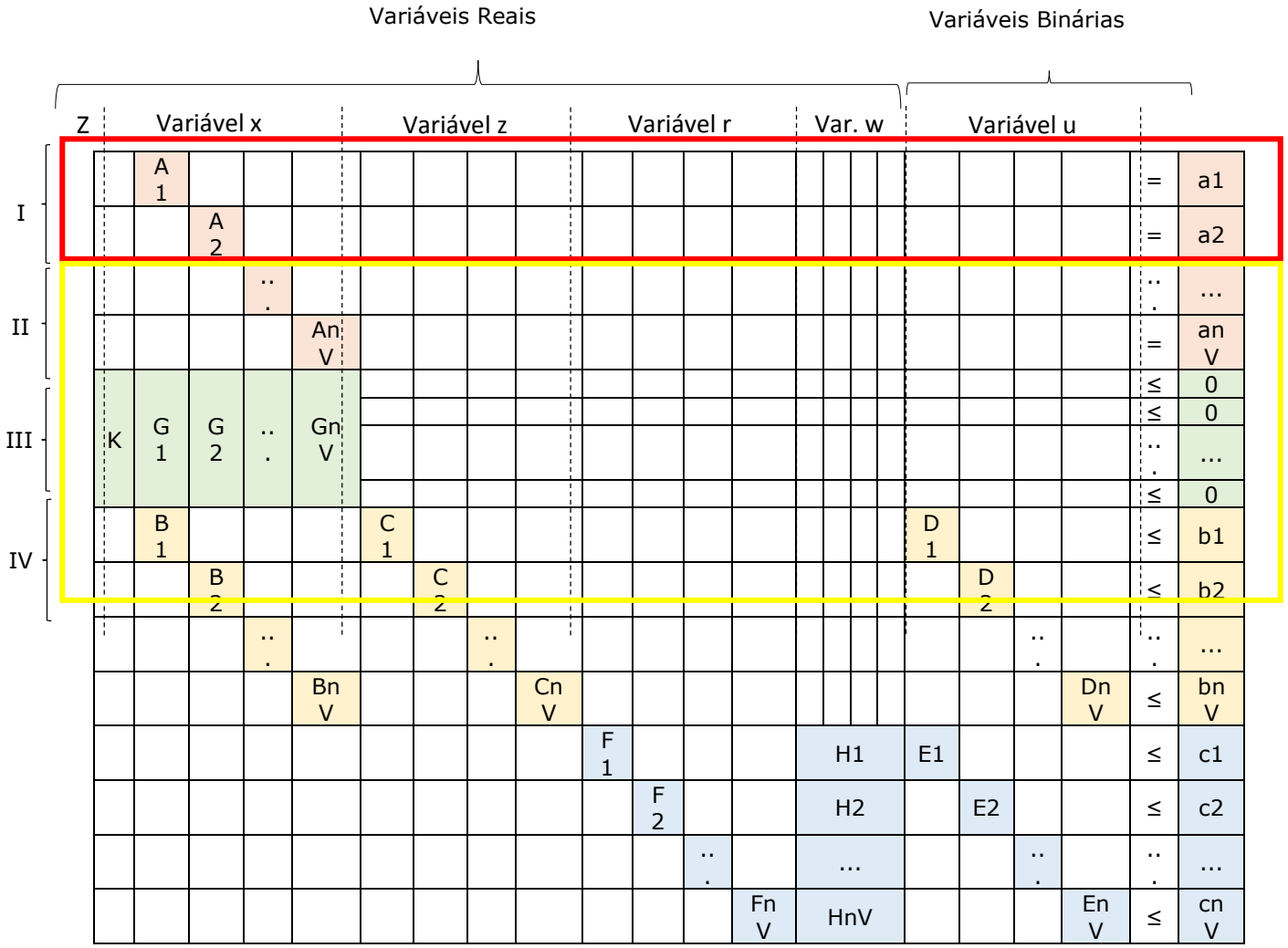
Todo elemento de F pertence ao conjunto {-1,0,1}

- Matriz H

$$H = \begin{pmatrix} H1 \\ H2 \\ \vdots \\ HnV \end{pmatrix}$$

Todo elemento de H pertence ao conjunto {-1,0,1}

Agora já nos encontramos em posse de todas as matrizes necessárias para a formulação do sistema a ser entregue para a função *linprog* do Matlab. Porém, essas ainda não se encontram na forma desejada. A tabela 4, a seguir, é uma arrumação matricial do sistema de equações e inequações proposto na figura 1. A partir dela conseguimos visualizar de forma bem definida o problema que queremos resolver.



Matriz de resolução

A quantidade de variáveis e restrições obtidas desse sistema está única e diretamente ligada ao tamanho da rede, isto é, o número de nós e enlaces da mesma. A relação pode ser observada pelas seguintes equações:

Número total de variáveis: $2 \cdot nV \cdot (nE + nV) + nE + 1$ Eq. (1)

Número total de restrições: $(nV)^2 + nE + 5 \cdot nE \cdot (nV - 1)$ Eq. (2)

A matriz de resolução está subdividida em três menores grupos: quanto à natureza da variável – Real ou Binária; quanto ao tipo de equação – igualdade ou desigualdade; quanto à equação de fluxo que o bloco obedece – sendo esses: *Flow Preservation, Flow Restriction*, ECMP, W-System. O bloco enquadrado em vermelho é o que leva a restrição de igualdade prevista anteriormente. Em amarelo está o bloco que obedece às restrições de desigualdade. Veremos posteriormente que a necessidade de separá-los dessa forma surge meramente por questões computacionais. O terceiro tipo de subdivisão merece atenção especial e, portanto, nos prolongaremos um pouco mais nele.

I. *Flow Preservation Constraints* – É o conjunto de equações de preservação do fluxo nos enlaces. As equações estão descritas a seguir:

Para todo t ∈ V

$$\sum_{e \in \delta^-(t)} x_{et} = D_t$$

Para todo v ∈ V \ {t}

$$\sum_{e \in \delta^+(v)} x_{et} - \sum_{e \in \delta^-(v)} x_{et} = d_{vt}$$

II. *Flow Restriction Constraints*

Para todo e ∈ E

$$\sum_{t \in V \setminus a(e)} x_{et} \leq Z \cdot C_e \leftrightarrow \sum_{t \in V \setminus a(e)} x_{et} = Z \cdot C_e \leq 0$$

III. *ECMP Constraints*

Para todo e ∈ E; Para todo t ∈ V\{a(e)}

$$\left.\begin{array}{l} 0 \leq x_{et} \leq D_t.u_{et} \\ 0 \leq z_{a(e),t} - x_{et} \leq D_t.(1-u_{et}) \end{array}\right\} \Leftrightarrow \begin{cases} x_{et} - D_t.u_{et} \leq 0 \\ x_{et} - z_{a(e),t} \leq 0 \\ -x_{et} + z_{a(e),t} + D_t.u_{et} \leq D_t \end{cases}$$

$$\left.\begin{array}{l} 0 \leq x_{et} \leq D_t \cdot u_{et} \\ 0 \leq z_{a(e),t} - x_{et} \leq D_t \cdot (1-u_{et}) \end{array}\right]$$

IV. *W-System Constraints*

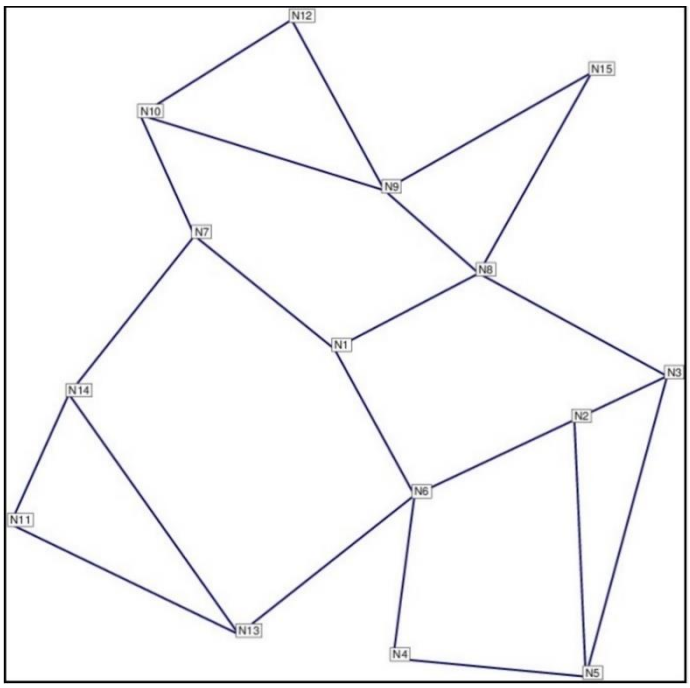
Para todo e ∈ E; Para todo t ∈ V\{a(e)}

$$1 - u_{et} \leq w_e + r_{b(e),t} - r_{a(e),t} \leq M \cdot (1-u_{et}) \leftrightarrow \begin{cases} -w_e - r_{b(e),t} + r_{a(e),t} - u_{et} \leq -1 \\ w_e + r_{b(e),t} - r_{a(e),t} + M \cdot u_{et} \leq M \end{cases}$$

4. Resultados

Após a execução do programa, o software nos apresenta um resultado de forma rápida, poucos segundos. Recebe-se nesse momento os valores de Xet e Z, e em X sabemos o quanto passa em cada enlace com destino a um nó e em Z a máxima ocupação. É esperado que os valores dessas variáveis respeitem a capacidade de cada enlace e, dessa forma, respeite os limites da rede. É possível que a solução exista, porém seja inviável. Isso ocorre quando a capacidade da rede está inadequada para o fluxo que nela passa. Assim, a medida para solucionar é aumentar a capacidade dos enlaces, e dessa forma haverá uma resposta viável.

Abaixo segue uma ilustração da rede que utilizamos como um exemplo do programa:



Rede Atlanta 1

A rede utilizada é chamada de Atlanta, rede essa que possui 22 enlaces e 15 nós. O programa usado para a criação das variáveis e, portanto o primeiro a ser 'rodado' teve tempo de execução de 2.86 segundos. Tempo consideravelmente baixo em se tratando de 2475 variáveis e 3349 restrições.

```
============ Best FULL Flow Program ================

    Nodes       :     15
    Links       :     44
    Variables   :   2475
    Constraints :   3349

Optimization terminated.

    Variável Z otimizada: 0.66

    Duração: 2.86 segundos.
```

O mais interessante do algoritmo de fluxo ótimo criado é que o tempo de cálculo é muito pequeno. O que é muito relevante para casos que a rede é muito dinâmica, já que o caminho ótimo iria, assim, alterar-se com o tempo. Dessa forma, o modelo computacional acompanharia essa mudança quase que instantaneamente.

A apresentação de dados para Atlanta ficaria inviável de ser mostrada aqui, pois se trata de matrizes enormes. Dessa forma, escolhemos uma rede, a qual nomeamos "rede teste". Essa possui 5 nós e 12 enlaces. E os resultado seguem a seguir:

Fluxo no link "e" destinado ao nó "t" induzido pelo sistema "w"

x =

0	0	4.71	0	0	6.19	45.71	49.29	12.05	23.36	16.64	9.55
28.43	6.27	0	0	0	4.41	0	24.71	51.44	60.13	29.87	24.54
9.60	31.52	39.99	0	0	13.62	21.11	0	0	21.51	58.49	5.13
35.76	8.71	6.05	60.46	14.54	0	10.28	19.19	31.02	0	0	64.54
22.91	18.24	9.87	28.68	85.10	24.90	12.16	8.99	7.79	0	13.77	-13.77

Distância do menor caminho(w) do nó s destinado ao nó "t"

r =

-1550	-1035	-1030	-968	-935
-1069	-1397	-1072	-973	-958
-1048	-1015	-1465	-945	-1027
-956	-1102	-932	-1566	-987
-914	-936	-981	-1091	-1713

Valor comum ao fluxo destinado ao nó t atribuído à todos outlinks do nó s pertencentes aos menores caminhos do nó s ao nó t

z =

0	50.15	53.40	36.72	32.37
47.54	0	64.16	75.89	54.86
49.17	54.63	0	46.70	71.23
50.77	65.72	48.01	0	73.30
43.84	39.90	86.98	32.35	0

```
Peso atribuído ao link "e"
w =

        1773
        1873
        1911
        1792
        1631
        1915
        1737
        1782
        1828
        1653
        1738
        1780
```

```
Variável indicando se o link "e" está no menor caminho destinado ao nó "t"
u =

    -1130    -1130        0        0        0        1        1        1        0        1        1        1
        1        1    -1130    -1130        0        0    -1130        1        1        1        1        1
        1        1        1        0    -1130        1        1    -1130    -1130        1        1        0
        1        1        0        1        1    -1130        0        1        1    -1130        0        1
        1        1        1        1        1        1        1        0        0        1    -1130    -1130
```

A matriz acima não foi inteiramente mostrada por problema de escala, mas ela representa o fluxo otimizado em cada enlace.

Interpretando o resultado podemos observar que os limites máximos não são ultrapassados e que a maior porcentagem de utilização de um enlace, definido pela variável z, foi de 0.14982, ou seja, na otimização da rede Abilene tivemos apenas 14.98% de uso no enlace com maior tráfego, valor bem distante da saturação comumente estipulada de 70%. Com a obtenção dessas respostas pode-se observar o quanto foi otimizado o tráfego na rede sem sobrecarregar nenhum dos enlaces. Podendo, agora, operar esse sistema com maiores folgas e margens de segurança, garantindo assim a estabilidade e qualidade.

5. Dificuldades

Os problemas iniciais foram principalmente a forma que seria escrito o código para que as variáveis fossem montadas de acordo com a teoria e um estudo mais aprofundado sobre o assunto de redes, fluxo de dados e otimização. Além disso, havia a necessidade de que o programa tivesse aplicação para diferentes redes, com variadas quantidades de nós e enlaces, assim deveria-se escrever um código generalizado. A formulação das equações e consequentemente a relação destas com a função linprog gerou uma confusão inicial, mas após uma profunda análise esse problema foi reduzido.

Por estagiarmos em uma empresa de telecomunicações, pode-se ver na prática como a pesquisa que estava sendo abordada era utilizada e que tipos de melhorias poderiam ser propostas. Na época, muitos casos foram vistos que o balanceamento do tráfego em cada central da rede era feito de forma manual e adinâmica. Seriam esses problemas que o projeto em estudo poderia ser utilizado como uma melhoria.

6. Bibliografia

[1] E. Castillo, R. Mínguez, A. J. Conejo e R. García-Bertrand, Decomposition Techniques in Mathematical Programming, Springer-Verlag Berlin Heidelberg, 2006.

[2] L. J. Jardim, "Análise de redes," Engenharia Elétrica PUC-Rio, Rio de Janeiro, 2016.

[3] Zuse-Institute Berlin (ZIB), "SndLIB," 2006. [Online]. Available: http://sndlib.zib.de/home.action. [Acesso em Agosto 2016].

[4] Emory University, "Emory Math/CS," Department of Mathematics and Computer Science, [Online]. Available: http://www.mathcs.emory.edu/~cheung/Courses/323/Syllabus/NetFlow/max-flow-lp.html. [Acesso em Agosto 2016].

7. Apêndice 1 (código)

```
function GenLinProg(fname,BigM)
%
%Função que vai gerar as matrizes necessárias para rodar a função
%BestFULLFlow.
%
    clc
    clear all
    tic;

    BigM=10000;
    IN  = -1;
    OUT = +1;

    fileID = fopen('redeteste.txt','rt');
    nV = fscanf(fileID,'%d',1);
    NB = 3+0;
    Nodes = fscanf(fileID,'%f',[NB,nV])';
    nE = fscanf(fileID,'%d',1);
    Links = fscanf(fileID,'%d',[4,nE])';
    dvt  = fscanf(fileID,'%d',[nV,nV])';
    fclose(fileID);

    Dt = sum(dvt);

    NumVarZ = 2*nV*(nE+nV)+nE+1;
    NumVarE = NumVarZ-1+nE+nV;
    NumCstZ = nV*nV+nE+5*nE*(nV-1);
    NumCstE = NumCstZ+2*nE+nE*nV;
    fprintf(1,'\tNós          : %6d\n',nV);
    fprintf(1,'\tEnlaces      : %6d\n',nE);
    fprintf(1,'\tVariáveis    : %6d (Z)   %6d (E)\n',NumVarZ,NumVarE);
    fprintf(1,'\tRestrições   : %6d (Z)   %6d (E)\n',NumCstZ,NumCstE);
    fprintf(1,'\tBig-M        : %6d\n\n',BigM);

%
%% ============ Preservação de Fluxo - Restrições de Igualdade ============
%  ========================================================================

     A_mem = zeros(nV,nE,nV);
     a_mem = zeros(nV,nV);
    for t=1:nV
        for v=1:nV
            LISTA0 = delta(v,IN);
            if v == t
                A_mem(v,LISTA0,t) = 1;
                a_mem(v,t) = Dt(t);
            else
                LISTA1 = delta(v,OUT);
                A_mem(v,LISTA1,t) =  1;
                A_mem(v,LISTA0,t) = -1;
                a_mem(v,t) = dvt(v,t);
```

```
            end
        end
    end


fprintf(1,'\tPreservação de Fluxo - Restrições de igualdade geradas...\n');


%% ========== Preservação de Fluxo - Restrições de Desigualdade ==========
%  ========================================================================

G_mem = zeros(nE,nE,nV);
K_mem = -Links(:,4);      %Matriz que guarda as capacidades dos enlaces
for e=1:nE
    for t=1:nV
        if t ~= Links(e,2)
            G_mem(e,e,t) = 1;
        end
    end
end
fprintf(1,'\tPreservação de Fluxo - Restrições de desigualdade geradas...\n');


%% ================== ECMP - Restrições de Desigualdade ===================
%  ========================================================================

n2E1 = 3*nE;  %  over-estimation
ecmp_size = zeros(1,nV);
B_mem = zeros(n2E1,nE,nV);
C_mem = zeros(n2E1,nV,nV);
D_mem = zeros(n2E1,nE,nV);
b_mem = zeros(n2E1,nV);
for e=1:nE
    ae = ORIGIN(e);
    for t=1:nV
        if t ~= ae
            pos = ecmp_size(t);
            pos = pos + 1;
            B_mem(pos,e,t) = 1;
            D_mem(pos,e,t) = -Dt(t);
            pos = pos + 1;
            B_mem(pos,e,t)  =  1;
            C_mem(pos,ae,t) = -1;
            pos = pos + 1;
            B_mem(pos,e,t)  = -1;
            C_mem(pos,ae,t) =  1;
            D_mem(pos,e,t)  = Dt(t);
            b_mem(pos,t)    = Dt(t);
            ecmp_size(t) = pos;
        end
    end
end
fprintf(1,'\tECMP Inequality Constraints generated...\n');
```

```
%% ================ W-System - Restrições de Desigualdade ================
%  =======================================================================
n2E1 = 2*nE;
E_mem = zeros(n2E1,nE,nV);
F_mem = zeros(n2E1,nV,nV);

H_mem = zeros(n2E1,nE,nV);
c_mem  = zeros(n2E1,nV);



ws_size = zeros(1,nV);
for e=1:nE
    ae = ORIGIN(e);
    be = DESTINATION(e);
    for t=1:nV
        if t ~= ae
% -------------------------------------------------------------------------
            ws_size(t) = ws_size(t)+1;
            pos = ws_size(t);
            E_mem(pos,e,t)  = -1;
            F_mem(pos,ae,t) =  1;
            F_mem(pos,be,t) = -1;
            H_mem(pos,e,t)  = -1;
            c_mem(pos,t) = -1;
% -------------------------------------------------------------------------
            ws_size(t) = ws_size(t)+1;
            pos = ws_size(t);
            E_mem(pos,e,t)  =  BigM;
            F_mem(pos,ae,t) = -1;
            F_mem(pos,be,t) = 1;
            H_mem(pos,e,t)  = 1;
            c_mem(pos,t)    = BigM
% -------------------------------------------------------------------------
        end
    end
end
fprintf(1,'\tw-System Inequality Constraints generated...\n');


%% ============== Salva as matrizes em .mat para uso futuro ==============
%  =======================================================================

Delta_Size = ws_size/2;

save('variables.mat','nE','nV','Delta_Size','Dt',...
                     'A_mem','a_mem',...
                     'B_mem','C_mem','D_mem','b_mem',...
                     'E_mem','F_mem','H_mem','c_mem',...
                     'G_mem','K_mem');


rname = sprintf('%s_check.dat','variables.mat');
fileID = fopen(rname,'wt');
```

```
    print_vector(Delta_Size,'Delta_Size');

    print_matrix(A_mem,'A');
    print_vector(a_mem,'a');

    print_matrix(B_mem,'B',ecmp_size);
    print_matrix(C_mem,'C',ecmp_size);
    print_matrix(D_mem,'D',ecmp_size);
    print_vector(b_mem,'b',ecmp_size);

    print_matrix(E_mem,'E',ws_size);
    print_matrix(F_mem,'F',ws_size);


    print_matrix(H_mem,'H',ws_size);
    print_vector(c_mem,'c',ws_size);

    print_matrix(G_mem,'G');
    print_vector(K_mem,'K');

fclose('all');
fprintf(1,'\tFile %s generated...\n',rname);

fprintf('\n\tDuração: %.2f segundos.\n\n',toc);

%% ======================== Função Delta ================================
function lista = delta(v,DIR)

%  Se DIR = +1 gera a lista de edges cujo vértice origem  é x
%  Se DIR = -1 gera a lista de edges cujo vértice destino é x

    if DIR == +1
        lista = find(Links(:,2)== v);
    else
        lista = find(Links(:,3)== v);
    end
end

%% ======================== Função ORIGIN ===============================
function t = ORIGIN(e)
    t = Links(e,2);
end

%% ======================== Função DESTINATION ==========================
function t = DESTINATION(e)
    t = Links(e,3);
end

%% ======================================================================
    function print_matrix(A,name,tam)
        lin = size(A,1);
        col = size(A,2);
        for i=1:size(A,3)
```

```
            if nargin == 2
                SS = lin;
            else
                SS = tam(i);
            end
            fprintf(fileID,'===== Matrix %s%d (%d
x %d)=====\n',name,i,SS,col);
            for j=1:SS
                for k=1:col
                    fprintf(fileID,'%4d',A(j,k,i));
                end
                fprintf(fileID,'\n');
            end
        end
        fprintf(fileID,'\n');
    end


%% ========================================================================
    function print_vector(a,name,tam)
        fprintf(fileID,'===== Vector %s =====\n',name);
        for j=1:size(a,2)
            if nargin == 2
                SS = size(a,1);
            else
                SS = tam(j);
            end
            for k=1:SS
                fprintf(fileID,'%5.1f ',a(k,j));
            end
            fprintf(fileID,'\n');
        end
        fprintf(fileID,'\n');
    end
end
```

8. Apêndice 2 (código)

```
function BestFULLFlowMATLAB(fname,ECMP_flag,WSYS_flag)
%
%   Essa função agrupa as matrizes criadas em Gera_variáveris e resolve o
%   problema de Optimal Effectiveness Problem.
%
%    min z = cT.x + 0T.z + 0T.r + 0T.w + 0T.u sujeito a
%        A.x                            = r0
%        R.x                           <=  0
%        B.x + C.z              + D.u <= s0
%                     F.r + H.w + E.u <= t0
%        x,z >= 0  x(1) <= 1
%        r,w >= 1
%        u vetor binário
%
    tic;

    ECMP_flag = 1;
    WSYS_flag = 1;

    fprintf(1,'\n============ Best FULL Flow Program ==============\n\n');
    if ECMP_flag == 0
        fprintf(1,'\tNo ECMP constraints...\n');
    end
    if WSYS_flag == 0
        fprintf(1,'\tNo w-System constraints...\n');
    end

    fname = 'variables.mat';

    S = load(fname);

    nV = S.nV;                  % número de nós
    nE = S.nE;                  % número de links
    nC = nE*nV;
    nX = 1+nC;                  % dimensão da variável x
    nZ = nV*nV;                 % dimensão da variável z
    nR = nV*nV;                 % dimensão da variável r
    nW = nE;                    % dimensão da variável w
    nU = nC;                    % dimensão da variável u
    nQ = 2*nE*(nV-1);           % w-System número total de restrições de
desigualdade
    nT = 3*nE*(nV-1);           % ECMP número total de restrições de
desigualdade
    nY = nX + nZ + nR + nW;

%% ....................................................................
%
%  Matrix A generation
%
    A = zeros(nV*nV,nX);
    lin_inic = 1;
    col_inic = 2;
```

```
    lin_d = nV;
    col_d = nE;
    for k=1:nV
        lin_fim = lin_inic + lin_d - 1;
        col_fim = col_inic + col_d - 1;
        A(lin_inic:lin_fim,col_inic:col_fim) = S.A_mem(:,:,k);
        lin_inic = lin_inic + lin_d;
        col_inic = col_inic + col_d;
    end
%% ......................................................................
%
%  Matrix R generation
%
    R = zeros(nE,nX);
    R(:,1) = S.K_mem;
    col_inic = 2;
    col_d    = nE;
    for k=1:nV
        col_fim = col_inic + col_d - 1;
        R(1:nE,col_inic:col_fim) = S.G_mem(:,:,k);
        col_inic = col_inic + col_d;
    end
%% ......................................................................
%
%  Vector r0 generation  -  VETOR RESTRIÇÕES DE IGUALDADE
%
    r0 = zeros(nV*nV,1);
    lin_inic = 1;
    lin_d = nV;
    for k=1:nV
        lin_fim = lin_inic + lin_d - 1;
        r0(lin_inic:lin_fim,1) = S.a_mem(:,k);
        lin_inic = lin_inic + lin_d;
    end
%% ......................................................................
%
%  Matrix B generation
%
    B = zeros(nT,nX);
    lin_inic = 1;
    col_inic = 2;
    col_d = nE;
    for k=1:nV
        lin_d = 3*S.Delta_Size(k);
        lin_fim = lin_inic + lin_d - 1;
        col_fim = col_inic + col_d - 1;
        B(lin_inic:lin_fim,col_inic:col_fim) = S.B_mem(1:lin_d,:,k);
        lin_inic = lin_inic + lin_d;
        col_inic = col_inic + col_d;
    end
%% ......................................................................
%
%  Matrix C generation
%
```

```
    C = zeros(nT,nZ);
    lin_inic = 1;
    col_inic = 1;
    col_d = nV;

    for k=1:nV
        lin_d = 3*S.Delta_Size(k);
        lin_fim = lin_inic + lin_d - 1;
        col_fim = col_inic + col_d - 1;
        C(lin_inic:lin_fim,col_inic:col_fim) = S.C_mem(1:lin_d,:,k);
        lin_inic = lin_inic + lin_d;
        col_inic = col_inic + col_d;
    end
%% ......................................................................
%
%  Matrix D generation
%
    D = zeros(nT,nU);
    lin_inic = 1;
    col_inic = 1;
    col_d = nE;
    for k=1:nV
        lin_d = 3*S.Delta_Size(k);
        lin_fim = lin_inic + lin_d - 1;
        col_fim = col_inic + col_d - 1;
        D(lin_inic:lin_fim,col_inic:col_fim) = S.D_mem(1:lin_d,:,k);
        lin_inic = lin_inic + lin_d;
        col_inic = col_inic + col_d;
    end
%% ......................................................................
%
%  Vector s0 generation  -  VETOR RESTRIÇÕES DE DESIGUALDADE ECMP
%
    s0 = zeros(nT,1);
    lin_inic = 1;
    for k=1:nV
        lin_d = 3*S.Delta_Size(k);
        lin_fim = lin_inic + lin_d - 1;
        s0(lin_inic:lin_fim,1) = S.b_mem(1:lin_d,k);
        lin_inic = lin_inic + lin_d;
    end
%% ......................................................................
%
%  Matrix E generation
%
    E = zeros(nQ,nU);
    lin_inic = 1;
    col_inic = 1;
    col_d = nE;
    for k=1:nV
        lin_d = 2*S.Delta_Size(k);
        lin_fim = lin_inic + lin_d - 1;
        col_fim = col_inic + col_d - 1;
        E(lin_inic:lin_fim,col_inic:col_fim) = S.E_mem(1:lin_d,:,k);
        lin_inic = lin_inic + lin_d;
        col_inic = col_inic + col_d;
```

```
    end
%% ....................................................................
%
%  Matrix F generation
%
    F = zeros(nQ,nR);
    lin_inic = 1;
    col_inic = 1;
    col_d = nV;
    for k=1:nV
        lin_d = 2*S.Delta_Size(k);
        lin_fim = lin_inic + lin_d - 1;
        col_fim = col_inic + col_d - 1;
        F(lin_inic:lin_fim,col_inic:col_fim) = S.F_mem(1:lin_d,:,k);
        lin_inic = lin_inic + lin_d;
        col_inic = col_inic + col_d;
    end
%% ....................................................................
%
%  Matrix H generation
%
    H = zeros(nQ,nW);
    lin_inic = 1;
    for k=1:nV
        lin_d = 2*S.Delta_Size(k);
        lin_fim = lin_inic + lin_d - 1;
        H(lin_inic:lin_fim,:) = S.H_mem(1:lin_d,:,k);
        lin_inic = lin_inic + lin_d;
    end
%% ....................................................................
%
%  Vector t0 generation  -  VETOR RESTRIÇÕES DE DESIGUALDADE w-System
%
    t0 = zeros(nQ,1);
    lin_inic = 1;
    for k=1:nV
        lin_d = 2*S.Delta_Size(k);
        lin_fim = lin_inic + lin_d - 1;
        t0(lin_inic:lin_fim,1) = S.c_mem(1:lin_d,k);
        lin_inic = lin_inic + lin_d;
    end
%% ....................................................................
%
%  Quantidade de variáveis e restrições
%
    nVAR = nX;
    nRES = nV*nV+nE;

    if ECMP_flag == 1
        nVAR = nVAR + nZ + nU;
        nRES = nRES + nT;
    end
    if WSYS_flag == 1
        nVAR = nVAR + nR + nW + nU;
        nRES = nRES + nQ;
    end
```

```
    fprintf(1,'\tNodes       : %6d\n',nV);
    fprintf(1,'\tLinks       : %6d\n',nE);
    fprintf(1,'\tVariables   : %6d\n',nVAR);
    fprintf(1,'\tConstraints : %6d\n\n',nRES);


%% ......................................................................
%
%  Construção das matrizes como irão ser entregues à linprog
%
    FlwRestr_M = [R zeros(nE,nZ) zeros(nE,nR) zeros(nE,nW) zeros(nE,nU)];
    ECMP_M = [B C zeros(nT,nR) zeros(nT,nW) D];
    w_sys_M = [zeros(nQ,nX) zeros(nQ,nZ) F H E];

    rSS = VECTOR(1-eye(nV));

    f = [1; zeros(nX-1+nZ+nR+nW+nU, 1)];
    Aineq = [FlwRestr_M; ECMP_M; w_sys_M];
    V_ineq = [zeros(nE,1); s0; t0];
    Aeq = [A zeros(nZ,nZ) zeros(nZ,nR) zeros(nZ,nW) zeros(nZ,nU)];
    LB = zeros(nU,1);
    UB = [1;inf*ones(nX-1+nZ,1);100*rSS;inf*ones(nW,1);ones(nU,1)];
%% ......................................................................
%
%  Optimal Effectiveness Problem resolution
%
    [z_min,fval]=linprog(f,Aineq,V_ineq,Aeq,r0,LB,UB);
%% ......................................................................


save('optimal.mat','f','Aineq','V_ineq','Aeq','r0','rSS','Aeq','LB','UB',...

'nV','nE','nC','nX','nZ','nR','nW','nU','nQ','nT','nY','z_min','fval');

    fprintf('\n\tVariável Z otimizada: %.2f\n',fval);
    fprintf('\n\tDuração: %.2f segundos.\n\n',toc);

%% ======================================================================
%   ROTINA DE SUPORTE
%
    function v = VECTOR(x)
        v = reshape(x,size(x,1)*size(x,2),1);
    end
end
```