Utilização de modelador e *solvers* para resolução de problemas de Programação Linear

Eric Hans Messias da Silva
Tribunal de Contas da União - TCU
Brasília, Brasil
https://orcid.org/0000-0003-0743-6627

João Laterza
Banco Central do Brasil - BCB
Brasília, Brasil
https://orcid.org/0000-0002-7698-1517

Marcos Paulo Pereira da Silva
Tribunal de Contas da União - TCU
Brasília, Brasil
https://orcid.org/0000-0003-3204-9661

I. INTRODUÇÃO

A Programação Linear (LP), também conhecida como otimização linear, é uma técnica para resolver problemas de otimização com restrições (injunções) em que a função objetivo é linear em relação às variáveis de decisão, e o domínio destas variáveis é injuncionado por um sistema de inequações lineares [1]. Em outras palavras, trata-se de um método para obter o melhor resultado em um modelo matemático cujos requisitos são representados por relações lineares [2].

A LP encontrou ampla aplicação nos campos da pesquisa operacional, principalmente em problemas de alocação ótima de recursos e otimização combinatória, e, atualmente, tem sido utilizada também no aprendizado de máquina (ML) e na inteligência artificial (AI), em atividades como classificação, previsão estruturada, aprendizagem por reforço inverso e agrupamento e redução de dimensionalidade. Embora os modelos LP muitas vezes pareçam intuitivos na teoria, aplicá-los na prática apresenta um desafio, que envolve estruturar o problema de otimização em uma representação não natural para os usuários, i. e., no formato de função objetivo, variáveis de decisão e restrições [3].

Contudo, uma vez que o problema esteja especificado em um formato apropriado, é possível utilizar ferramentas computacionais - chamadas solucionadores de otimização, ou apenas *solvers* - para solucioná-lo rapidamente [3]. Os *solvers* abrangem algoritmos poderosos que podem resolver modelos de programação matemática, programação de restrições e modelos de escalonamento baseados em restrições, auxiliando a tomada de decisões em atividades de planejamento e alocação de recursos [4]. Existem diversos *solvers* disponíveis, tais como CPLEX, GUROBI, Mosek, Xpress, etc., que fornecem soluções ótimas para os gestores e tomadores de decisão [3].

Atualmente, tem crescido, no mercado, a necessidade de se planejar e tomar ótimas decisões regularmente, de modo inerente às operações diárias das organizações. Avanços computacionais recentes forneceram a infraestrutura para incorporar modelos de otimização em soluções de *software* analíticas. Consequentemente, os profissionais de pesquisa operacional precisam projetar, modelar e implementar mecanismos computacionais robustos baseados em modelos de programação linear, recorrendo, para isso, a linguagens de programação como C ++, Java e Python [3]. Nesse contexto ressalta-se o PuLP, um modelador de LP escrito em Python, que chama *solvers* como o GLPK, o CPLEX e o GUROBI para resolver problemas lineares [5].

PuLP é uma biblioteca de modelagem de alto nível que permite a descrição de programas matemáticos na linguagem de programação de computadores Python, alavancando seu poder e permitindo ao usuário criar programas usando expressões que são naturais a ela, evitando, sempre que possível, sintaxe especial e palavras-chave. Dessa maneira, o PuLP adota uma abordagem modular com *solvers*, lidando com a conversão de expressões Python-PuLP internamente e, em seguida, expondo os dados a uma classe de interface do *solver*. Diversos solucionadores de otimização podem ser empregados, tanto comerciais (como o CPLEX e o GUROBI) quanto de código aberto (GLPK) [6].

A presente pesquisa selecionou três problemas apresentados em [7], formulou-os como problemas de otimização linear, estabelecendo um modelo matemático em termos de uma função objetivo, variáveis de decisão e restrições, e solucionou-os aplicando métodos de três *solvers* distintos: CPLEX (v. 20.1.0.0), GUROBI (v. 9.1.2), que estão entre os principais *solvers* comerciais, e GLPK (v. 4.65), um dos *solvers* de código aberto (*software livre*) mais popular atualmente [8]. Para isso foi utilizada a linguagem de programação Python (v.3.7.1) com a biblioteca de modelagem PuLP (v2.4).

II. METODOLOGIA

A. Métodos

Existem diversos métodos para se resolver um problema de LP. Os mais comuns são o Primal Simplex, o Dual Simplex e o de Pontos Interiores, também conhecido como Barrier [9]. Outros algoritmos de otimização, visando desempenhos melhores na resolução de problemas específicos, têm sido implementados, a exemplo dos otimizadores Network e Sifting [10]. Os métodos para resolução de problemas de LP disponíveis nos *solvers* selecionados nesta pesquisa são [10][11][12]:

1) GLPK: Primal Simplex, Dual Simplex, Interior-Point.

2) CPLEX: Primal Simplex, Dual Simplex, Interior-Point. Network e Sifting.

3) GUROBI: Primal Simplex, Dual Simplex e Interior-Point.

Frequentemente os *solvers* são configurados para, na ausência de comandos de seleção do método, decidir aquele que melhor se enquadra no problema a ser resolvido de acordo com um critério pré-estabelecido. No CPLEX e no GUROBI, é avaliado se a execução do processo ocorre de forma serial ou em paralelo, podendo, dependendo do caso, serem executados vários otimizadores em múltiplas *threads* simultaneamente, e escolhido aquele que termina primeiro [10][12]. Predominantemente é selecionado o Dual Simplex para resolução de problemas de LP. O GLPK, por sua vez, não

possui uma rotina de seleção automática, adotando o Primal Simplex como padrão [11]. Reitera-se que é possível ao usuário selecionar manualmente o método de seu interesse, passando parâmetros específicos para cada um dos *solvers*. Na abordagem modular do PuLP, esse comando é padronizado em um parâmetro de opções, facilitando a interação com os otimizadores por meio da linguagem Python.

B. Pré-processamento

Muitos *solvers* dispõem ainda de uma etapa de pré-processamento, buscando-se simplificar as restrições, reduzir o tamanho do problema e eliminar eventuais redundâncias, possibilitando, assim, uma resolução mais rápida. Nesse contexto, são empregadas técnicas como a Pré-solução, em que a entrada do problema pelo usuário é examinada em busca de oportunidades lógicas de redução, e a Agregação, que tenta eliminar variáveis e linhas por meio de substituição [10][13]. Na presente pesquisa, os *solvers* selecionados apresentam as seguintes configurações [10][11][12]:

1) GLPK: Pré-solução disponível para problemas de LP, desativado por padrão. Não conta com técnicas de agregação no pré-processamento.

2) CPLEX: Pré-solução e Agregação ativadas por padrão.

3) GUROBI: Pré-solução e Agregação ativadas por padrão.

É também na etapa de pré-processamento que ocorre a relaxação de programação linear, utilizada em problemas em que uma ou mais variáveis de decisão assumem apenas valores inteiros. Essa técnica, disponível em todos os *solvers* selecionados nessa pesquisa, consiste em relaxar o problema original, eliminando as condições de integralidade, e resolver o problema de programação linear contínua resultante sobre o sistema de restrição, por meio de métodos como os descritos neste artigo. Caso a solução do problema de programação linear relaxada satisfaça as restrições de integralidade, a solução obtida é ótima. Do contrário, ou seja, se a LP é inviável, a programação inteira é também inviável [13]. Nesse caso, existem outros métodos que podem ser usados, específicos para problemas de Programação Linear Inteira (ILP) e Programação Linear Inteira-Mista (MILP), que, contudo, não serão objeto de estudo desta pesquisa.

III. DESENVOLVIMENTO

Foram escolhidos de [7] três problemas de LP para serem resolvidos utilizando o modelador PuLP e os solvers GLPK, CPLEX e GUROBI, apresentados a seguir, junto do modelo matemático formulado pelos autores em cada um deles.

A. Problema 1

"A Empresa de Manufatura Ômega descontinuou a produção de determinada linha de produtos não lucrativa. Esse fato acabou criando considerável excesso de capacidade produtiva. A direção está levando em conta a possibilidade de dedicar esse excesso de capacidade produtiva para um ou mais produtos. Vamos chamá-los produtos 1, 2 e 3. A capacidade disponível nas máquinas que poderiam limitar a produção encontra-se resumida na tabela a seguir:" [7, pp. 72-73]

TABELA I
CAPACIDADE DISPONÍVEL DAS MÁQUINAS [7, PP. 72-73]

Tipo de máquina	Tempo disponível (horas/máquina por semana)
Fresadora	500
Torno	350
Retificadora	150

"O número de horas/máquina exigidas para cada unidade do respectivo produto é:" [7, pp. 72-73]

TABELA II
COEFICIENTE DE PRODUTIVIDADE DAS MÁQUINAS [7, PP. 72-73]

Tipo de máquina	Produto 1	Produto 2	Produto 3
Fresadora	9	3	5
Torno	5	4	0
Retificadora	3	0	2

"O departamento de vendas sinaliza que o potencial de vendas para os produtos 1 e 2 excede a taxa de produção máxima e que o potencial de vendas para o produto 3 é de 20 unidades por semana. O lucro unitário seria, respectivamente, de US$ 50, 20 e 25 para os produtos 1, 2 e 3. O objetivo é determinar quanto de cada produto a Ômega deveria produzir para maximizar os lucros." [7, pp. 72-73]

O problema pode ser formulado da seguinte forma:

1) Função objetivo (Z): $50x_1 + 20x_2 + 25x_3$, sendo as variáveis de decisão x_1, x_2 e x_3 referentes à quantidade - em unidades - dos produtos a ser produzida, e seus coeficientes relativos ao respectivos lucros unitários.

2) Conjunto viável (Ω): (i) Restrição de produtividade da fresadora (r_1): $9x_1 + 3x_2 + 5x_3 \leqslant 500$; (ii) Restrição de produtividade do torno (r_2): $5x_1 + 4x_2 + 0x_3 \leqslant 350$; Restrição de produtividade da retificadora (r_3): $3x_1 + 0x_2 + 2x_3 \leqslant 150$; Restrição de potencial de vendas (r_4): $x_3 \leqslant 20$.

Considera-se, portanto, um problema de otimização linear visando a maximização dos lucros, estabelecendo-se, para as variáveis de decisão, condições de integralidade, visto não ser possível produzir produtos parciais. Assim sendo, modela-se o problema em linguagem Python com auxílio da biblioteca PuLP, criando o modelo por meio da função *LpProblem* e iniciando as variáveis de decisão x_1, x_2 e x_3 com a classe *LpVariable*. Por fim, são acrescidas ao modelo as restrições r_1, r_2, r_3 e r_4 e a função objetivo *Z*. A Figura 1 ilustra o código utilizado para a modelagem do Problema 1.

```
# Criando o modelo
model = LpProblem(name = 'PROBLEMA_1', sense = LpMaximize)

# Iniciando as variáveis de decisão
cat = 'Integer'
x1 = LpVariable(name = 'x1', lowBound=0, cat = cat)
x2 = LpVariable(name = 'x2', lowBound=0, cat = cat)
x3 = LpVariable(name = 'x3', lowBound=0, cat = cat)

# Adicionando as restrições ao modelo
model += (9*x1 + 3*x2 + 5*x3 <= 500, 'r1')
model += (5*x1 + 4*x2 + 0*x3 <= 350, 'r2')
model += (3*x1 + 0*x2 + 2*x3 <= 150, 'r3')
model += (x3 <= 20, 'r4')

# Adicionando a função objetivo ao modelo
model += 50*x1 + 20*x2 + 25*x3
```

Fig. 1. Código para modelagem do Problema 1

B. Problema 2

"A empresa Medequip produz equipamentos de diagnóstico médico de precisão em duas fábricas. As clínicas médicas fizeram pedidos para a produção deste mês. A tabela abaixo mostra qual seria o custo para despachar cada unidade de equipamento de cada fábrica para cada um desses clientes. Também é indicado o número de unidades que será produzido em cada fábrica, bem como o número de unidades destinado a cada cliente." [7, p. 76]

TABELA III
CUSTO DE REMESSA POR UNIDADE [7, P. 76]

De / Para	Custo de remessa por unidade			Produção
	Cliente 1	Cliente 2	Cliente 3	
Fábrica 1	US$ 600	US$ 800	US$ 700	400 unidades
Fábrica 2	US$ 400	US$ 900	US$ 600	500 unidades
Tamanho do pedido	300 unidades	200 unidades	400 unidades	

"Agora, é necessário tomar uma decisão em relação ao plano de remessa da mercadoria, ou seja, quantas unidades de cada fabrica para cada cliente." [7, p. 76]

O problema pode ser formulado da seguinte forma:

1) Função objetivo (Z): $600x_1 + 800x_2 + 700x_3 + 400x_4 + 900x_5 + 600x_6$, sendo as variáveis de decisão referentes à quantidade de equipamentos - em unidades - a ser enviada de cada uma das fábricas para cada um dos cliente, com os coeficientes representando os respectivos custos das remessas.

2) Conjunto viável (Ω): (i) Restrição de produtividade da Fábrica 1 (r_1): $x_1 + x_2 + x_3 = 400$; (ii) Restrição de produtividade do Fábrica 2 (r_2): $x_4 + x_5 + x_6 = 500$; (iii) Restrição pedidos do Cliente 1 (r_3): $x_1 + x_4 = 300$; (iv) Restrição pedidos do Cliente 2 (r_4): $x_2 + x_5 = 200$; (v) Restrição pedidos do Cliente 3 (r_5): $x_3 + x_6 = 400$.

Como a empresa deve atender a todos os pedidos dos clientes e a quantidade de produção é igual a quantidade demandada, as restrições adotadas são de igualdade. Novamente as variáveis de decisão devem ser inteiras, uma vez que não é possível remeter equipamentos parciais. Assim, formula-se um problema de otimização linear visando a minimização dos custos das remessas. A modelagem é então realizada da mesma forma descrita no Problema 1. A Figura 2 mostra o código utilizado para a modelagem do Problema 2.

```
# Criando o modelo
model = LpProblem(name = 'PROBLEMA_2', sense = LpMinimize)

# Iniciando as variáveis de decisão
cat = 'Integer'
x1 = LpVariable(name = 'x1', lowBound=0, cat = cat)
x2 = LpVariable(name = 'x2', lowBound=0, cat = cat)
x3 = LpVariable(name = 'x3', lowBound=0, cat = cat)
x4 = LpVariable(name = 'x4', lowBound=0, cat = cat)
x5 = LpVariable(name = 'x5', lowBound=0, cat = cat)
x6 = LpVariable(name = 'x6', lowBound=0, cat = cat)

# Adicionando as restrições ao modelo
model += (x1 + x2 + x3 == 400, 'r1')
model += (x4 + x5 + x6 == 500, 'r2')
model += (x1 + x4 == 300, 'r3')
model += (x2 + x5 == 200, 'r4')
model += (x3 + x6 == 400, 'r5')

# Adicionando a função objetivo ao modelo
model += 600*x1 + 800*x2 + 700*x3 + 400*x4 + 900*x5 + 600*x6
```

Fig. 2. Código para modelagem do Problema 2

C. Problema 3

"Universidade de Oxbridge mantém um poderoso mainframe para fins de pesquisa utilizado pelo seu corpo docente, alunos dos cursos de Ph.D. e pesquisadores associados. Durante todo o período de funcionamento, é preciso ter um funcionário disponível para operar e fazer a manutenção do computador, bem como realizar alguns serviços de programação. Beryl Ingram, a diretora desse centro de computação, supervisiona a operação" [7, p. 77].

"Agora é o inicio do semestre letivo, e Beryl está se deparando com a questão de fazer a escala de seus diversos operadores. Pelo fato de todos eles estarem atualmente matriculados na universidade, estão disponíveis para trabalhar somente durante um período limitado em cada dia da semana, conforme mostra a tabela a seguir:" [7, p. 77]

TABELA IV
NUMERO MÁXIMO DE HORAS DE DISPONIBILIDADE [7, P. 77]

Operadores	Salário/hora	Número máximo de horas de disponibilidade				
		Seg.	Ter.	Qua.	Qui.	Sex.
K. C.	US$ 25/hora	6	0	6	0	6
D. H.	US$ 26/hora	0	6	0	6	0
H. B.	US$ 24/hora	4	8	4	0	4
S. C.	US$ 23/hora	5	5	5	0	5
K. S.	US$ 28/hora	3	0	3	8	0
N. K.	US$ 30/hora	0	0	0	6	2

"Há seis operadores (há dois estudantes graduados e quatro ainda no curso). Todos eles têm salários diferentes em razão da experiência diversa com computadores e em termos de habilidade de programação. A tabela anterior mostra os salários junto com o número máximo de horas que cada deles um trabalha por dia" [7, p. 77].

"Cada operador tem a garantia de certo número de horas por semana, que fará com que eles mantenham o conhecimento adequado sobre a operação. O nível é estabelecido arbitrariamente em oito horas por semana para os estudantes não formados (K. C., D. H., H. B. e S.C.) e sete horas por semana para os estudantes graduados (K. S. e N. K.)" [7, p. 77].

"O centro de computação deve permanecer aberto para operação das 8h até as 22h, de segunda-feira à sexta-feira, com exatamente um operador de plantão durante esse período. Aos sábados e domingos o computador deve ser operado por outra equipe" [7, p. 77].

"Devido ao orçamento apertado, Beryl tem de minimizar o custo. Ela quer determinar o número de horas que deve atribuir a cada operador em cada dia da semana" [7, p. 77].

O problema pode ser formulado da seguinte forma:

1) Função objetivo (Z): $25 (x_1 + x_2 + x_3 + x_4 + x_5) + 26 (x_6 + x_7 + x_8 + x_9 + x_{10}) + 24 (x_{11} + x_{12} + x_{13} + x_{14} + x_{15}) + 23 (x_{16} + x_{17} + x_{18} + x_{19} + x_{20}) + 28 (x_{21} + x_{22} + x_{23} + x_{24} + x_{25}) + 30 (x_{26} + x_{27} + x_{28} + x_{29} + x_{30})$, sendo as variáveis de decisão referentes à disponibilidade de cada operador em cada dia da semana e os coeficientes ao respectivos salários/hora.

2) Conjunto viável (Ω): Para cara operador foram criadas cinco restrições referentes à sua disponibilidade em cada dia da semana, e mais uma referente ao mínimo de horas para manter o conhecimento adequado sobre a operação. Adicionalmente, foram estabelecidas restrições para garantir que em cada dia da semana fossem cobertas 14 horas de operação no centro de computação. Consequentemente, foram impostas, ao final, 41 restrições.

Para esse problema não há necessidade de estabelecer condições de integralidade, sendo possível a um operador trabalhar em turnos fracionados de horas. Logo, tem-se um problema de otimização linear visando a minimização dos custos com salário de operadores, como descrito no esquema da Figura 3, e cuja modelagem pode ser efetuada como ilustrado nas Figuras 4 e 5.

Resolução:

Função objetivo (**Z**):

$$25(x_1+x_2+x_3+x_4+x_5)+26(x_6+x_7+x_8+x_9+x_{10})+24(x_{11}+x_{12}+x_{13}+x_{14}+x_{15})+$$
$$23(x_{16}+x_{17}+x_{18}+x_{19}+x_{20})+28(x_{21}+x_{22}+x_{23}+x_{24}+x_{25})+30(x_{26}+x_{27}+x_{28}+x_{29}+x_{30})$$

Conjunto viável:

Restrição de K.C.:
- (KC_1): $x_1 \leq 6$ (agenda)
- (KC_2): $x_2 = 0$ (agenda)
- (KC_3): $x_3 \leq 6$ (agenda)
- (KC_4): $x_4 = 0$ (agenda)
- (KC_5): $x_5 \leq 6$ (agenda)
- (KC_6): $x_1+x_2+x_3+x_4+x_5 \geq 8$ (nv mín)

Restrição de D.H.:
- (DH_1): $x_6 = 0$ (agenda)
- (DH_2): $x_7 \leq 6$ (agenda)
- (DH_3): $x_8 = 0$ (agenda)
- (DH_4): $x_9 \leq 6$ (agenda)
- (DH_5): $x_{10} = 0$ (agenda)
- (DH_6): $x_6+x_7+x_8+x_9+x_{10} \geq 8$ (nv mín)

Restrição de H.B.:
- (HB_1): $x_{11} \leq 4$ (agenda)
- (HB_2): $x_{12} \leq 8$ (agenda)
- (HB_3): $x_{13} \leq 4$ (agenda)
- (HB_4): $x_{14} = 0$ (agenda)
- (HB_5): $x_{15} \leq 4$ (agenda)
- (HB_6): $x_{11}+x_{12}+x_{13}+x_{14}+x_{15} \geq 8$ (nv mín)

Restrição de S.C.:
- (SC_1): $x_{16} \leq 5$ (agenda)
- (SC_2): $x_{17} \leq 5$ (agenda)
- (SC_3): $x_{18} \leq 5$ (agenda)
- (SC_4): $x_{19} = 0$ (agenda)
- (SC_5): $x_{20} \leq 5$ (agenda)
- (SC_6): $x_{16}+x_{17}+x_{18}+x_{19}+x_{20} \geq 8$ (nv mín)

Restrição de K.S.:
- (KS_1): $x_{21} \leq 3$ (agenda)
- (KS_2): $x_{22} = 0$ (agenda)
- (KS_3): $x_{23} \leq 3$ (agenda)
- (KS_4): $x_{24} \leq 8$ (agenda)
- (KS_5): $x_{25} = 0$ (agenda)
- (KS_6): $x_{21}+x_{22}+x_{23}+x_{24}+x_{25} \geq 7$ (nv mín)

Restrição de N.K.:
- (NK_1): $x_{26} = 0$ (agenda)
- (NK_2): $x_{27} = 0$ (agenda)
- (NK_3): $x_{28} = 0$ (agenda)
- (NK_4): $x_{29} \leq 6$ (agenda)
- (NK_5): $x_{30} \leq 2$ (agenda)
- (NK_6): $x_{26}+x_{27}+x_{28}+x_{29}+x_{30} \geq 7$ (nv mín)

Cota diária:
- (D_1): $x_1+x_6+x_{11}+x_{16}+x_{21}+x_{26} = 14$ (seg)
- (D_2): $x_2+x_7+x_{12}+x_{17}+x_{22}+x_{27} = 14$ (ter)
- (D_3): $x_3+x_8+x_{13}+x_{18}+x_{23}+x_{28} = 14$ (qua)
- (D_4): $x_4+x_9+x_{14}+x_{19}+x_{24}+x_{29} = 14$ (qui)
- (D_5): $x_5+x_{10}+x_{15}+x_{20}+x_{25}+x_{30} = 14$ (sex)

Fig. 3. Modelo matemático formulado para o Problema 3

```
# Criando lista com o nome dos operadores
oprs = ['KC', 'DH', 'HB', 'SC', 'KS', 'NK']

# Criando lista de listas contendo o número máximo de horas
# de disponibilidade por operador por dia de semana
hmax = [[6, 0, 6, 0, 6],
        [0, 6, 0, 6, 0],
        [4, 8, 4, 0, 4],
        [5, 5, 5, 0, 5],
        [3, 0, 3, 8, 0],
        [0, 0, 0, 6, 2]]

# Criando lista com o mínimo de horas de cada operador para
# manter o conhecimento adequado sobre a operação
hmin = [8, 8, 8, 8, 7, 7]

# Criando lista com os salários/hora dos operadores
slrs = [25, 26, 24, 23, 28, 30]
```

Fig. 4. Código para modelagem do Problema 3 - parte 1

```
# Criando o modelo
model = LpProblem(name = 'PROBLEMA_3', sense = LpMinimize)

# Iniciando as variáveis de decisão
cat = 'Continuous'
X = [LpVariable(name='x{0}'.format(i+1),lowBound=0,cat=cat) for i in range(30)]

# Adicionando as restrições e a função objetivo ao modelo
def rst_agg(rst,lst,hrs):
    return [(rst+str(i+1),j) for i, j in enumerate(lst+[hrs])]
d = 0
f_obj = []
for e, f, g, h in zip(oprs, hmax, hmin, slrs):
    tpl = rst_agg(e, f, g)
    for i, j in zip(X[d:d+5], tpl[:-1]):
        if j[1] == 0:
            model += (i == j[1], j[0])
        else:
            model += (i <= j[1], j[0])
    f_obj.append(h*lpSum(X[d:d+5]))
    model += (lpSum(X[d:d+5]) >= tpl[-1][1], tpl[-1][0])
    d += 5
for i in range(5):
    model += (lpSum([X[j] for j in range(i, 30, 5)]) == 14, 'D{0}'.format(i+1))
model += lpSum(f_obj)
```

Fig. 5. Código para modelagem do Problema 3 - parte 2

IV. Resultados

Uma vez modelado os problemas, é executada a função *solve,* passando-se como parâmetro o *solver* desejado. Como resultado, o problema é ajustado para o formato de entrada do otimizador escolhido e a resolução encontrada é capturada pela função interna *actualSolve*, passando a incorporar o objeto *LpProblem.* É também possível passar outros parâmetros de configuração, caso o usuário deseje, por exemplo, imprimir os resultados obtidos em arquivos específicos, selecionar o método a ser utilizado ou fixar um tempo limite para o *solver* retornar a solução. O rol de opções disponíveis vai depender do software de otimização selecionado.

Nesta pesquisa os problemas foram resolvidos com as configurações padrão de cada *solver*. Consequentemente, foram realizadas atividades de pré-processamento, com técnicas de Pré-solução e Agregação (esta última não contemplada no GLPK) e, nos dois primeiros problemas, modelados com condições de integralidade, técnicas de Relaxação. Os métodos de LP finais empregados para obtenção das soluções ótimas, em todos os problemas, foram o Primal e o Dual Simplex. Os resultados são apresentados a seguir.

A. Problema 1

Para maximizar seus lucros, a Empresa de Manufatura Ômega deveria produzir semanalmente 26, 55 e 20 unidades dos produtos 1, 2 e 3, respectivamente, obtendo, dessa maneira, US$ 2.900,00 por semana.

$$Z = 50(26) + 20(55) + 25(20) = 2.900$$

$$r_1: 9(26) + 3(55) + 5(20) = 499 \leqslant 500$$

$$r_2: 5(26) + 4(55) + 0(20) = 350 \leqslant 350$$

$$r_3: 3(26) + 0(55) + 2(20) = 118 \leqslant 150$$

$$r_4: 0(26) + 0(55) + 1(20) = 20 \leqslant 20$$

Observa-se que, ao relaxar o problema original, a solução ótima encontrada não foi inteira, sendo, portanto, necessária a aplicação de métodos alternativos, como *Branch and Bound* e algoritmos heurísticos, que, contudo, não foram objetos de

estudo nesta pesquisa, visto não ter abrangido técnicas para resolução de problemas de ILP e MILP. Não obstante, constatou-se que, no caso concreto, o resultado arredondado do problema de LP relaxado coincidiu com a solução inteira ótima do problema, como se observa nas Figuras 6, 7 e 8, que contém os resultados obtidos pelos *solvers*.

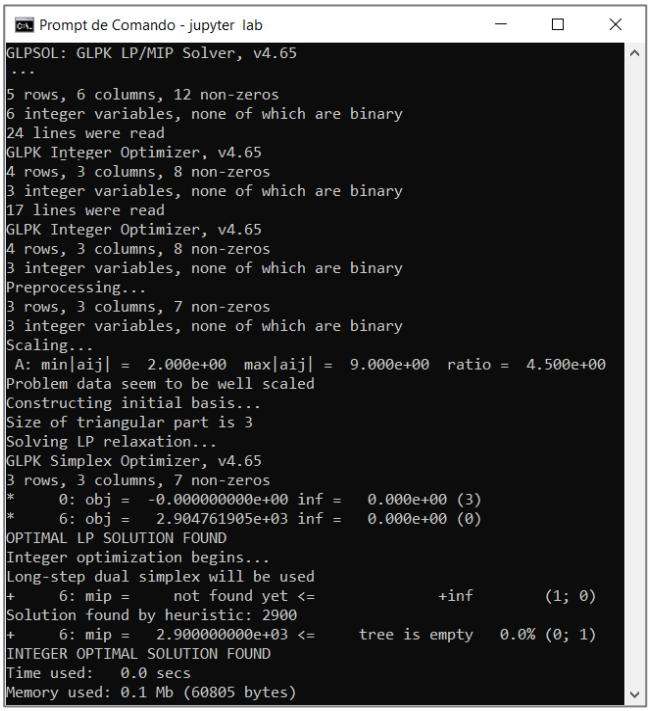

Fig. 6. Resolução do Problema 1 pelo *solver* GLPK

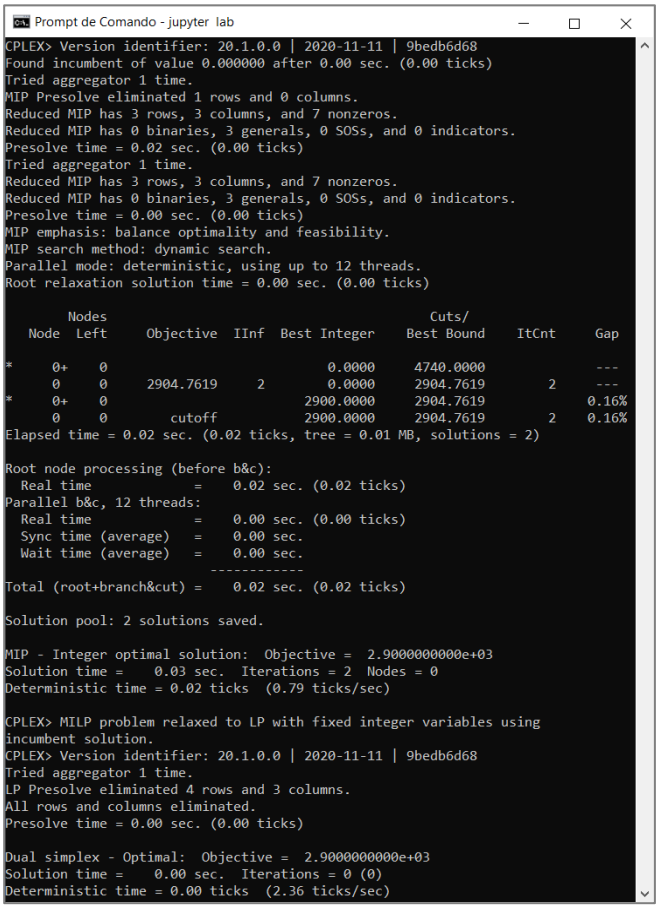

Fig. 7. Resolução do Problema 1 pelo *solver* CPLEX

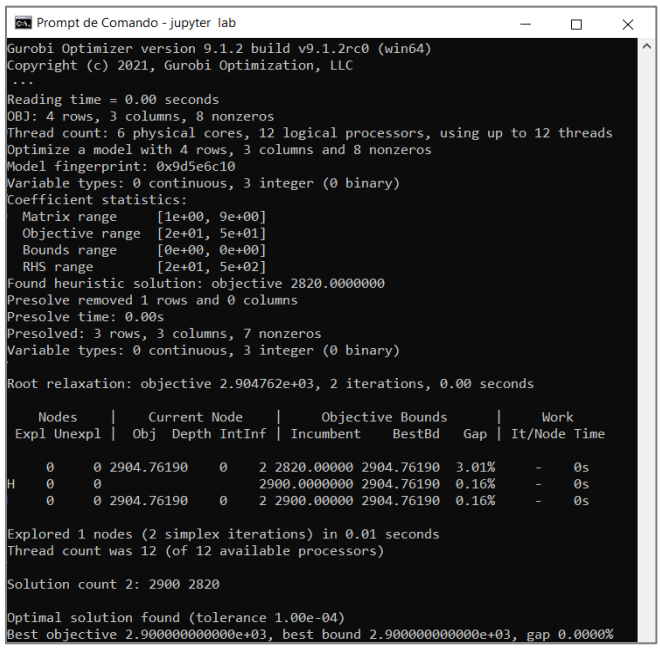

Fig. 8. Resolução do Problema 1 pelo *solver* GUROBI

B. Problema 2

A empresa Medequip, visando minimizar seus custos, deveria fazer as seguintes remessas: (i) 300 unidades da fábrica 2 para o cliente 1; (ii) 200 unidades da fábrica 1 para o cliente 2; (iii) 200 unidades da fábrica 1 e 200 unidades da fábrica 2 para o cliente 3. Dessa forma, o custo total seria de apenas US$ 540.000,00.

$$Z = 800(200) + 700(200) + 400(300) + 600(200) = 540.000$$

r_1: 0 + 200 + 200 = 400 r_2: 300 + 0 + 200 = 500

r_3: 0 + 300 = 300 r_4: 200 + 0 = 200

r_5: 200 + 200 = 400

A eliminação das condições de integralidade das variáveis de decisão resultou em um problema LP cuja solução é inteira, de modo que a aplicação dos métodos Primal e Dual Simplex após a técnica de relaxação já foi suficiente para resolver o problema original, como se observa nos resultados dos *solvers* apresentados nas Figuras 9, 10 e 11.

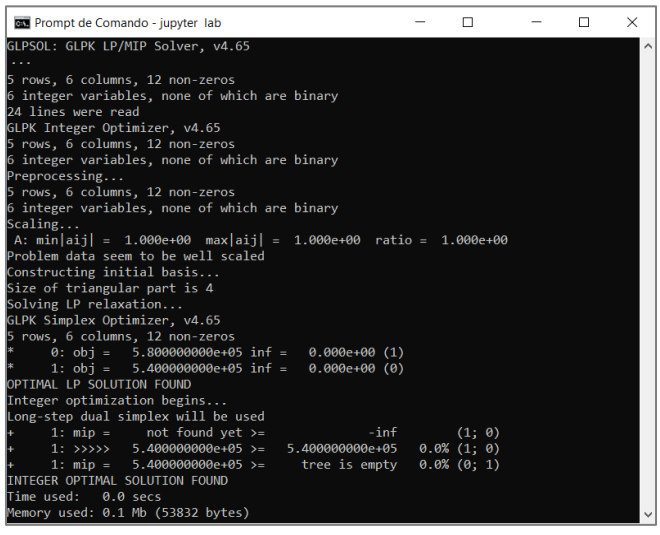

Fig. 9. Resolução do Problema 2 pelo *solver* GLPK

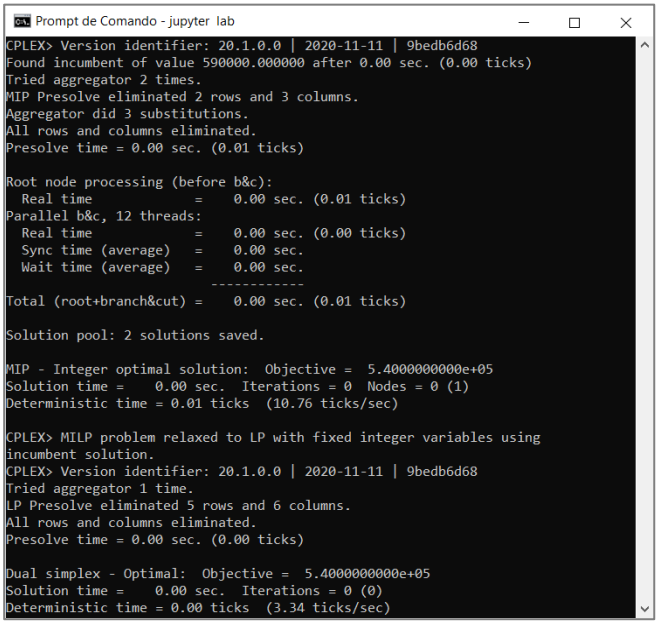

Fig. 10. Resolução do Problema 2 pelo *solver* CPLEX

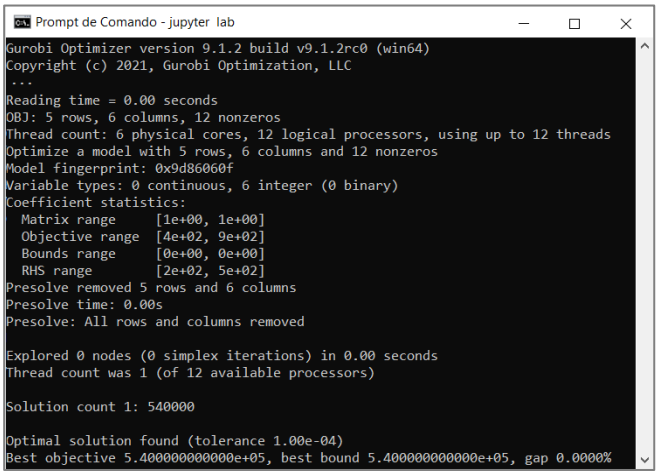

Fig. 11. Resolução do Problema 2 pelo *solver* GUROBI

C. Problema 3

Para minimizar o custo semanal com salário de operadores, deveria ser adotada uma das escalas a seguir, existindo mais de uma combinação possível resultando na solução ótima do problema. Em todo o caso, o menor custo seria de US$ 1.755, considerando os valores de salário/hora dos operadores.

TABELA V
ALTERNATIVA A PARA ESCALA DE OPERADORES

	Seg.	Ter.	Qua.	Qui.	Sex.	
K.C.	2	0	3	0	4	= 9
D.H.	0	2	0	6	0	= 8
H.B.	4	7	4	0	4	= 19
S.C.	5	5	5	0	5	= 20
K.S.	3	0	2	2	0	= 7
N.K.	0	0	0	6	1	= 7
	= 14	= 14	= 14	= 14	= 14	

TABELA VI
ALTERNATIVA B PARA ESCALA DE OPERADORES

	Seg.	Ter.	Qua.	Qui.	Sex.	
K.C.	4	0	2	0	3	= 9
D.H.	0	2	0	6	0	= 8
H.B.	4	7	4	0	4	= 19
S.C.	5	5	5	0	5	= 20
K.S.	1	0	3	3	0	= 7
N.K.	0	0	0	5	2	= 7
	= 14	= 14	= 14	= 14	= 14	

Observa-se que em ambas as escalas foi respeitada a disponibilidade de horas dos operadores em cada dia da semana, e mantido o nível mínimo estabelecido para a manutenção do conhecimento adequado sobre a operação. Ademais, todos os dias contam com 14 horas escaladas, de modo que todas as restrições impostas foram atendidas. As figuras 12, 13 e 14 apresentam a resolução dos *solvers*.

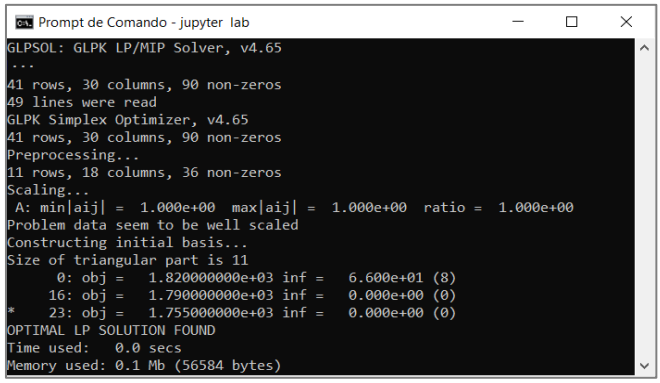

Fig. 12. Resolução do Problema 3 pelo *solver* GLPK

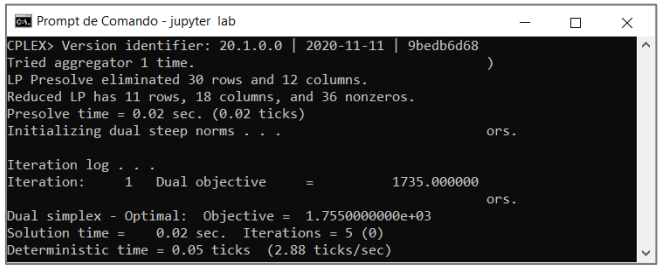

Fig. 13. Resolução do Problema 3 pelo *solver* CPLEX

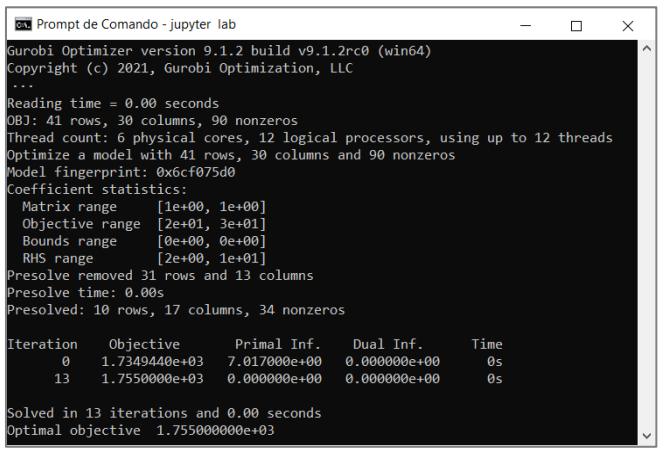

Fig. 14. Resolução do Problema 3 pelo *solver* CPLEX

Como as técnicas de pré-processamento, bem como os critérios de iteração adotados no método *Simplex*, podem variar de *solvers* para *solver*, é possível que as variáveis de decisão assumam valores divergentes nas soluções encontradas. No Problema 3, os otimizadores GLPK e CPLEX chegaram na escala descrita na Tabela V, ao passo que a solução do GUROBI resultou na escala da Tabela VI. De toda forma, o valor da função objetivo (1.755) não muda, já que o conjunto de soluções ótimas para um LP é um conjunto convexo.

V. Conclusão

Atualmente não existe um *solver* com melhor desempenho em todos os tipos de problemas. A adequação de um solucionador de otimização depende da natureza do problema e do tempo de computação [14]. Contudo, para problemas mais complexos, os *solvers* comerciais superam os de código aberto. Assim, o CPLEX pode ser de 10 a 100 vezes mais rápido para a resolução de problemas com o método Dual Simplex do que o GLPK [11]. Ainda, realizando testes em *benchmarks* independentes mantidos por Hans Mittelmann, o GUROBI apresentou rápidas resoluções, enquanto o GLPK não conseguir chegar a um resultado [12]. As Figuras 15 e 16 apresentam comparações estatísticas entre esses três *solvers*, com base em testes efetuados com instâncias de problemas do mesmo *benchmark*, usando um computador com processador Intel Xeon X5680 com 2*6 núcleos, 32 GB de RAM e sistema operacional linux 64bit [15].

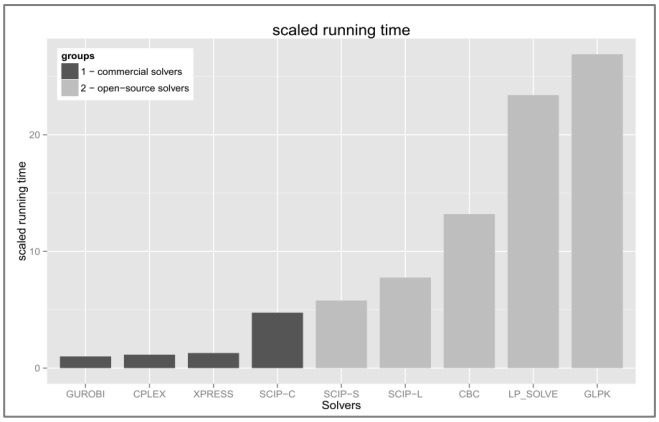

Fig. 14. Tempos de execução dos *solvers* [15]

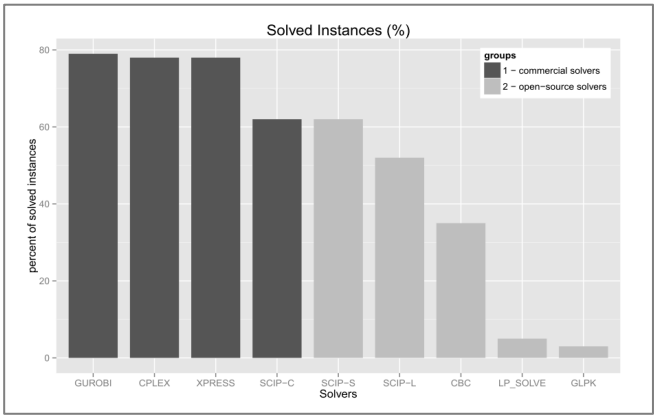

Fig. 15. Percentual de problemas resolvidos por *solvers* [15]

Pequenos problemas, por outro lado, podem ser resolvidos usando qualquer solucionador indiscriminadamente, uma vez que, nesses casos, o tempo necessário para resolver o problema é sempre satisfatório [15].

Os problemas selecionados nesta pesquisa foram de baixa complexidade, de modo que todos os três *solvers* chegaram à solução ótima em tempo mínimo. Dessa forma, todos os solucionadores se mostraram efetivos para as demandas em questão. Ademais, a utilização da linguagem Python facilitou a modelagem dos problemas, possibilitando sua formulação com poucas linhas de código. Por fim, a biblioteca PuLP propiciou uma comunicação direta com os *solvers* sem a necessidade de inserção de sintaxes especiais e palavras-chave na interface de cada um dos solucionadores. Os códigos empregados podem ser acessados no endereço eletrônico https://github.com/laterza22unb/PPCA2020-02-FPO. Como resultado, conclui-se que os profissionais de pesquisa operacional, por meio das ferramentas apresentadas, podem, em suas atividades diárias, facilmente projetar, modelar e implementar mecanismos computacionais robustos baseados em modelos de programação linear, sendo necessário apenas o conhecimento mínimo acerca do problema, de modo a modelá-lo adequadamente e selecionar o solver que melhor se adequa a ele.

Referências

[1] F. Nogueira, "Programação Linear", *ufjf.br*, p. 1, Jun, 2010. [Online]. Available on: https://www.ufjf.br/epd015/files/2010/06/IntrodPL.pdf [Accessed May 11, 2021].

[2] "Linear programming," *Wikipedia*. Acessed on: May 11, 2021. [Online] Available on: https://en.wikipedia.org/wiki/Linear_programming.

[3] K. Kersting, M. Mladenov, and P. Tokmakov, "Relational linear programming", *Artificial Intelligence*, vol. 244, pp. 188-216, March 2017.

[4] International Business Machines Corporation, "Optimization solvers for faster answers," IBM, 590 Madison Avenue, New York, NY, USA, 2015. Accessed on: May 5, 2021. [Online]. Available on: https://www.ibm.com/analytics/optimization-solver.

[5] Computational Infrastructure for Operations Research Foundation, "PuLP", COIN-OR. Accessed on: May 11, 2021. [Online]. Available on: https://projects.coin-or.org/PuLP/wiki/WikiStart.

[6] S. Mitchell, M. O'Sullivan, and I. Dunning, *PuLP: A Linear Programming Toolkit for Python*. Department of Engineering Science, The University of Auckland, Auckland, New Zealand, 2011.

[7] F. S. Hillier, G. J. and Lieberman, *Introdução à Pesquisa Operacional*, 9ª ed. Porto Alegre: AMGH, 2013.

[8] E. Oki, *Linear Programming and Algorithms for Communication Networks: A Practical Guide to Network Design, Control, and Management*. Boca Raton, FL: CRC Press, 2012.

[9] PuLP Documentation Team, *PuLP Documentation Release 1.4.6*, IBM PuLP, 2010. Acessed on: May 15, 2021. [Online] Available on: https://projects.coin-or.org/PuLP/export/353/trunk/doc/pulp.pdf

[10] International Business Machines Corporation, *Optimization Studio CPLEX User's Manual*, ILOG CPLEX, 2009. [Online] Available on: https://www.ibm.com/docs/en/SSSA5P_12.8.0/ilog.odms.studio.help/pdf/usrcplex.pdf. [Accessed on May 15, 2021].

[11] GNU Linear Programming Kit, *GNU Linear Programming Kit Reference Manual*, GLPK project, DRAFT, 2009. [Online] Available on: https://www.gnu.org/software/glpk/#documentation. [Accessed on May 15, 2021].

[12] Gurobi Optimization, *Gurobi Optimizer Reference Manual Version 9.1*, Gurobi Optimization, LLC, 2021. Available on: https://www.gurobi.com/documentation/9.1/refman/index.html. [Accessed on May 15, 2021].

[13] K. Genova and V. Guliashki. "Linear integer programming methods and approaches - a survey." Journal of Cybernetics and Information Technologies 11.1, 2011.

[14] R. Anand, D. Aggarwal and V. Kumar, "A comparative analysis of optimization solvers", Journal of Statistics and Management Systems, 20:4, pp. 623-635, 2017.

[15] B. Meindl, and M. Templ, "Analysis of Commercial and Free and Open Source Solvers for the Cell Suppression Problem". Tansaction on Data Privacy, Volume 6, pp. 147-159, 2013.