REGMPE

Revista de Empreendedorismo e Gestão de Micro e Pequenas Empresas

USO DA PESQUISA OPERACIONAL NA GESTÃO DE UMA MICROEMPRESA

USE OF OPERATIONAL RESEARCH FOR THE MANAGEMENT OF A MICROENTERPRISE

Yudi Matsuguma Yoshida[1]
Stella Jacyszyn Bachega[2]
Dalton Matsuo Tavares[3]

Resumo

A busca pelo crescimento de uma empresa com os menores custos é um dos maiores desafios dos gestores nos dias de hoje. Uma dentre várias opções é o uso de técnicas de otimização presentes na Pesquisa Operacional. Há vários otimizadores baseados em linguagens de programação voltados à resolução de problemas de otimização. Entretanto, cada linguagem possui suas particularidades e, consequentemente, suas limitações. A linguagem Julia foi criada com o objetivo de combinar as vantagens de diversas linguagens com uma sintaxe relativamente simples, o que a torna mais amigável ao usuário. O objetivo deste artigo é aplicar técnicas de Pesquisa Operacional para auxiliar a gestão de uma microempresa. Para tanto, foi empregada a abordagem quantitativa e o procedimento de pesquisa experimental. Com as devidas traduções dos problemas para a linguagem Julia, o otimizador determinou o melhor *mix* de produção de salgados para uma empresa do ramo alimentício e o menor trajeto para a entrega dos mesmos. Por abordar as técnicas de programação linear e problema do caixeiro viajante, o presente artigo pode ser explorado como ferramenta instrutiva em futuras pesquisas científicas e/ou para auxílio no processo de tomada de decisão de uma organização.

Palavras-chave: Pesquisa Operacional. Linguagem Julia. Programação Linear. Problema do Caixeiro Viajante.

Abstract

The search for the growth of a company with the lowest costs is one of the biggest challenges for managers today. One of several options is the use of optimization techniques present in Operational Research. There are several optimizers based on programming languages aimed at solving optimization problems, however, each language has its traits and therefore, its limitations. The Julia language was created with the aim of combining the advantages of several languages, with a relatively simple syntax, which makes it more user friendly. The purpose of this article is to apply Operational Research techniques to assist the management of a microenterprise. For this, the quantitative approach and the experimental research procedure were used. With the proper coding of the problems into the Julia language, the optimizer determined the best mix for the production of snacks for a food company and the shortest route

[1] Universidade Federal de Goiás – Regional Catalão – Unidade Acadêmica Especial de Engenharia (FENG). E-mail: yudimatsuguma@gmail.com. ORCID: https://orcid.org/0000-0002-5701-7837
[2] Universidade Federal de Goiás – Regional Catalão – Unidade Acadêmica Especial de Engenharia (FENG). E-mail: stella@ufg.br. ORCID: https://orcid.org/0000-0002-7533-5361
[3] Universidade Federal de Goiás – Regional Catalão – Unidade Acadêmica Especial de Biotecnologia (IBIOTEC). E-mail: dalton_tavares@ufg.br. ORCID: https://orcid.org/0000-0001-8531-5578

Yoshida, Y.M., Bachega, S.J., Tavares, D.M.; Uso da Pesquisa Operacional na Gestão de uma Microempresa. Revista de Empreendedorismo e Gestão de Micro e Pequenas Empresas V.5, Nº1, p.58-78, Jan/Abr. 2019. Artigo recebido em 15/02/2020. Última versão recebida em 10/03/2020. Aprovado em 05/04/2019.

for their delivery. By addressing the techniques of linear programming and the traveling salesman problem, the paper can be explored as an instructive tool in future scientific research. It can also aid in the decision-making process of an organization.
Keywords: Operational Research. Julia language. Linear Programming. Traveling Salesman Problem.

1. Introdução

A Pesquisa Operacional (PO) trata-se de uma pesquisa das operações, a qual é utilizada em problemas que exigem uma condução, coordenação e tomada de decisões em relação as operações de uma organização (HILLIER; LIEBERMAN, 2013). Portanto, a PO pode ser abordada em diversos segmentos, como na produção, logística, gestão financeira e planejamento e controle da produção, o que denota a sua interdisciplinaridade e multidisciplinaridade (ARENALES et al., 2007).

Várias dessas abordagens, as quais requeriam um grande volume de contas e que, muitas vezes, eram inviáveis de ser resolvidas manualmente, puderam ser solucionadas com o avanço da tecnologia dos computadores e a criação de diversos softwares de otimização especializados (MOREIRA, 2010). Tais programas possuem vantagens e desvantagens em relação a produtividade e desempenho. Como exemplo, a linguagem C e C++, as quais podem ser usadas em estudos de PO, possibilitam a implementação de algoritmos para uma rápida execução, mas possuem uma sintaxe complexa que exige conhecimentos específicos de computação. Já a linguagem Phyton, contém diversas funções e ferramentas de alto nível, o que oferece a execução de algoritmos com poucas linhas de código. Em contrapartida, seu desempenho é inferior ao C/C++ (JULIALANG.ORG, 2018).

Com o intuito de substituir várias linguagens, de modo a combinar suas vantagens com uma sintaxe relativamente simples, a linguagem Julia surgiu em 2009 no Massachusetts Institute of Technology (MIT). Esta foi baseada em diversas linguagens em uma tentativa de mantê-la genérica, de forma similar ao C, Fortran e Phyton, com o objetivo de ser utilizada por programadores, matemáticos, estatísticos e pesquisadores (BEZANSON et al., 2012).

Mesmo sendo uma linguagem sendo relativamente ‘jovem’, já existem vários trabalhos científicos publicados em diversas áreas e disciplinas, como em genética, álgebra linear, estatística, econometria, neurociência computacional, análise de riscos e também, PO. Com base nesse contexto, a presente pesquisa justifica-se pela importância do tema, como pode ser observado em publicações recentes que utilizaram a linguagem Julia, como em Anderson et al. (2017), Baldassi (2017), Bezanson et al. (2017), Castelluccia (2017), Mogensen e Riseth (2018), Rackauckas et al. (2018). Ainda, autores como Almeida et al. (2017), Santana et al.

(2018), Sousa et al. (2019), Souza et al. (2019) e Teixeira et al. (2017) incentivaram a aplicação de técnicas de PO em micro e pequenas empresas para a melhoria da gestão empresarial. Assim, unir a linguagem Julia para resolver problemas de PO em micro e pequenas empresas pode trazer benefícios gerenciais a estas, ao possibilitar que empresas de menor porte possam ter o benefício de técnicas de complexidade.

Assim, o objetivo geral do artigo é aplicar técnicas de Pesquisa Operacional para auxiliar a gestão de uma microempresa que atua no ramo alimentício. Para tanto, foi utilizada a linguagem Julia. Por meio da análise da realidade empresarial, foi selecionado um problema de Programação Linear (PL) com o objetivo de maximizar a receita da empresa por meio da determinação do melhor *mix* de salgados, desde que suas restrições sejam respeitadas. Além disso, a entrega dos salgados é feita pela própria empresa, surgindo também a oportunidade de aplicar o problema do caixeiro viajante.

A estrutura do artigo é a que segue: a seção dois apresenta o referencial teórico e a seção três discorre sobre a abordagem da metodologia utilizada para a realização do trabalho. A seção quatro trata dos resultados e discussões. Por fim, na última seção estão as considerações finais.

2. Referencial Teórico

2.1 Julia

Criada em 2009 no MIT, a linguagem de programação Julia é considerada de alto nível, dinâmica e possui alto desempenho para computação gráfica e numérica. Possui um compilador sofisticado, execução paralela distribuída, precisão numérica e uma extensa biblioteca de funções matemáticas. Além disso, possui uma sintaxe genérica baseada em diversas linguagens (C, Fortran, Matlab, Phyton, entre outras), para que o software possa ser utilizado não somente por programadores, mas também por matemáticos, estatísticos e pesquisadores (BEZANSON et al., 2012).

De acordo com o site oficial, a programação possui como principais características: (i) multimétodos, onde possui a capacidade de definir o comportamento da função por meio de várias combinações de tipos de argumentos; (ii) capacidade de paralelismo e computação distribuída, sendo possível dividir o problema em partes e distribuí-lo entre vários computadores; (iii) possui um gerenciador de pacotes práticos e simples de usar; (iv) boa performance, aproximando-a da linguagem C em vários aspectos; e (v) poderosa capacidade para gerenciamento de outros processos (JULIALANG.ORG, 2018).

2.2 Pesquisa operacional: definição e algumas técnicas

Segundo Hillier e Lieberman (2013), a PO se trata de pesquisar as operações, sendo utilizada em problemas que necessitam de uma condução, coordenação e tomada de decisões quanto as operações em uma organização. Assim sendo, várias áreas distintas (como logística, gestão financeira, planejamento e controle da produção, saúde, entre outros) podem ter seus problemas resolvidos por meio das técnicas da PO.

Uma das técnicas mais gerais usadas na PO é a simulação de sistemas, que consiste na tentativa da representação exata de um sistema real em um sistema computacional, permitindo testes de hipóteses de novos cenários e, consequentemente, auxiliando no processo de tomada de decisão da organização (PEREIRA et al., 2015). A introdução de uma nova bomba de combustível em um posto de gasolina é um exemplo de estudo da simulação.

A Programação Linear (PL) é responsável por resolver os problemas de alocação dos recursos limitados para as atividades que competem entre si por tais recursos. Todavia, a PL possui outras aplicações, desde que a formulação do problema se encaixe com o formato matemático genérico da mesma (HILLIER; LIEBERMAN, 2013).

Já os problemas que possuem uma relação entre seus componentes, de forma que possam ser formulados como uma rede, são resolvidos por meio da otimização de redes. Seu uso é muito amplo uma vez que as redes existem em diversas áreas, como redes de transportes, elétrica, comunicações, produção e na gestão de projetos (ARENALES et al., 2007).

Dentro da otimização em redes, existem vários algoritmos que auxiliam na resolução dos problemas e tomadas de decisão, tais como: algoritmo da árvore geradora mínima, cujo objetivo é conectar todos os nós da rede, direta ou indiretamente, de modo que o comprimento total dos arcos seja o menor possível; algoritmo de fluxo máximo, o qual objetiva determinar a rota de passagem da origem até o nó de destino (sorvedouro) com a maior quantidade total de fluxo possível; algoritmo do caminho crítico, usado amplamente em gestão de projetos na determinação de atividades críticas do projeto; problema do caminho mais curto, o qual trata da determinação do caminho que possui o menor comprimento entre o nó de origem e o nó de destino; e o problema do caixeiro viajante (TAHA, 2008).

2.3 Programação linear

A Programação Linear (PL) surgiu no final da década de quarenta e, com a criação do computador na década seguinte, seu desenvolvimento foi acelerado e bastante difundido (PRADO, 2007). Para Moreira (2010), a PL é um dos modelos matemáticos mais populares, configurado para solucionar problemas que tenham variáveis mensuráveis e correlacionadas, expressas por equações e/ou inequações matemáticas. A popularização do modelo deve-se ao

fato de que há inúmeros problemas nas áreas científicas e sociais que podem ser formulados dessa maneira.

Segundo Ehrlich (1985), a PL é uma ferramenta de planejamento que ajuda na decisão de quais atividades (variáveis de decisão) empreender, uma vez que estas competem entre si pela utilização de recursos escassos (restrições), já que tais recursos são insuficientes para que todas as atividades sejam realizadas no nível máximo que se deseja.

Atualmente, tornou-se uma ferramenta padrão que economizou milhões de dólares para muitas empresas. Vale ressaltar que a PL não se restringe apenas ao setor industrial, mas também a várias áreas da sociedade (HILLIER; LIEBERMAN, 2013).

De acordo com Passos (2008), a PL é uma técnica de otimização aplicada em um sistema de equações, ou inequações, matemáticas lineares de um problema. Seu objetivo é maximizar ou minimizar uma função linear através da determinação dos valores das variáveis de decisão, levando em conta que as restrições devem ser satisfeitas.

Segundo Passos (2008), o padrão do modelo de um problema de PL é dado segundo as Eq. 1 até 7.

$$\text{Max (ou Min) } Z = c_1x_1 + c_2x_2 + c_3x_3 + ... + c_nx_n \quad (1)$$

$$\text{Sujeito a: } a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + ... + a_{1n}x_n = b_1 \quad (2)$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + ... + a_{2n}x_n = b_2 \quad (3)$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + ... + a_{3n}x_n = b_3 \quad (4)$$

$$... \quad ... \quad ... \quad ... \quad ... \quad (5)$$

$$a_{mn}x_1 + a_{m2}x_2 + a_{m3}x_3 + ... + a_{mn}x_n = b_m \quad (6)$$

$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, ..., x_n \geq 0 \quad (7)$$

Onde X: $\{x_1, x_2, x_3, ..., x_n\}$: são as variáveis de decisão; A: $\{a_1, a_2, a_3, ..., a_m\}$: são os coeficientes das variáveis; e B: $\{b_1, b_2, b_3, ..., b_m\}$: são os termos independentes que representam os recursos disponíveis. As variáveis de decisão são essenciais para a resolução do problema. A função objetivo, dada pela equação (1), deseja otimizar os recursos envolvidos por meio da maximização ou minimização dos mesmos. As restrições, dadas segundo as Eq. 2 até 7, são as limitações do problema, como por exemplo, a capacidade produtiva do sistema, quantidade de mão de obra, matéria prima ou estoque (LINS; CALÔBA, 2010, p. 7).

Para que a linearidade seja satisfeita, três propriedades básicas devem ser atendidas pela estrutura do problema: proporcionalidade, aditividade e certeza. A primeira propriedade propõe que a contribuição de cada variável de decisão seja diretamente proporcional ao valor da variável, tanto na função objetivo quanto nas restrições. A segunda propriedade afirma que a

contribuição total das variáveis da função objetivo e das restrições é a soma das contribuições individuais de cada variável. Por fim, a terceira propriedade diz respeito a todos os coeficientes da função objetivo e às restrições do modelo de PL, os quais são constantes conhecidas (TAHA, 2008).

2.4 Problema do caixeiro viajante

O problema do caixeiro viajante (PCV) trata-se de um problema de otimização combinatória, portanto, sendo muito utilizado em experimentos de vários métodos de otimização, uma vez que é de fácil descrição, compreensão e possui larga aplicabilidade (KARP, 1975). De acordo com Hillier e Lieberman (2013), o PCV pode ser descrito como um problema da definição da menor rota (em termos de distância ou custos) de um vendedor que viaja várias cidades (nós) por uma rede e, no final, regressa à cidade de origem. Vale ressaltar que cada cidade só pode ser visitada uma única vez.

Um problema desse tipo pode ser resolvido pela programação linear, mais especificamente, pelo método Simplex, no qual serão analisadas todas as rotas possíveis para o problema. Entretanto, a partir de certa quantidade de nós no problema, torna-se inviável a sua resolução por esse método. Considerando que "n" seja o número de cidades e que o PCV seja simétrico, ou seja, quando o caminho da cidade A até a cidade B é igual ao caminho da cidade B à cidade A, a quantidade de rotas possíveis é dada por (n-1)!/2, onde existem (n-1) possibilidades para a primeira cidade, (n-2) para a segunda e assim sucessivamente. A divisão por 2 indica que cada rota possui uma rota com o mesmo módulo, mas no sentido oposto. Caso o PCV seja assimétrico, o número de rotas possíveis é dado por (n-1)!. Portanto, um problema com apenas 10 cidades possui 181.400 rotas para um caso de PCV simétrico e 362.800 para um PCV assimétrico.

Sendo G = (N, E) um grafo, onde N = {1, 2, ..., n} é o conjunto de nós, E = {1, 2, ..., m} o conjunto de arestas de G, o objetivo é determinar o menor ciclo Hamiltoniano do grafo G (ARENALES et al., 2007).

É possível perceber que dependendo do valor de "n", a resolução do problema pelos métodos exatos torna-se inviável. Diante de tais dificuldades, várias abordagens heurísticas têm sido aplicadas e proporcionaram soluções de boa qualidade, contudo não garantem a otimalidade de um resultado, uma vez que é determinado por métodos intuitivos (SILVEIRA, 2000).

Um método heurístico bastante conhecido é o "vizinho mais próximo", o qual seleciona a rota mais curta para os nós vizinhos do qual o caixeiro se encontra. Portanto, define-se o nó

mais próximo do nó de origem como sendo o segundo nó e, a partir desse, seleciona o nó mais próximo dentre os nós não selecionados, o qual será o terceiro nó. Repete o processo até que todos os nós sejam conectados (HILLIER; LIEBERMAN, 2013).

3. Metodologia

A explicação científica utilizada nesta pesquisa foi a hipotético-dedutiva, conforme as considerações de Carvalho (2000). Assim, a proposição de que é possível aplicar técnicas de PO em uma microempresa para melhoria da gestão empresarial foi corroborada ao final da pesquisa.

Ainda, foi empregada a abordagem quantitativa segundo Creswell (2010), devido à forma como os dados foram manipulados na pesquisa. Com isso, foram gerados modelos matemáticos com equações e inequações estruturadas, incorporando diretrizes causais e a identificação da correlação de múltiplas variáveis.

O procedimento de pesquisa experimental foi adotado. Este é mais indicado para abordagens quantitativas e pode ser relacionado a modelagens matemáticas e simulações computacionais (BRYMAN, 1989). Gil (2009) complementa que este procedimento abrange tanto a determinação do objeto de estudo, quanto a seleção das variáveis que o influencia.

As etapas para a condução do estudo de PO foram fundamentadas por Hillier e Lieberman (2013). As etapas desempenhas nesta pesquisa foram:

Etapa 1) Definição do problema de interesse e coleta de dados: nesta etapa, foram identificados problemas clássicos de pesquisa operacional com o intuito de selecionar o problema a ser modelado e os dados para a modelagem matemática foram reunidos. Foram selecionados problemas de programação linear e do caixeiro viajante. Os dados para embasar os exemplos presentes neste trabalho foram obtidos com um microempresário do ramo de alimentação.

Etapa 2) Formulação do modelo matemático que representa determinado problema: aqui o modelo matemático foi formalizado em funções, equações e inequações, conforme a técnica de PO adequada para representação e posterior resolução.

Etapa 3) Desenvolvimento do procedimento computacional para encontrar uma solução para o problema modelado: o modelo matemático foi traduzido para a linguagem Julia e os otimizadores empregados determinaram uma possível solução ótima.

Etapa 4) Teste e aprimoramento do modelo: houve a preocupação em validar os modelos, fazendo que estes representassem fielmente a realidade. Assim, verificou-se novamente a estrutura do modelo e foi possível encontrar a resposta certa para seus respectivos problemas.

Etapa 5) Documentação: a documentação foi feita para favorecer o entendimento da pesquisa efetuada e garantir sua continuidade futura.

4. Resultados obtidos

Nesta seção, há a apresentação dos resultados obtidos primeiramente no problema de programação linear e, posteriormente, no problema do caixeiro viajante.

4.1 Programação Linear

A programação linear (PL) é definida como um problema de otimização onde a função objetivo e as restrições são representadas por equações, ou inequações, lineares. Além disso, a PL é responsável por resolver problemas de alocação de recursos limitados para as atividades que competem entre si por tais recursos. Como o problema definido para a realização do estudo tem o objetivo de determinar o melhor *mix* de salgados que maximiza a receita da empresa, baseado nas quantidades de matérias-primas disponíveis no estoque, capacidade de produção e tempo médio gasto por produto e que a função objetivo e as restrições podem ser representadas de forma linear, fica evidente que o mesmo pode ser tratado como um problema de PL. Ainda, as variáveis serão tratadas como números inteiros; dessa forma, é possível aplicar a Programação Linear Inteira (PLI).

A empresa de salgados, situada em uma cidade no sudeste goiano, deseja maximizar sua receita com a venda de seis tipos de salgados. As informações referentes a cada salgado podem ser consultadas na Tabela 1.

Tabela 1 - Dados sobre a receita com a venda dos salgados

Salgados	Variável (unidade)	Receita (R$) por unidade
Coxinha	x_1	2,50
Enroladinho de salsicha	x_2	2,50
Pastel de carne	x_3	2,50
Pastel de presunto e queijo	x_4	2,50
Quibe de carne	x_5	2,50
Quibe de queijo	x_6	2,50

Cada salgado é composto por diferentes quantidades de matérias-primas (Tabela 2). Como pode ser visto, uma coxinha (quantidade de coxinhas é representada pela variável x_1) consome aproximadamente 25g de batata e 65g de frango. Cabe ressaltar que os itens de matéria-prima representados nestas restrições foram somente os disponibilizados pelo empresário, pois estes representam maior custo por item.

Tabela 2 - Quantidade de matéria-prima utilizada para cada salgado e disponível no estoque

Recurso	x_1	x_2	x_3	x_4	x_5	x_6	Estoque (g)
Batata (g)	25	0	0	0	0	0	2.500
Carne (g)	0	0	40	0	75	65	28.000
Frango (g)	65	0	0	0	0	0	6.000
Milho (g)	0	0	20	0	0	0	4.000
Muçarela (g)	0	35	0	50	0	25	10.000
Presunto (g)	0	0	0	50	0	0	3.000
Salsicha (g)	0	100	0	0	0	0	4.000
Trigo (g)	0	0	0	0	65	50	16.000

Além disso, cada salgado possui sua demanda mínima e máxima diária baseada na capacidade de produção, como pode ser visto na Tabela 3. Cada salgado deve ter uma produção diária mínima de quatro itens para atender a demanda mínima. A produção máxima, por exemplo da coxinha, deve ser de 90 unidades.

Tabela 3 - Demanda mínima e máxima para cada tipo de salgado

Salgado	Demanda mínima	Demanda máxima
Coxinha	$x_1 \geq 4$	$x_1 \leq 90$
Enroladinho de salsicha	$x_2 \geq 4$	$x_2 \leq 40$
Pastel de carne	$x_3 \geq 4$	$x_3 \leq 180$
Pastel de presunto e queijo	$x_4 \geq 4$	$x_4 \leq 60$
Quibe de carne	$x_5 \geq 4$	$x_5 \leq 160$
Quibe de queijo	$x_6 \geq 4$	$x_6 \leq 120$

Por último, os salgados levam cerca de 25 segundos para serem montados, independentemente do tipo. O tempo total reservado para a montagem dos salgados são de 4 horas e 30 minutos, ou 16.200 segundos. Dessa forma, a Eq. 8 refere-se a última restrição do problema.

$$25x_1 + 25x_2 + 25x_3 + 25x_4 + 25x_5 + 25x_6 \leq 16.200 \quad (8)$$

Sendo assim, é possível representar o problema da empresa por equações/inequações matemáticas lineares. A Eq. 9 refere-se à função objetivo, enquanto que as Eq. 10 até a Eq. 32 são as restrições.

$$\text{Max } Z = 2,5x_1 + 2,5x_2 + 2,5x_3 + 2,5x_4 + 2,5x_5 + 2,5x_6 \quad (9)$$

Sujeito a:

$$25x_1 \leq 2500 \quad (10)$$

$$40x_3 + 75x_5 + 65x_6 \leq 28.000 \quad (11)$$

$$65x_1 \leq 6.000 \quad (12)$$

$$20x_3 \leq 4.000 \quad (13)$$

$$35x_2 + 50x_4 + 25x_6 \leq 10.000 \quad (14)$$

$$50x_4 \leq 3.000 \quad (15)$$

$$100x_2 \leq 4.000 \quad (16)$$

$$65x_5 + 50x_6 \leq 16.000 \quad (17)$$

$$x_1 \leq 90 \quad (18)$$

$$x_2 \leq 40 \quad (19)$$

$$x_3 \leq 180 \quad (20)$$

$$x_4 \leq 60 \quad (21)$$

$$x_5 \leq 160 \quad (22)$$

$$x_6 \leq 120 \quad (23)$$

$$x_1 \geq 4 \quad (24)$$

$$x_2 \geq 4 \quad (25)$$

$$x_3 \geq 4 \quad (26)$$

$$x_4 \geq 4 \quad (27)$$

$$x_5 \geq 4 \quad (28)$$

$$x_6 \geq 4 \quad (29)$$

$$25x_1 + 25x_2 + 25x_3 + 25x_4 + 25x_5 + 25x_6 \leq 16.200 \quad (30)$$

$$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \quad (31)$$

$$x_1, x_2, x_3, x_4, x_5, x_6 \text{ inteiros} \quad (32)$$

Definido o modelo matemático, o próximo passo foi traduzi-lo para a linguagem Julia, onde o otimizador encontrará uma possível solução. Primeiramente, foi necessário instalar o pacote Julia for Mathematical Programming (JuMP) para tornar o Julia hábil a ler problemas de otimização. Trata-se de uma linguagem matemática utilizada para resolução de problemas de programação linear, não linear, inteira mista, cônica de segunda ordem e semidefinida. Entretanto, o pacote não é capaz de solucionar problemas de otimização sozinho, sendo necessário que otimizadores também sejam instalados juntos para interagir com o JuMP. Neste caso, em particular, foi instalado o otimizador "Cbc", o qual é gratuito e suporta tanto problemas de PL quanto de problemas de programação inteira mista.

A instalação dos pacotes necessita de apenas duas linhas de comando, uma linha para cada pacote. Recomenda-se que, ao final da instalação, os mesmos sejam atualizados para a última versão por meio de uma outra linha de comando. Depois, é necessário informar quais pacotes serão utilizados para a resolução, além de criar o objeto que armazenará o modelo do problema (Figuras 1 e 2). A última linha de comando da Figura 2 serviu para criar o objeto chamado ModeloMat. O nome do modelo não necessariamente precisa ser o mesmo, podendo receber qualquer nome. Vale ressaltar que futuramente o objeto precisa ser escrito da mesma forma, pois há diferenciação de letras maiúsculas das minúsculas. O argumento entre parênteses indica que o otimizador utilizado para resolver o problema foi o Cbc.

```
julia> import Pkg; Pkg.add("JuMP")
julia> import Pkg; Pkg.add("Cbc")
julia> Pkg.update()
```

Figura 1 – Comandos para instalação e atualização dos pacotes JuMP e Cbc no Julia

```
julia> using JuMP
julia> using Cbc
julia> ModeloMat = Model(with_optimizer(Cbc.Optimizer))
```

Figura 2 – Comandos para carregar os pacotes instalados e criação do objeto “ModeloMat”

Em seguida, conforme Figura 3, definiram-se as variáveis de decisão do modelo com o comando *@variable (nome do objeto, nome da variável e limites, tipo da variável)*. Os limites podem ser inferior, superior ou ambos. Caso o tipo da variável não seja especificado, a mesma é considerada como real. Usa-se “Bin” para variáveis binárias e “Int” para inteiras.

```
julia> @variable(ModeloMat, x1 >= 0, Int)
julia> @variable(ModeloMat, x2 >= 0, Int)
julia> @variable(ModeloMat, x3 >= 0, Int)
julia> @variable(ModeloMat, x4 >= 0, Int)
julia> @variable(ModeloMat, x5 >= 0, Int)
julia> @variable(ModeloMat, x6 >= 0, Int)
```

Figura 3 – Definindo as variáveis do problema no Julia

Como as variáveis presentes no modelo elaborado são os tipos de salgados, a quantidade produzida deve ser inteira, o que justifica serem classificadas como inteiras. O próximo passo consistiu na definição da função objetivo através do comando *@objective (nome do modelo, Min ou Max, equação),* como pode ser observado na Figura 4. Por fim, foram adicionadas as restrições pelo comando *@constraint (nome do modelo, equação/inequação),* conforme a Figura 5.

```
julia> @objective(ModeloMat, Max, 2.5x1 + 2.5x2 + 2.5x3 + 2.5x4 + 2.5x5 + 2.5x6)
```

Figura 4 – Definindo a função objetivo do problema no Julia

```
julia> @constraint(ModeloMat, 25x1 <= 2500)
julia> @constraint(ModeloMat, 40x3 + 75x5 + 65x6 <= 28000)
julia> @constraint(ModeloMat, 65x1 <= 6000)
julia> @constraint(ModeloMat, 20x3 <= 4000)
julia> @constraint(ModeloMat, 35x2 + 50x4 + 25x6 <= 10000)
julia> @constraint(ModeloMat, 50x4 <= 3000)
julia> @constraint(ModeloMat, 100x2 <= 4000)
julia> @constraint(ModeloMat, 65x5 + 50x6 <= 16000)
julia> @constraint(ModeloMat, x1 <= 90)
julia> @constraint(ModeloMat, x2 <= 40)
julia> @constraint(ModeloMat, x3 <= 180)
julia> @constraint(ModeloMat, x4 <= 60)
julia> @constraint(ModeloMat, x5 <= 160)
julia> @constraint(ModeloMat, x6 <= 120)
julia> @constraint(ModeloMat, x1 >= 4)
julia> @constraint(ModeloMat, x2 >= 4)
julia> @constraint(ModeloMat, x3 >= 4)
julia> @constraint(ModeloMat, x4 >= 4)
julia> @constraint(ModeloMat, x5 >= 4)
julia> @constraint(ModeloMat, x6 >= 4)
julia> @constraint(ModeloMat, 25x1 + 25x2 + 25x3 + 25x4 + 25x5 + 25x6 <= 16200)
```

Figura 5 – Definindo as restrições do problema no Julia

Depois da definição das restrições do modelo, o comando *print (nome do modelo)* pode ser usado para mostrar todo o modelo construído (Figura 6).

```
julia> print(ModeloMat)
Max 2.5 x1 + 2.5 x2 + 2.5 x3 + 2.5 x4 + 2.5 x5 + 2.5 x6
Subject to
 x1 integer
 x2 integer
 x3 integer
 x4 integer
 x5 integer
 x6 integer
 x1 >= 0.0
 x2 >= 0.0
 x3 >= 0.0
 x4 >= 0.0
 x5 >= 0.0
 x6 >= 0.0
 x1 >= 4.0
 x2 >= 4.0
 x3 >= 4.0
 x4 >= 4.0
 x5 >= 4.0
 x6 >= 4.0
 25 x1 <= 2500.0
 40 x3 + 75 x5 + 65 x6 <= 28000.0
 65 x1 <= 6000.0
 20 x3 <= 4000.0
 35 x2 + 50 x4 + 25 x6 <= 10000.0
 50 x4 <= 3000.0
 100 x2 <= 4000.0
 65 x5 + 50 x6 <= 16000.0
 x1 <= 90.0
 x2 <= 40.0
 x3 <= 180.0
 x4 <= 60.0
 x5 <= 160.0
 x6 <= 120.0
```

Figura 6 – Modelo definido demonstrado no Julia

Por fim, o comando *optimize!(nome do modelo)* serve para resolver o problema e determinar a solução ótima (Figura 7). Nesta etapa, algumas respostas são possíveis: (i) *Optimal*, significa que o problema possui uma solução ótima e a mesma foi determinada; (ii) *Infeasible*, ou seja, o problema é considerado inviável por não ter uma solução ótima; e (iii) *Unbounded*, onde valor da função objetivo cresce (ou diminui) indefinidamente à medida que os valores das variáveis também aumentam (ou diminuam).

Como demonstrado na Figura 7, o problema possui uma solução ótima. Além disso, o otimizador Cbc descreve algumas características sobre a resolução do problema. Na parte superior, tem-se a descrição do otimizador. Logo abaixo, há o comentário sobre a resolução do problema por outros métodos. Caso o problema permitisse valores contínuos nas variáveis, o

valor da função objetivo seria de R$1.609,62. O programa também cita outros métodos de resolução, porém os mesmos não foram utilizados por não atenderem os pré-requisitos, como é constatado pela frase "*was tried 0 times [...]"*.

```
julia> optimize!(ModeloMat)
Welcome to the CBC MILP Solver
Version: 2.9.9
Build Date: Dec 31 2018

command line - Cbc_C_Interface -solve -quit (default strategy 1)
Continuous objective value is 1609.62 - 0.22 seconds
Cgl0004I processed model has 1 rows, 2 columns (2 integer (0 of which binary)) and 2 elements
Cutoff increment increased from 1e-05 to 2.4999
Cbc0012I Integer solution of -1607.5 found by DiveCoefficient after 0 iterations and 0 nodes (1.64 seconds)
Cbc0001I Search completed - best objective -1607.5, took 1 iterations and 0 nodes (2.54 seconds)
Cbc0035I Maximum depth 0, 0 variables fixed on reduced cost
Cuts at root node changed objective from -1609.62 to -1607.5
Probing was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Gomory was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Knapsack was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Clique was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
FlowCover was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
TwoMirCuts was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Result - Optimal solution found

Objective value:                1607.50000000
Enumerated nodes:               0
Total iterations:               1
Time (CPU seconds):             2.61
Time (Wallclock seconds):       2.61

Total time (CPU seconds):       2.64   (Wallclock seconds):       2.64
```

Figura 7 – Solução ótima determinada pelo Julia

Por fim, o relatório engloba informações referentes ao número de iterações para atingir a solução ótima e os tempos de CPU e Wallclock, onde apenas o primeiro considera os tempos de parada durante a execução de uma tarefa. O Wallclock *time* contabiliza todo o tempo passado para se executar uma tarefa, contabilizando inclusive as interrupções. Assim, como se pode notar, não houve interrupção na execução da atividade, pois o Wallclock *time* foi igual ao CPU *time*.

Os resultados são impressos ao utilizar a combinação do comando *print()* com o comando do que se deseja saber. Usa-se *JuMP.objective_value(nome do modelo)* para o valor da função objetivo e *JuMP.value(variável)* para as variáveis de decisão. Portanto, conforme a Figura 8, a empresa terá uma receita máxima de R$1.607,50 caso decida produzir 90 coxinhas, 40 enroladinhos de salsicha, 180 pasteis de carne, 60 pasteis de presunto e queijo, 153 quibes de carne e 120 quibes de queijo.

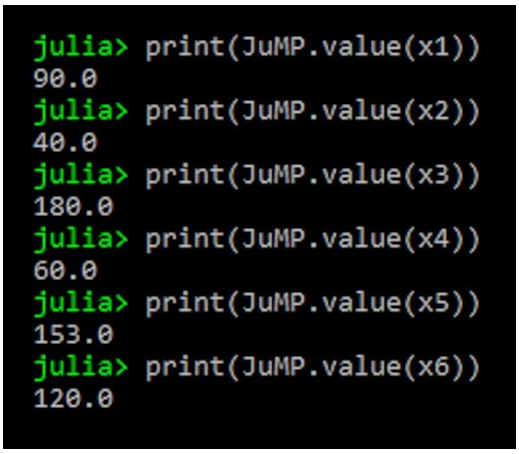

Figura 8 – Valores das variáveis determinada pelo Julia

4.2 Problema do caixeiro viajante

Além da definição do melhor *mix* de salgados a ser produzido, a empresa também deseja otimizar sua rota de entrega dos mesmos. Para isso, a técnica utilizada foi o problema do caixeiro viajante, que determina a menor rota para percorrer uma série de pontos apenas uma vez e, no final, retorna ao ponto de origem.

Ao todo, a empresa realiza a entrega dos salgados para quatro clientes. Cada cliente é definido como um ponto (ou nó) na rede. Esta, por sua vez, é definida como um conjunto de nós conectados por arcos. Neste caso em particular, os nós representam os clientes e o arcos são as distâncias entre os clientes. O nó de origem é o C1, ou seja, é o nó onde se iniciará o trajeto (a empresa estudada). Assim, a partir da empresa foco do estudo (C1) até o cliente C2, a distância percorrida é de 1,5 km. A Tabela 4 expõe as distâncias dadas em quilômetros (km).

Tabela 4 – Distância entre os clientes em quilômetros

Nós / Nós	**C_1**	**C_2**	**C_3**	**C_4**	**C_5**
C_1	0	1,5	1,4	2,6	3,5
C_2	1,5	0	0,8	2,0	3,2
C_3	1,4	0,8	0	1,5	2,2
C_4	2,6	2,0	1,5	0	0,8
C_5	3,5	3,2	2,2	0,8	0

É possível perceber que o problema é simétrico, uma vez que a distância para se deslocar do cliente *i* para o cliente *j* é a mesma no caminho inverso. A Figura 9 representa o problema de forma visual.

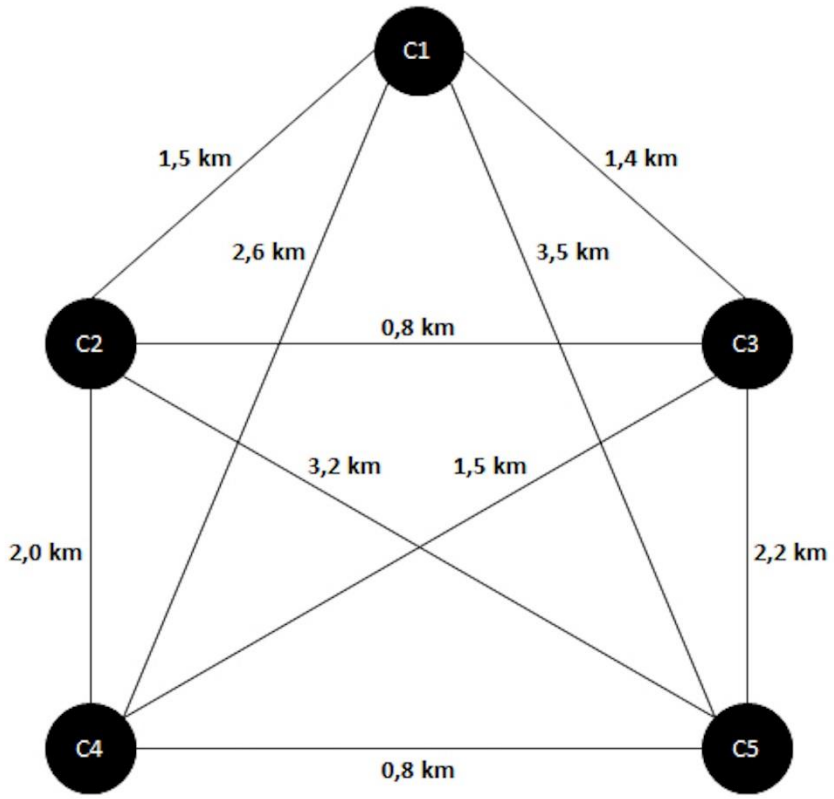

Figura 9 – Representação visual do problema do caixeiro viajante

Antes de traduzir o problema proposto para a linguagem Julia, é necessário instalar os pacotes necessários para a resolução. Desse modo, o pacote utilizado foi "TravelingSalesmanHeuristics" (Figura 10).

```
julia> import Pkg; Pkg.add("TravelingSalesmanHeuristics")

julia> Pkg.update()
```

Figura 10 – Representação visual do problema do caixeiro viajante

Dentro do pacote há várias funções, como por exemplo: (i) *cheapest_insertion*, onde calcula-se a rota mais barata em termos monetários; (ii) *farthest_insertion*, que calcula a rota com base nos vértices mais distantes do vértice de origem; e (iii) *nearest_neighbor*, que se trata da heurística do vizinho mais próximo.

Depois da instalação e atualização do pacote é necessário declará-lo e inserir as distâncias entre os clientes em formato de matriz quadrada (Figura 11). Dessa forma, a quantidade de linhas deve ser igual à quantidade de colunas. O não cumprimento desta etapa

retornará um erro nas etapas seguintes. A matriz deve ser armazenada em um objeto, vale lembrar que o nome do objeto é livre, porém, deve ser escrito da mesma forma futuramente, pois as letras maiúsculas e minúsculas são diferenciadas na linguagem.

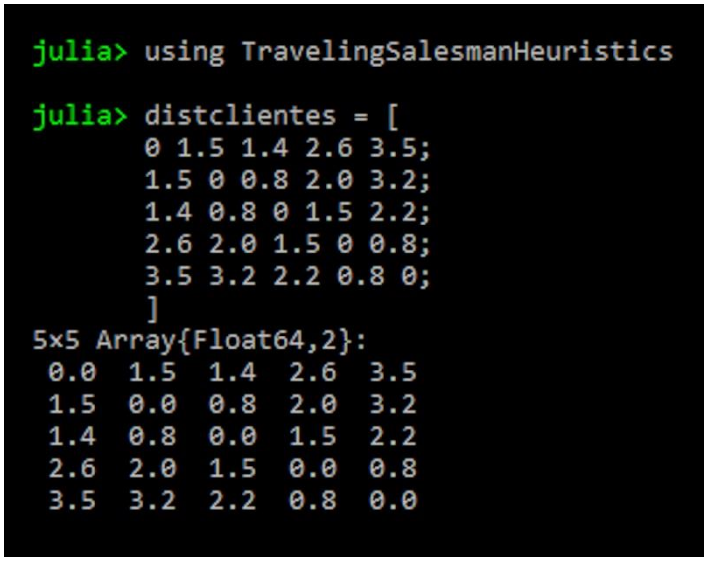

Figura 11 – Declarando o pacote e definindo o objeto com as distâncias entre os clientes

O objeto criado para armazenar as distâncias entre os clientes foi "*distclientes*". Os valores são separados por espaços e ao final de cada linha, é necessário inserir um ponto e vírgula para indicar o fim da mesma. A matriz é iniciada e finalizada com colchetes. Logo abaixo é possível conferir se a matriz foi criada corretamente. A matriz criada é de ordem cinco, possui dados do tipo real 64-bit e é de dimensão 2.

Em seguida, chama-se a função correspondente da heurística escolhida para a resolução do problema. Os parâmetros necessários para a função do vizinho mais próximo são *nearest_neighbor (distmat, firstcity)*. O parâmetro *distmat* corresponde a matriz com as distâncias e o segundo parâmetro define qual nó será o nó de origem. Como a matriz "*distclientes*" possui 5 linhas, o parâmetro pode receber um valor de um a cinco, uma vez que o Julia inicia a contagem a partir do número um (indicando o primeiro nó da rede), como observado na Figura 12.

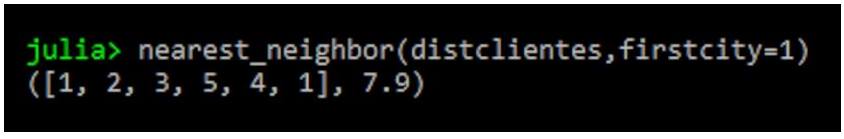

Figura 12 – Solução encontrada no Julia

Portanto, analisando a Figura 12, a melhor rota de entrega dos salgados é dada pelos valores entre colchetes. Assim, primeiramente deve-se partir da empresa (nó C1) e ir para o cliente 2. Em seguida, o cliente 3, descer para o cliente 5 e por fim, ir para o cliente 4 antes de retornar ao nó de origem (C1), que é a empresa. O valor após a vírgula corresponde a distância total da rota, que vale 7,9 km. A Figura 12 ilustra o menor caminho em destaque.

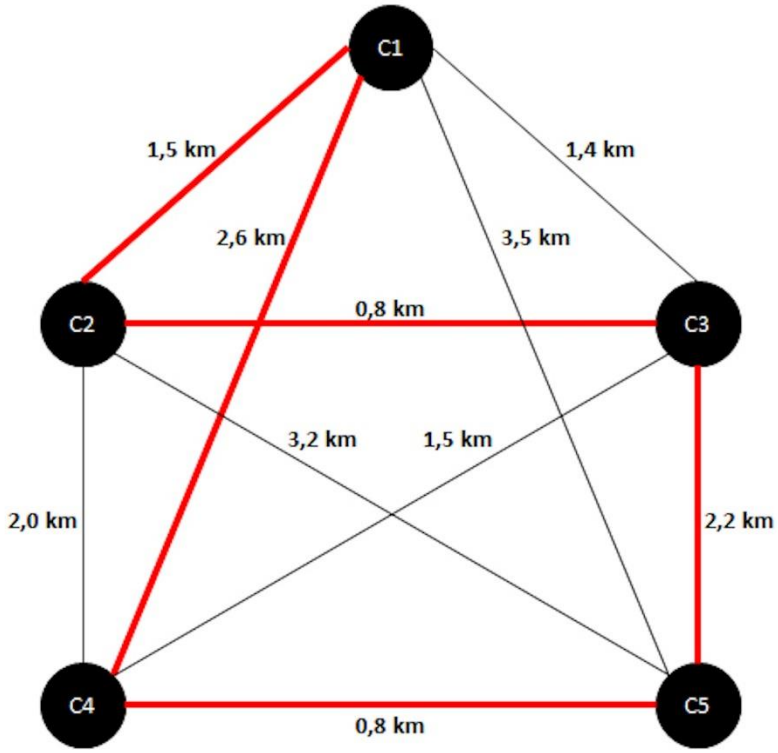

Figura 11 – Solução encontrada no Julia

5. Considerações Finais

A solução ótima para o problema de programação linear que visava a identificação do melhor *mix* de salgados a ser produzido para maximizar a receita da microempresa estudada foi identificada. Também, a melhor rota de entrega, minimizando as distâncias de entrega entre todos os clientes, foi encontrada por meio do problema do caixeiro viajante. Logo, o objetivo almejado no artigo foi alcançado.

Em relação ao primeiro problema, para que a empresa obtenha a máxima receita possível, é necessário que a mesma mantenha suas produções próximas da demanda máxima, uma vez que houve uma diferença de apenas sete quibes de carne. Sobre o problema do caixeiro viajante, salienta-se que a rota de entrega dos salgados terá a mesma distância caso seja feita pelo caminho inverso.

Para os dois problemas, o método de resolução é relativamente simples devido a linguagem de programação Julia conter funções e pacotes específicos para cada um. Consequentemente, o Julia se aproveita das vantagens de diferentes linguagens de programação para obter eficiência e produtividade. Destaca-se, também, o fato de ser uma linguagem livre e *open source*. Ainda, salienta-se que o artigo apresenta duas das várias técnicas de otimização que estão disponíveis para resolução na linguagem Julia. O fato desta reunir várias técnicas auxilia na comodidade para os analistas, pois estes podem se especializar somente na linguagem e resolverem diversos problemas empresariais.

Quanto a contribuição no âmbito empresarial, este artigo pode ser utilizado como um tutorial para uso dessas técnicas de pesquisa operacional nas micro e pequenas empresas, uma vez que as resoluções dos problemas foram feitas passo-a-passo de modo que o leitor consiga utilizá-las. O uso da linguagem Julia foi considerado útil e possível de ser aplicável em empresas de pequeno porte. Já para o âmbito acadêmico, o presente artigo pode ser utilizado como um tutorial para futuros projetos e pesquisas que englobem programação linear e/ou problema do caixeiro viajante. Além disso, divulga aplicação prática de problemas de PO em microempresa, aumentando a divulgação de que uma área complexa pode ser utilizada para apoio gerencial também em micro e pequenas empresas.

Sugere-se, para pesquisas futuras, a aplicação da linguagem Julia em outras micro e pequenas empresas com o intuito de obter o *feedback* dos empresários quanto a experiência de uso e benefícios gerenciais percebidos após implantação dos resultados alcançados.

Referências Bibliográficas

ALMEIDA, W. S.; SILVA, M. E.; SANTOS, N. V. M.; RIBEIRO, L. M.; BACHEGA, S. J. Aplicação de programação linear inteira na maximização do lucro de uma empresa do setor de beleza e estética. In: ENCONTRO NACIONAL DE ENGENHARIA DE PRODUÇÃO, 37., 2017, Joinville. **Anais...**. Rio de Janeiro: ABEPRO, 2017. p. 01-16.

ANDERSON, T. A., LIU, H., KUPER, L., TOTONI, E., VITEK, J.; SHPEISMAN, T. Parallelizing Julia with a Non-Invasive DSL. **DARTS**. v. 3, n. 2, p. 01-29, 2017.

ARENALES, M.; MORABITO, R.; ARMENTANO, V.; YANASSE, H. **Pesquisa Operacional**. 1. ed. Rio de Janeiro: Elsevier, 2007.

BALDASSI, C. A method to reduce the rejection rate in Monte Carlo Markov chains. **Journal of Statistical Mechanics: Theory and Experiment**. Vol. 2017, pp. 01-19, 2017.

BEZANSON, J.; EDELMAN, A.; KARPINSKI, S.; SHAH, V. B. Julia: A Fresh Approach to Numerical Computing. **SIAM Review**, Vol. 59, n. 1, pp. 65-98, 2017.

BEZANSON, J.; KARPINSKI, S.; SHAH, V.; EDELMAN, A. **Why we Created Julia**. 2012. Disponível em: http://julialang.org/blog/2012/02/why-we-created-julia . Acesso em 09/10/2019.

BRYMAN, A. **Research methods and organization studies**. London: Uniwin Hyman, 1989.

CARVALHO, M. C. M. de. A construção do saber científico: algumas proposições. In: CARVALHO, M. C. M. de (org.). **Construindo o saber**. 2.ed. Campinas: Papirus, 2000.

CASTELLUCCIA, P. B. JULIA E JuMP: NOVAS FERRAMENTAS PARA PROGRAMAÇÃO. **Pesquisa Operacional para o Desenvolvimento**, v. 9, n. 2, p. 48-61, 2017.

CRESWELL, J. W. **Projeto de pesquisa.** Métodos qualitativo, quantitativo e misto. 3. ed. Porto Alegre: Artmed, 2010.

EHRLICH, P. J. **Pesquisa Operacional**. 5. ed. São Paulo: Atlas, 1985.

GIL, A. C. **Como elaborar projetos de pesquisa**. 4. ed. São Paulo: Atlas, 2009.

HILLIER, F. S.; LIEBERMAN, G. J. **Introdução à Pesquisa Operacional**. 9. ed. Porto Alegre: AMGH, 2013.

JULIALANG.ORG. **Julia Micro-Benchmarks**. 2018. Disponível em: https://julialang.org/benchmarks/. Acesso em 28/03/2019.

KARP, R. M. **On the computational complexity of combinatorial problems**. Newtorks 5, 1975.

LINS, M. P. E.; CALÔBA, **Guilherme M. Programação Linear**. 4. ed. Rio de Janeiro: Interciência, 2010.

MOGENSEN, P. K.; RISETH, A. N. Optim: A mathematical optimization package for Julia. **Journal of Open Source Software**, v. 3, n. 24, p. 01-03, 2018.

MOREIRA, D. A. **Pesquisa Operacional** – Curso introdutório. 2. ed. São Paulo: Cengage Learning, 2010.

PASSOS, E. J. P. F. **Programação Linear como instrumento da pesquisa operacional**. 1. ed. São Paulo: Atlas, 2008.

PEREIRA, C. D.; CUNHA, G. F.; SILVA, M. G. **A simulação na pesquisa operacional: uma revisão literária**. EEPA, Campo Mourão. Disponível em: http://www.fecilcam.br/anais/ix_eepa/data/uploads/3-pesquisa-operacional/3-03.pdf. Acesso em 10/10/2019.

PRADO, Darci. **Programação Linear**. 5. ed. Belo Horizonte: INDG, 2007.

RACKAUCKAS, C.; SCHILLING, T.; NIE, Q. Mean-Independent Noise Control of Cell Fates via Intermediate States. **IScience**, v. 3, pp. 11-20, 2018.

SANTANA, T. M. N.; GUIMARAES, R. R. S.; FONSECA, G. S.; GONCALVES, L. F.; BACHEGA, S. J. Aplicação do problema da árvore geradora mínima em uma empresa provedora de internet. In: SIMPÓSIO DE ENGENHARIA DE PRODUÇÃO, 2018, Catalão. **Anais...** Catalão: UFG-RC, 2018. p. 01-09.

SILVEIRA, J. F. P. **Problema do caixeiro viajante**. UFRGS, Rio Grande do Sul. Disponível em: http://www.mat.ufrgs.br/~portosil/caixeiro.html. Acesso em 17/10/2018.

SOUSA, L. T.; VEIGA, A. G.; CHAVES, M. F. C.; VAZ, M. V.; BACHEGA, S. J. Aplicação da otimização em redes em uma empresa do setor avícola. In: MACHADO, M. W. K. (Org.). **Engenharia de Produção:** What's Your Plan? Ponta Grossa: Atena Editora, 2019, p. 356-367.

SOUZA, L. C.; VIANA, G. G.; YOSHIDA, Y. M.; TELLAROLI, L. F.; BACHEGA, S. J. O problema do caminho mais curto aplicado em uma empresa do ramo de jornais. In: POISSON (Org.). **Gestão da Produção em Foco** - Volume 26. Belo Horizonte: Poisson, 2019, v. 26, p. 127-135.

TAHA, H. A. **Pesquisa Operacional**. 8. ed. São Paulo: Pearson Pretince Hall, 2008.

TEIXEIRA, N. B.; SILVA, C. F.; VIRGOLINO, G. A.; SOUZA, A.; BACHEGA, S. J. Aplicação da programação linear inteira em uma pequena fábrica de churrasqueiras pré-moldadas. In: SIMPÓSIO DE ENGENHARIA DE PRODUÇÃO, 24., 2017, Bauru. **Anais...** Bauru: UNESP, 2017. p. 01-12.