



UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS RUSSAS
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MARÍLIA CRISTINA DO CARMO VIANA

**PROBLEMA DE FORMAÇÃO DE MÚLTIPLOS TIMES COM MÚLTIPLAS
HABILIDADES: UMA ABORDAGEM HEURÍSTICA**

RUSSAS

2018

MARÍLIA CRISTINA DO CARMO VIANA

PROBLEMA DE FORMAÇÃO DE MÚLTIPLOS TIMES COM MÚLTIPLAS
HABILIDADES: UMA ABORDAGEM HEURÍSTICA

Trabalho de Conclusão de Curso apresentado ao
Curso de Graduação em Ciência da Computação
do Campus Russas da Universidade Federal do
Ceará, como requisito parcial à obtenção do
grau de bacharel em Ciência da Computação.

Orientadora: Prof. Ms. Tatiane Fernan-
des Figueiredo

RUSSAS

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

V668p Viana, Marília Cristina do Carmo.
Problema de Formação de Múltiplos Times com Múltiplas Habilidades: uma abordagem heurística /
Marília Cristina do Carmo Viana. – 2018.
61 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Russas,
Curso de Ciência da Computação, Russas, 2018.
Orientação: Profa. Ma. Tatiane Fernandes Figueiredo.

1. Formação de Múltiplos Times. 2. Otimização Combinatória. 3. Métodos Heurísticos. I. Título.
CDD 005

MARÍLIA CRISTINA DO CARMO VIANA

PROBLEMA DE FORMAÇÃO DE MÚLTIPLOS TIMES COM MÚLTIPLAS
HABILIDADES: UMA ABORDAGEM HEURÍSTICA

Trabalho de Conclusão de Curso apresentado ao
Curso de Graduação em Ciência da Computação
do Campus Russas da Universidade Federal do
Ceará, como requisito parcial à obtenção do
grau de bacharel em Ciência da Computação.

Aprovada em:

BANCA EXAMINADORA

Prof. Ms. Tatiane Fernandes
Figueiredo (Orientadora)
Universidade Federal do Ceará (UFC)

Prof. Dr. Bonfim Amaro Júnior
Universidade Federal do Ceará (UFC)

Prof. Dr. Pablo Luiz Braga Soares
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

À Deus, por ter me dado forças durante toda a minha jornada na graduação.

À toda a minha família, em especial à minha mãe Cristiana Cordeiro, meu pai João Valberto, meu irmão Luiz Alberto, minha avó Maria de Lourdes e meus tios João Edilson, Ana Maria, Paulo e Stephania, por todo o apoio e incentivo durante esses quatro anos, principalmente neste último ano. Também ao meu namorado Mailson Bandeira, por todo o apoio e por conseguir me distrair em momentos difíceis.

À todos os meus professores, em especial à minha orientadora Tatiane Fernandes, por toda a ajuda, incentivo, conselhos e conhecimentos repassados.

Aos meus amigos de turma Carlos Victor (Fred), Afonso Matheus, Hugo Venâncio, Vinícius Almeida, Isaac Rahel, Thomas Dillan, Isaías Ferreira, Erik Almeida, Marcos Paulo, Igor Mendes e Guilherme Sombra, que estiveram comigo nestes últimos quatro anos, enfrentando juntos, com muitos risos, as batalhas da graduação. Às minhas amigas Paloma Bispo e Francisca Tágila e ao nosso grupo TPM, as grandes mulheres dessa turma. Em especial, ao meu grande amigo Vinícius Almeida, um irmão que a faculdade me trouxe e minha eterna duplinha da UFC.

Às minhas amigas do tempo de escola Dalila Molinare, Isabel Lima, Carolina Oliveira e Beatriz Azevedo, que comemoraram junto comigo assim que passei na universidade e estão comemorando agora a conclusão dessa jornada.

À todos que contribuíram, direta e indiretamente, para a execução deste trabalho.

“Só se pode alcançar um grande êxito quando
nos mantemos fiéis a nós mesmos.”

(Friedrich Nietzsche)

RESUMO

Para atender os requisitos do mercado e se colocarem como fortes concorrentes no mesmo, muitas empresas precisam realizar projetos multidisciplinares simultaneamente. Nestes casos, além de questões de contratação e treinamento de pessoal, as relações sociais entre os membros da equipe assumem um papel importante que influencia diretamente o sucesso de um projeto, podendo afetar significativamente os seus resultados. Com intuito de auxiliar empresas a agrupar melhores equipes considerando os aspectos já mencionados, este trabalho apresenta o Problema de Formação de Múltiplos Times com Múltiplas Habilidades, como uma generalização do problema de Gutiérrez *et al.* (2016). Utilizando conceitos de Teoria dos Grafos para a representação dos dados de entrada do problema, é aplicado um método heurístico para buscar por soluções que satisfaçam as restrições de demanda de habilidades dos projetos, procurando também maximizar as relações sociais entre os membros de um mesmo time. São realizados testes computacionais com o Algoritmo Genético proposto e com os algoritmos heurísticos encontrados na literatura para o problema existente. Em termos de GAP (diferença entre o valor da solução heurística e o valor ótimo), o algoritmo proposto apresentou-se melhor em comparação aos da literatura, precisando, no entanto, de um pouco mais de tempo de execução. Para o problema generalizado proposto neste trabalho, foram geradas novas instâncias e realizados experimentos. Por não possuímos os valores ótimos para estas instâncias, não são apresentados os GAPs, e sim as qualidades das soluções e o tempo de execução. Mas, de acordo com a maneira de como é calculada a qualidade de uma solução para este problema, sabemos que o máximo valor possível sempre é 1 e as qualidades encontradas pela heurística proposta são próximas a este valor.

Palavras-chave: Formação de Múltiplos Times. Otimização Combinatória. Métodos Heurísticos.

ABSTRACT

To meet market requirements and to place themselves as strong competitors in it, many companies need to carry out multidisciplinary projects simultaneously. In these cases, in addition to personnel recruitment and training issues, the social relationships between team members take on an important role that directly influences the success of a project and can significantly affect its outcomes. In order to help companies to group better teams considering the aforementioned aspects, this work presents the Multiple Team Formation Problem with Multiple Skills, as a generalization of Gutiérrez *et al.* (2016)'s problem. Using Graph Theory concepts for problem's input data representation, a heuristic method is applied to search solutions that satisfy the projects' skills demand constraints, while also seeking to maximize the social relationships between team members. Computational tests are performed with the proposed Genetic Algorithm and with the heuristic algorithms found in the literature for the existing problem. In terms of GAP (difference between the value of the heuristic solution and the optimum value), the proposed algorithm was better compared to the literature, however, requiring a little more execution time. For the generalized problem proposed in this work, new instances were generated and experiments were performed. Because we do not have the optimal values for these instances, the GAPs are not presented, just the qualities of the solutions and the execution time. But, according to the way in which the quality of a solution to this problem is calculated, we know that the maximum possible value is always 1 and the qualities found by the proposed heuristic are close to this value.

Keywords: Multiple Team Formation. Combinatorial Optimization. Heuristic Methods.

LISTA DE FIGURAS

- Figura 1 – Rede $G(I)$ gerada a partir de uma instância I do Problema de Formação de Múltiplos Times com Múltiplas Habilidades ($PFMT_{MH}$). Solução destacada pelos arcos mais escuros, onde o valor em vermelho representa o fluxo que passa pelo arco. 37
- Figura 2 – Representação de uma solução viável X , de acordo com a instância I apresentada na Seção 4.2. 39
- Figura 3 – Representação dos *swaps* 1 e 2, de acordo com a instância I definida na Seção 4.2 e a solução X apresentada na Figura 2. As pessoas marcadas com as cores vermelha e amarela foram as escolhidas para os *swaps* 1 e 2, respectivamente. 43
- Figura 4 – Representação de uma solução viável X' , dada a instância I apresentada na Seção 4.2. 46
- Figura 5 – Representação da solução Y gerada através de um *crossover* entre as soluções X e X' , definidas nas figuras 2 e 4, respectivamente. A pessoa h_3 , marcada com a cor vermelha, está excedendo 100% de seu tempo de trabalho. Portanto, esta solução está inviável e, possivelmente, poderá ser corrigida com a mutação. 47
- Figura 6 – Representação da solução Y' gerada através de um *crossover* entre as soluções X e X' , definidas nas figuras 2 e 4, respectivamente. A pessoa h_4 , marcada com a cor vermelha, está excedendo 100% de seu tempo de trabalho. Portanto, esta solução está inviável e, possivelmente, poderá ser corrigida com a mutação. 47
- Figura 7 – Representação de uma mutação na solução Y , definida na Figura 5. A pessoa h_6 , marcada em azul, foi inserida no projeto p_2 , substituindo h_3 , que agora não está mais excedendo 100% de seu tempo. Assim, a solução Y torna-se viável. 48
- Figura 8 – GAPS das heurísticas da literatura e do Algoritmo Genético proposto. 56
- Figura 9 – Tempos de execução das heurísticas da literatura e do Algoritmo Genético proposto. 57

LISTA DE TABELAS

Tabela 1 – Valores dos parâmetros para cada classe de instâncias do Problema de Formação de Múltiplos Times (PFMT)	55
Tabela 2 – Análise comparativa entre as heurísticas, onde o GAP é a diferença entre o valor heurístico e o valor ótimo, e o tempo é medido em segundos.	56
Tabela 3 – Análise dos resultados obtidos com o Algoritmo Genético proposto para as novas instâncias geradas. A eficiência é calculada como definido na Subseção 4.4.1.2 e o tempo é medido em segundos.	58

LISTA DE ALGORITMOS

Algoritmo 1	– <i>Push(u,v)</i>	20
Algoritmo 2	– <i>Relabel(u)</i>	21
Algoritmo 3	– <i>Inicializar-Prefluxe(G,s)</i>	22
Algoritmo 4	– <i>Push-Relabel-Genérico(G)</i>	22
Algoritmo 5	– <i>Discharge(u)</i>	24
Algoritmo 6	– <i>Relabel-To-Front(G,s,t)</i>	25
Algoritmo 7	– Algoritmo Genético Clássico	28
Algoritmo 8	– $LS(N^1, X)$	31
Algoritmo 9	– Variabel Neighborhood Search k-habilidade	32
Algoritmo 10	– Simulated Annealing	33
Algoritmo 11	– Algoritmo Genético	40
Algoritmo 12	– Seleção de Operador Genético	50
Algoritmo 13	– <i>calculaProbSwap2(probOperadores, s)</i>	51
Algoritmo 14	– <i>calculaProbCrossover(probOperadores, s)</i>	52

LISTA DE ABREVIATURAS E SIGLAS

<i>PFMT_{MH}</i>	Problema de Formação de Múltiplos Times com Múltiplas Habilidades
LS	Local Search
PFMT	Problema de Formação de Múltiplos Times
PFT	Problema de Formação de Time
PFTS	Problema de Formação de Times Sociotécnicos
PO	Pesquisa Operacional
SA	Simulated Annealing
VNS	Variable Neighborhood Search

SUMÁRIO

1	INTRODUÇÃO	14
1.1	Objetivos	15
<i>1.1.1</i>	<i>Objetivo Geral</i>	<i>15</i>
<i>1.1.2</i>	<i>Objetivos Específicos</i>	<i>15</i>
1.2	Metodologia	15
1.3	Organização do Trabalho	16
2	REFERENCIAL TEÓRICO	17
2.1	Teoria dos Grafos	17
2.2	Problema de Fluxo Máximo	18
<i>2.2.1</i>	<i>Redes Residuais</i>	<i>19</i>
<i>2.2.2</i>	<i>Algoritmo de Goldberg</i>	<i>19</i>
<i>2.2.3</i>	<i>Algoritmo Relabel-To-Front</i>	<i>22</i>
2.3	Pesquisa Operacional	25
2.4	Heurísticas	26
<i>2.4.1</i>	<i>Meta-heurísticas</i>	<i>26</i>
<i>2.4.1.1</i>	<i>Algoritmos Genéticos</i>	<i>27</i>
3	TRABALHOS RELACIONADOS	29
3.1	Problema de Formação de Time utilizando Redes Sociais	29
3.2	Problema de Formação de Múltiplos Times	29
<i>3.2.1</i>	<i>Variable Neighborhood Search</i>	<i>30</i>
<i>3.2.2</i>	<i>Simulated Annealing</i>	<i>32</i>
3.3	Problema de Formação de Times Sociotécnicos	33
4	O PROBLEMA DE FORMAÇÃO DE MÚLTIPLOS TIMES COM MÚLTIPLAS HABILIDADES	35
4.1	Definição do problema	35
4.2	Geração de Solução Viável	37
4.3	Representação da Solução	38
4.4	Limite Superior	39
<i>4.4.1</i>	<i>Heurística baseada em Algoritmos Genéticos</i>	<i>39</i>
<i>4.4.1.1</i>	<i>Geração da População Inicial</i>	<i>39</i>

4.4.1.2	<i>Aptidão de uma Solução</i>	40
4.4.1.3	<i>Crítérios de Parada</i>	41
4.4.1.4	<i>Operadores Genéticos</i>	41
4.4.1.4.1	<i>Swap 1</i>	42
4.4.1.4.2	<i>Swap 2</i>	44
4.4.1.4.3	<i>Crossover</i>	45
4.4.1.4.4	<i>Mutação</i>	47
4.4.1.5	<i>Seleção de Operador Genético</i>	49
4.4.1.6	<i>Seleção de Indivíduos para Reprodução</i>	51
4.4.1.7	<i>Seleção da Nova População</i>	53
5	RESULTADOS	54
5.1	Configuração do Ambiente Computacional	54
5.2	Instâncias	54
5.2.1	<i>Instâncias do PFMT</i>	54
5.2.2	<i>Instâncias do PFMT_{MH}</i>	55
5.3	Estudo Comparativo entre os Algoritmos	55
6	CONCLUSÕES E TRABALHOS FUTUROS	59
	REFERÊNCIAS	60

1 INTRODUÇÃO

Para aumentar sua capacidade de produção e atender aos requisitos do mercado, colocando-se como fortes concorrentes no mesmo, a maioria das empresas realiza múltiplos projetos de forma simultânea (FITZPATRICK; ASKIN, 2005). Nesse contexto, a gerência desses projetos e dos recursos utilizados, principalmente os recursos humanos, é de suma importância, uma vez que uma das despesas operacionais mais relevantes para a execução de projetos multidisciplinares é a contratação e treinamento de pessoas (MAURER, 2010).

Há alguns anos, as empresas costumavam atribuir tarefas a indivíduos isolados (LIU; YANG, 2011), enquanto que nos dias atuais, o que costuma ocorrer são grupos de pessoas, reunidas em times, trabalhando juntas para desenvolver um produto ou serviço (BAILEY, 1997). Nesse contexto, as relações sociais entre os membros de um time assumem um papel importante, influenciando diretamente na produtividade da equipe e, por conseguinte, no sucesso do projeto desempenhado (BALLESTEROS-PÉREZ *et al.*, 2012).

O trabalho de Lappas *et al.* (2009) estudou o caso de formação de um único time, a partir de um conjunto de pessoas disponíveis, cada qual com um conjunto de habilidades (áreas de experiência). Para o caso de formação de múltiplos times, Gutiérrez *et al.* (2016) e Campêlo *et al.* (2018) propuseram problemas semelhantes, onde o primeiro considera que as pessoas, com apenas uma habilidade cada uma, podem participar de mais de um time, enquanto o segundo considera pessoas com múltiplas habilidades, mas que só podem estar presentes em uma única equipe. Ambos os problemas buscam maximizar a afinidade entre os membros de um mesmo time.

Nesse contexto, o presente trabalho propõe o Problema de Formação de Múltiplos Times com Múltiplas Habilidades ($PFMT_{MH}$), como uma união dos problemas da literatura citados anteriormente, possuindo, assim, as seguintes características: múltiplos times, pessoas com múltiplas habilidades e frações de dedicação de pessoas. Dado um conjunto de projetos e um conjunto de pessoas disponíveis, o problema consiste em alocar essas pessoas aos projetos, de maneira a atender as demandas de habilidades dos mesmos, além de maximizar a afinidade entre os membros de um time. Utilizando conceitos de Teoria dos Grafos para a representação dos dados de entrada do problema, foi escolhida, para a resolução deste, a meta-heurística Algoritmos Genéticos (HOLLAND, 1975).

O algoritmo proposto neste trabalho também é aplicável ao problema de Gutiérrez *et al.* (2016), uma vez que o $PFMT_{MH}$ trata-se de uma generalização do mesmo. Por este motivo,

foram realizados experimentos computacionais a fim de comparar o algoritmo proposto com os encontrados na literatura (GUTIÉRREZ *et al.*, 2016; FIGUEIREDO; CAMPÊLO, 2018), para o problema de Gutiérrez *et al.* (2016), onde as pessoas possuem apenas uma habilidade. Para o $PFMT_{MH}$, foram geradas novas instâncias, onde as pessoas possuem múltiplas habilidades, e realizados testes com o algoritmo proposto.

1.1 Objetivos

1.1.1 Objetivo Geral

Obter soluções para o Problema de Formação de Múltiplos Times com Múltiplas Habilidades através de métodos heurísticos.

1.1.2 Objetivos Específicos

- Implementar um algoritmo heurístico para o Problema de Formação de Múltiplos Times com Múltiplas Habilidades;
- Criar um gerador de instâncias para o problema, baseado em problemas semelhantes existentes na literatura;
- Efetuar uma análise estatística do desempenho do algoritmo heurístico apresentado.

1.2 Metodologia

O presente trabalho será desenvolvido com base nas fases padronizadas da área da Pesquisa Operacional, apresentadas a seguir:

1. Realização de um estudo aprofundado das áreas de Teoria dos Grafos e Pesquisa Operacional com foco em técnicas de programação heurística;
2. Definição do problema de interesse;
3. Desenvolvimento de um procedimento computacional a fim de derivar boas soluções para o problema utilizando técnicas meta-heurísticas;
4. Análise dos resultados do algoritmo proposto e aprimoramento conforme necessário.

O algoritmo heurístico proposto será alimentado com as instâncias obtidas da literatura e do gerador criado, a fim de comparar seu desempenho com outros algoritmos existentes. O estudo será explicativo e seguirá uma abordagem algorítmica e de simulação. Neste método, um

algoritmo heurístico será utilizado para auxílio na tomada de decisões e resolução do Problema de Formação de Múltiplos Times com Múltiplas Habilidades.

1.3 Organização do Trabalho

O restante deste trabalho divide-se em 5 capítulos. O Capítulo 2 apresenta os conceitos fundamentais para entendimento do problema e dos algoritmos abordados nas subseções 3.2.1, 3.2.2 e 4.4.1. No Capítulo 3 são apresentados os trabalhos relacionados ao problema proposto nesta monografia. O Capítulo 4 apresenta formalmente o Problema de Formação de Múltiplos Times com Múltiplas Habilidades, assim como o passo a passo do algoritmo proposto para sua resolução. São apresentados, no Capítulo 5, os experimentos computacionais e os resultados obtidos. Por fim, no Capítulo 6, são apresentadas as conclusões e os trabalhos futuros.

2 REFERENCIAL TEÓRICO

Para entendimento do problema abordado neste trabalho, faz-se necessário definir alguns conceitos de Teoria dos Grafos na Seção 2.1. Um algoritmo de fluxo máximo foi utilizado para a geração de uma solução viável para o problema aqui abordado e, por este motivo, a Seção 2.2 aborda o problema de fluxo máximo, bem como o algoritmo de fluxo utilizado. Por fim, para entendimento da meta-heurística utilizada neste trabalho, as seções 2.3 e 2.4 abordam alguns conhecimentos iniciais da área de Pesquisa Operacional e de métodos heurísticos, respectivamente.

2.1 Teoria dos Grafos

Um *grafo* $G = (V(G), E(G))$ consiste de um conjunto finito não vazio $V(G)$ de elementos, denominados *vértices*, e um conjunto de pares não-ordenados de elementos distintos de $V(G)$, denominados *arestas*, isto é, $E(G) \subseteq \{\{u, v\} | u, v \in V(G), u \neq v\}$. Algumas vezes, quando o grafo estiver claro pelo contexto, utilizamos V e E para $V(G)$ e $E(G)$ respectivamente, a fim de simplificar a notação. Se $e = \{u, v\}$ é uma aresta, dizemos que e *incide em* u e *em* v ; que u e v *são seus extremos*; e que u e v *são adjacentes*. Para simplificar a notação, denotamos uma aresta $e = \{u, v\}$ por uv .

A *vizinhança* ou *adjacência* de um vértice v em um grafo G é o conjunto contendo todos os vértices adjacentes a v . A vizinhança é frequentemente denotada $N_G(v)$ ou simplesmente $N(v)$ quando o grafo está claro pelo contexto. A vizinhança de um subconjunto de vértices $W \subseteq V$ é $N(W) = \bigcup_{v \in W} N(v)$. Se V' é um subconjunto de vértices em $V(G)$, denotamos por $\delta(V')$ o conjunto de arestas em $E(G)$ com um extremo em V' e o outro em $V \setminus V'$. Se G e G' são dois grafos tais que $V(G') \subseteq V(G)$ e $E(G') \subseteq E(G)$, então G' é dito um *subgrafo* de G e escrevemos $G' \subseteq G$.

Um *grafo orientado* (*digrafo*) $D = (V, A)$ consiste de um conjunto finito não-vazio $V(D)$ de elementos, denominados *vértices* ou *nós*, e um conjunto $A(D)$ de elementos, denominados *arcos* ou *arestas orientadas*, que são pares ordenados de vértices distintos, isto é, $A(D) \subseteq \{(u, v) | u, v \in V(D), u \neq v\}$. Se $a = (u, v)$ é um arco, dizemos que a *incide em* u e *em* v ; que u e v *são seus extremos*. Também nos referimos a u como *início* ou *origem* e a v como *término* ou *destino* do arco a . No problema de fluxo máximo, abordado na seção 2.2, um grafo orientado é chamado de *rede*.

Dado um vértice $v \in V(D)$ denotamos por $\delta^+(v)$ o conjunto de arcos que possuem o vértice v como início e $\delta^-(v)$ o conjunto de arcos que possuem o vértice v como término. Se $\delta^-(v) = \emptyset$, v é dito uma *fonte*; Se $\delta^+(v) = \emptyset$, v é dito um *sumidouro*. Se V' é um subconjunto de vértices em $V(D)$, denotamos por $\delta^+(V')$ o conjunto dos arcos de D com início em V' e término em $V \setminus V'$. Um conjunto da forma $\delta^+(V')$, onde $\emptyset \neq V' \subsetneq V$, é dito um *corte* em D . Se v' é um vértice em V' e v um vértice em $V \setminus V'$, então dizemos que $\delta^+(V')$ é um (v', v) -*corte*. Usamos também a notação $[V', V \setminus V']$ para representar um corte $\delta^+(V')$.

Todos os conceitos definidos para grafos que não envolvem a noção de orientação se aplicam analogamente para os grafos orientados. Embora usemos preferencialmente a notação $D = (V, A)$ para um grafo orientado, vamos usar também $G = (V, E)$, quando não houver prejuízo para o entendimento e, neste caso, E será um conjunto de arcos.

2.2 Problema de Fluxo Máximo

Dada uma rede $G = (V, E)$, com um *vértice fonte* $s \in V(G)$ e um *vértice sorvedouro* $t \in V(G)$, onde cada arco $(u, v) \in E(G)$ possui um número não-negativo $c_{uv} \in \mathbb{R}$ associado à ele. Este número é chamado de *capacidade* de (u, v) . O Problema de Fluxo Máximo consiste em encontrar um fluxo de intensidade máxima, que sai de s em direção a t , respeitando as restrições de capacidade e de conservação de fluxo.

Para uma melhor compreensão das restrições de conservação de fluxo e capacidade, podemos definir uma variável $x_{uv} \in \mathbb{R}$ que representa o fluxo que passa em um arco (u, v) . A restrição de conservação de fluxo, apresentada na equação 2.2, diz que o fluxo total que entra em um vértice, exceto s ou t , deve ser igual ao fluxo total que sai deste. Já a restrição de capacidade, apresentada na equação 2.3, diz que o fluxo que passa em um arco (u, v) deve ser não negativo e não deve exceder a capacidade c_{uv} , para cada arco $(u, v) \in E(G)$. Por fim, $f \in \mathbb{R}$ é o valor do fluxo que se deseja maximizar.

$$\max f \tag{2.1}$$

$$\text{s.a: } \sum_{v \in V(G)} x_{vu} - \sum_{v \in V(G)} x_{uv} = \begin{cases} -f, & \text{se } u = s \\ 0, & \text{se } u \neq s, t \\ f, & \text{se } u = t \end{cases} \quad \forall u \in V(G) \tag{2.2}$$

$$0 \leq x_{uv} \leq c_{uv} \quad \forall (u, v) \in E(G) \tag{2.3}$$

$$x_{uv} \in \mathbb{Z}^+ \quad \forall (u, v) \in E(G) \quad (2.4)$$

2.2.1 Redes Residuais

Dada uma rede G e um fluxo f , uma *rede residual* $G_f = (V, E_f)$ contém arcos com capacidades que representam como podemos alterar o fluxo nos arcos de G . Um arco $(u, v) \in E(G)$ pode admitir uma quantidade de fluxo adicional igual à capacidade do arco menos o fluxo nesse arco, ou seja, $c_f(u, v) = c_{uv} - x_{uv}$, onde $c_f(u, v)$ é chamado de *capacidade residual* do arco (u, v) . Se $c_f(u, v) > 0$, significa que o arco $(u, v) \in E(G)$ pode receber mais fluxo e, então, este arco é colocada em G_f e chamado de *arco residual*. A rede residual G_f também pode conter arcos que não estão presentes em G . Ao aplicar um algoritmo em um fluxo com o objetivo de aumentar o fluxo total, pode ser necessário reduzir o fluxo em um determinado arco. Para representar uma possível diminuição de fluxo em um arco $(u, v) \in E(G)$, um arco invertido (v, u) é inserido em G_f com capacidade residual $c_f(v, u) = x_{uv}$. Um arco invertido $(v, u) \in E_f(G_f)$ permite que um algoritmo envie de volta o fluxo que já enviou ao longo de um arco $(u, v) \in E(G)$, operação conhecida como *cancelamento*.

2.2.2 Algoritmo de Goldberg

O algoritmo de fluxo máximo genérico de Goldberg e Tarjan (1988), conhecido como algoritmo *push-relabel* (empurrar-remarcas), possui uma implementação simples que é executada em tempo polinomial $O(V^2E)$ (CORMEN *et al.*, 2012). Este algoritmo não mantém a propriedade de conservação de fluxo durante toda a sua execução. Em vez disso, ele mantém uma função $p : V \times V \rightarrow \mathbb{R}$, conhecida como *pré-fluxo*, que satisfaz a restrição de capacidade e o seguinte relaxamento da restrição de conservação de fluxo:

$$\sum_{v \in V(G)} x_{vu} - \sum_{v \in V(G)} x_{uv} \geq 0 \quad (2.5)$$

para todo $u \in V(G) \setminus \{s\}$. Ou seja, o fluxo que entra em um vértice pode exceder o fluxo que sai deste. Esta quantidade

$$e(u) = \sum_{v \in V(G)} x_{vu} - \sum_{v \in V(G)} x_{uv} \quad (2.6)$$

é denominada *excesso de fluxo* no vértice u . Para $u \in V(G) \setminus \{s, t\}$, se $e(u) > 0$, dizemos que o vértice u está *transbordando*.

O algoritmo *push-relabel* executa duas operações básicas: empurrar o excesso de fluxo de um vértice que está transbordando até um de seus vizinhos (*push*) e remarcar um vértice (*relabel*). Estas operações dependem de um termo conhecido como *altura* de um vértice. Uma função $h : V \rightarrow \mathbb{N}$ é uma função altura se $h(s) = |V|$, $h(t) = 0$ e

$$h(u) \leq h(v) + 1 \quad (2.7)$$

para todo arco residual $(u, v) \in E_f(G_f)$.

Algoritmo 1: *Push*(u, v)

Entrada: $u, v \in V(G)$

Resultado: Empurra $\Delta_f(u, v) = \min(e(u), c_f(u, v))$ unidades de fluxo de u até v .

```

1 início
2    $\Delta_f(u, v) = \min(e(u), c_f(u, v));$ 
3   se  $(u, v) \in E(G)$  então
4     |    $x_{uv} = x_{uv} + \Delta_f(u, v);$ 
5   fim
6   senão
7     |    $x_{uv} = x_{uv} - \Delta_f(u, v);$ 
8   fim
9    $e(u) = e(u) - \Delta_f(u, v);$ 
10   $e(v) = e(v) + \Delta_f(u, v);$ 
11 fim

```

A operação *Push*(u, v) é aplicada se $e(u) > 0$, $c_f(u, v) > 0$ e $h(u) = h(v) + 1$. O Algoritmo 1 atualiza o pré-fluxo f e os excessos de fluxo para u e v , onde $\Delta_f(u, v)$ é uma variável temporária que armazena a quantidade de fluxo que pode ser empurrada de u para v . Como o vértice u tem um excesso de fluxo $e(u) > 0$ e a capacidade residual do arco (u, v) é positiva, pode-se aumentar o fluxo de u a v em $\Delta_f(u, v) = \min(e(u), c_f(u, v))$ unidades de fluxo, sem que $e(u)$ torne-se negativo nem que a capacidade c_{uv} seja excedida. A linha 2 calcula o valor $\Delta_f(u, v)$. Na linha 4, o fluxo na aresta (u, v) é aumentado porque este arco residual também é um arco original em G . A linha 6 diminui o fluxo no arco (u, v) porque este arco residual é o inverso de um arco em G . Enfim, as linhas 9-10 atualizam os excessos de fluxo nos vértices u e v .

Push(u, v) é dito um *empurrão saturador* se o arco (u, v) na rede residual tornar-se saturado ($c_f(u, v) = 0$); caso contrário, é dito um *empurrão não saturador*. Se um arco torna-se

saturado, ele sai da rede residual, uma vez que não é possível aumentar a quantidade de fluxo que passa por ele.

A operação $Relabel(u)$ é aplicada se $e(u) > 0$ e $h(u) \leq h(v)$ para todo arco $(u, v) \in E_f(G_f)$. Isto é, um vértice u que está transbordando pode ser remarcado se, para todo vértice v adjacente a ele em G_f , o fluxo não puder ser empurrado porque a altura de v não é menor do que a altura de u . O Algoritmo 2, na linha 2, dá a u a maior altura permitida pela restrição imposta à função altura.

Algoritmo 2: $Relabel(u)$

Entrada: $u \in V(G)$

Resultado: Aumenta a altura de u

1 **início**

2 $h(u) = 1 + \min\{h(v) : (u, v) \in E_f(G_f)\};$

3 **fim**

O Algoritmo 3 cria um pré-fluxo inicial. Nas linhas 2-5, a altura e o excesso de cada vértice $v \in V(G)$ são inicializados com o valor 0. O fluxo de cada arco $(u, v) \in E(G)$ é inicializado com o valor 0, nas linhas 6-8. A linha 9 inicializa a altura do vértice s . Nas linhas 10-14, cada arco que sai de s recebe um fluxo igual à capacidade total daquele arco e os excessos dos vértices s e seus adjacentes são recalculados.

Por fim, o Algoritmo 4 realiza a inicialização de um pré-fluxo, na linha 2, seguida de uma sequência de operações $push$ e $relabel$, executadas sem qualquer ordem definida, nas linhas 3-5. Ao final da execução deste algoritmo, um fluxo máximo para G foi calculado.

Algoritmo 3: Inicializar-Prefluxo(G, s)

Entrada: Um grafo $G = (V, E)$ e $s \in V(G)$ **Resultado:** Cria um pré-fluxo inicial p

```

1 início
2   para  $v \in V(G)$  faça
3      $h(v) = 0$ ;
4      $e(v) = 0$ ;
5   fim
6   para  $(u, v) \in E(G)$  faça
7      $x_{uv} = 0$ ;
8   fim
9    $h(s) = |V(G)|$ ;
10  para  $v \in V(G)$  adjacente à  $s$  faça
11     $x_{sv} = c_{sv}$ ;
12     $e(v) = c_{sv}$ ;
13     $e(s) = e(s) - c_{sv}$ ;
14  fim
15 fim

```

Algoritmo 4: Push-Relabel-Genérico(G)

Entrada: Um grafo $G = (V, E)$ **Resultado:** Um fluxo máximo para G

```

1 início
2   Inicializar-Prefluxo( $G, s$ );
3   enquanto existir uma operação push ou relabel aplicável faça
4     selecionar uma operação push ou relabel aplicável e executá-la;
5   fim
6 fim

```

2.2.3 Algoritmo Relabel-To-Front

No método *Push-Relabel*, apresentado no Algoritmo 4, pode-se aplicar as operações básicas em absolutamente qualquer ordem. O algoritmo *relabel-to-front* é um algoritmo *push-relabel* cujo tempo de execução é $O(V^3)$, que é assintoticamente no mínimo tão bom quanto $O(V^2E)$ e até melhor para redes densas (CORMEN *et al.*, 2012).

O algoritmo *relabel-to-front* mantém uma lista dos vértices da rede. Começando na frente, o algoritmo percorre a lista, selecionando repetidamente um vértice u que está transbordando e depois "descarregando-o", ou seja, executando as operações básicas *push* e *relabel* até que u não tenha mais um excesso de fluxo positivo. Sempre que um vértice é remarcado, este é deslocado para a frente da lista (por isso o nome "*relabel-to-front*") e o algoritmo inicia novamente sua varredura.

Para o entendimento deste algoritmo, faz-se necessária a noção de arcos admissíveis. Se $G = (V, E)$ é uma rede de fluxo com fonte s e sorvedouro t , f é um pré-fluxo em G e h é uma função altura, diz-se que (u, v) é um *arco admissível* se $c_f(u, v) > 0$ e $h(u) = h(v) + 1$. Caso contrário, (u, v) é *inadmissível*. A *rede admissível* é $G_{f,h} = (V, E_{f,h})$, onde $E_{f,h}$ é o conjunto dos arcos admissíveis. Em outras palavras, a rede admissível contém os arcos pelos quais pode-se empurrar o fluxo.

O Algoritmo 5 recebe um vértice u como entrada e u é *descarregado* empurrando todo o seu excesso de fluxo por arcos admissíveis para vértices vizinhos (linha 10) e remarcando u conforme necessário para transformar os arcos que saem de u em arcos admissíveis (linha 5). O atributo $u.N$ é a *lista de vizinhos* do vértice u . Dada uma rede de fluxo $G = (V, E)$, um vértice $v \in V$ aparece na lista de vizinhos de um vértice $u \in V$ se $(u, v) \in E$ ou $(v, u) \in E$. O atributo $u.N.inicio$ aponta para o primeiro vértice da lista $u.N$ e $v.próximo$ aponta para o vértice que vem depois de v na lista de vizinhos; se v é o último vértice na lista, esse ponteiro é *NIL*. Por fim, o atributo $u.atual$ aponta para o vértice que está sendo considerado em $u.N$ no momento em

questão.

Algoritmo 5: $Discharge(u)$

Entrada: Um vértice u

Resultado: Todo o excesso de u é eliminado.

```

1 início
2   enquanto  $e(u) > 0$  faça
3      $v = u.atual$ ;
4     se  $v == NIL$  então
5        $Relabel(u)$ ;
6        $u.atual = u.N.inicio$ ;
7     fim
8     senão se  $c_f(u, v) > 0$  e  $h(u) = h(v) + 1$  então
9        $Push(u, v)$ ;
10    fim
11    senão
12       $u.atual = v.próximo$ 
13    fim
14  fim
15 fim
```

O Algoritmo 6 funciona da seguinte maneira. A linha 2 inicializa o pré-fluxo e as alturas para os mesmos valores que no Algoritmo 4. A linha 3 inicializa a lista L para conter todos os vértices que potencialmente podem estar transbordando, em qualquer ordem. Nas linhas 4-6, o ponteiro $atual$ de cada vértice u é inicializado como o primeiro vértice na lista de vizinhos de u . Nas linhas 8-15, a lista L é percorrida, descarregando os vértices, começando do primeiro vértice da lista, que foi atribuído a u na linha 7. A cada passagem pelo laço, a linha 10 descarrega um vértice u . Se u foi remarcado pelo procedimento $Discharge$, a linha 12 o desloca para a frente da lista L . Pode-se determinar se u foi remarcado comparando a sua altura antes da operação, que foi gravada na variável $altura-antiga$ na linha 9, com sua altura depois da operação, na linha 11. A linha 14 obriga a próxima iteração do laço a usar o vértice que vem depois de u na lista L . Se u foi deslocado para a frente da lista, o vértice usado na próxima iteração é aquele que vem

depois de u em sua nova posição na lista. Isto se dá para manter a lista L em ordem topológica.

Algoritmo 6: *Relabel-To-Front*(G,s,t)

Entrada: Uma rede $G = (V,E)$; um vértice fonte $s \in V$ e um vértice sorvedouro $t \in V$

Resultado: Um fluxo máximo para G .

```

1 início
2   Inicializar-Prefluxo( $G,s$ );
3    $L = V(G) - \{s,t\}$ , em qualquer ordem;
4   para cada vértice  $u \in V(G) - \{s,t\}$  faça
5     |  $u.atual = u.N.inicio$ ;
6   fim
7    $u = L.inicio$ ;
8   enquanto  $u \neq NIL$  faça
9     |  $altura-antiga = h(u)$ ;
10    |  $Discharge(u)$ ;
11    | se  $h(u) > altura-antiga$  então
12      | mover  $u$  para a frente da lista  $L$ ;
13    | fim
14    |  $u = u.próximo$ ;
15  fim
16 fim
```

2.3 Pesquisa Operacional

A Pesquisa Operacional (PO) é aplicada a problemas envolvendo como conduzir e coordenar as operações (atividades) em uma organização (HILLIER; LIEBERMAN, 2013). Ela é utilizada para analisar sistemas complexos do mundo real, com o objetivo de melhorar e/ou otimizar seus desempenhos. A PO possui aplicações em diversas áreas, como manufatura, medicina, transportes, serviços públicos e telecomunicações, por exemplo.

A fim de solucionar problemas complexos do mundo real, a PO busca por soluções que maximizem ou minimizem uma função objetivo. Ela busca a melhor solução para o problema considerado, conhecida como solução ótima (HILLIER; LIEBERMAN, 2013). Uma solução ótima é uma solução viável de custo ótimo, ou seja, é a melhor dentre todas as soluções. Uma solução viável é aquela em que nenhuma restrição do problema é violada. Se pelo menos uma restrição é violada, a solução é dita inviável.

Alguns problemas do mundo real são muito complexos e possuem grandes espaços

de soluções e, portanto, pode não ser possível encontrar uma solução ótima em tempo computacional razoável. Os métodos heurísticos são utilizados para encontrar, nesses casos, uma solução viável razoavelmente boa. O foco de estudo deste trabalho consiste na aplicação de um método heurístico para resolução do Problema de Formação de Múltiplos Times com Múltiplas Habilidades.

2.4 Heurísticas

Heurísticas são procedimentos que, provavelmente, encontrarão uma boa solução viável, mas não necessariamente uma solução ótima, para o problema específico que se deseja tratar (HILLIER; LIEBERMAN, 2013). Não há nenhuma garantia sobre a qualidade da solução encontrada, no entanto, um algoritmo heurístico bem elaborado, geralmente, é capaz de fornecer uma solução próxima do ótimo, ou concluir que tais soluções não existem.

Dentre as várias aplicabilidades dos algoritmos heurísticos, destaca-se o seu uso para resolução de problemas de otimização, que em sua grande maioria, são caracterizados por grandes espaços de soluções, dificultando assim, a obtenção de uma solução de custo ótimo em tempo computacional razoável. Por esse motivo, as heurísticas abrem mão da otimalidade e buscam as melhores soluções possíveis de um problema em tempo viável.

As heurísticas podem ser classificadas em algumas categorias genéricas: Heurísticas Construtivas, Heurísticas de Busca em Vizinhança, Heurísticas Sistemáticas, Heurísticas Híbridas e Meta-Heurísticas, onde esta última é o foco de estudo deste trabalho.

2.4.1 Meta-heurísticas

Segundo Luke (2013), Meta-Heurística é a subárea principal de otimização estocástica, a classe geral de algoritmos e técnicas que utilizam algum grau de aleatoriedade para encontrar soluções ótimas, ou tão ótimas quanto possível, para problemas que possuem um espaço de soluções muito grande. Elas são aplicadas em problemas sobre os quais há poucas informações: não se sabe de antemão como é a aparência de uma solução ótima e algoritmos de força-bruta não são viáveis. Contudo, dada uma solução para o problema, pode-se testar e verificar sua otimalidade.

As meta-heurísticas são métodos heurísticos genéricos que podem lidar com qualquer problema de otimização. Entretanto, para serem aplicadas de forma eficiente e eficaz, dependem

de conhecimentos específicos do problema que se deseja tratar. O objetivo deste trabalho reside na aplicação da meta-heurística Algoritmos Genéticos (HOLLAND, 1975) para resolução do Problema de Formação de Múltiplos Times com Múltiplas Habilidades.

2.4.1.1 Algoritmos Genéticos

O termo Algoritmo Genético foi utilizado pela primeira vez por Holland (1975). A motivação para esta meta-heurística é uma analogia ao processo de evolução natural da biologia, onde as populações evoluem de acordo com os princípios de seleção natural e os indivíduos mais bem adaptados ao ambiente têm maiores chances de sobreviver e de se reproduzir. Os principais termos da genética simulados nos algoritmos genéticos são:

- População: conjunto de indivíduos (soluções);
- Cromossomo (ou indivíduo): conjunto de genes. Cada indivíduo representa uma solução do problema;
- Gene: menor unidade de um cromossomo. Cada gene representa uma variável de solução do problema;
- Cruzamento (ou *crossover*): processo de combinar os genes dos cromossomos pais a fim de gerar novos indivíduos, garantindo assim a diversidade e a evolução populacional;
- Mutação: processo de alteração aleatória de genes, seja no seu conteúdo ou na sua localização. Como na biologia, esta anomalia possui baixa probabilidade de ocorrência;
- Função de aptidão (ou *fitness*): função que avalia quão bom é um indivíduo. Em geral, a aptidão é o valor da solução.

O Algoritmo 7 apresenta o Algoritmo Genético Clássico. Primeiramente, é gerada a população inicial e avaliada a aptidão de cada indivíduo. Então, enquanto o critério de parada não for atendido, são executadas operações de reprodução e mutação. Os cromossomos são selecionados para reprodução de forma aleatória ou por sua aptidão (Método da Roleta). Após o cruzamento, em pontos de *crossover* aleatórios, a mutação pode ocorrer em alguns dos cromossomos gerados. Por fim, são selecionados os melhores indivíduos para a próxima geração, ou seja, para a próxima iteração do algoritmo. Estes passos são repetidos até que um determinado critério de parada seja atingido. Os critérios de parada mais comuns são: tempo de processamento (MENDES, 2013), quantidade de gerações (CORTES, 2004), um determinado valor da solução (definido a priori) foi atingido (SOUZA *et al.*, 2006) e convergência (não há uma melhora considerável na solução durante um determinado número de gerações) (SILVA, 2001).

Algoritmo 7: Algoritmo Genético Clássico

Entrada: Instância do problema.

Saída: Melhor solução encontrada.

```
1 início
2   Gerar população inicial;
3   Avaliar aptidão da população;
4   enquanto critério de parada não for atendido faça
5     Selecionar indivíduos para reprodução;
6     Reprodução, de acordo com a taxa de reprodução;
7     Mutação, de acordo com a taxa de mutação;
8     Selecionar nova população;
9   fim
10  retorna melhor solução encontrada.
11 fim
```

Conhecer o problema que se deseja resolver e suas propriedades é essencial para determinar como ajustar esses parâmetros e quais critérios de parada utilizar.

3 TRABALHOS RELACIONADOS

Neste capítulo, são descritos os trabalhos da literatura mais relevantes para contextualizar o problema proposto nesta monografia. As seções 3.1, 3.2 e 3.3 apresentam problemas semelhantes encontrados na literatura.

3.1 Problema de Formação de Time utilizando Redes Sociais

O trabalho de Lappas *et al.* (2009) foi o primeiro a considerar o Problema de Formação de Time (PFT) na presença de uma rede social de pessoas. Dada uma tarefa T , um conjunto de pessoas X com diferentes habilidades e uma rede social G , que exibe a compatibilidade entre essas pessoas, este problema consiste em encontrar um subconjunto $X' \subseteq X$ de pessoas para desempenhar a tarefa. O PFT foi demonstrado ser NP-Difícil por Lappas *et al.* (2009).

A rede social $G(X, E)$ é um grafo ponderado e não-orientado onde cada vértice representa uma pessoa em X e os pesos em uma aresta ij representam a compatibilidade entre as pessoas i e j . A tarefa T requer um conjunto de habilidades, onde cada uma delas deve ser exibida por pelo menos uma pessoa em X' . Além de cumprir esta restrição de demanda, o PFT também objetiva maximizar a compatibilidade entre as pessoas de X' , uma vez que a produtividade de um time é fortemente influenciada pelas relações sociais entre os seus membros (BALLESTEROS-PÉREZ *et al.*, 2012). Por fim, para resolução deste problema, foram utilizados algoritmos enumerativos: *Rarest First*, *CoverSteiner* e *EnhancedSteiner*.

3.2 Problema de Formação de Múltiplos Times

O PFMT (GUTIÉRREZ *et al.*, 2016) é uma generalização do PFT (LAPPAS *et al.*, 2009), onde duas novas dimensões são adicionadas: múltiplos projetos e frações de dedicação de pessoas, ou seja, uma pessoa pode dedicar frações de seu tempo de trabalho a diferentes projetos.

Este problema utiliza sociometria, uma técnica aplicada para descrever a saúde de um grupo, e possui, como uma das entradas principais, uma matriz sociométrica (MORENO, 1941). Esta é uma matriz $n \times n$, sendo n o número de pessoas disponíveis, que contém a predisposição de cada pessoa i em trabalhar com a pessoa j . Os elementos desta matriz podem assumir três possíveis valores: o valor $+1$ é escolhido se a pessoa i está disposta a trabalhar com a pessoa j , o valor -1 se i prefere não trabalhar com j , e o valor 0 se i é neutro em relação a trabalhar com j .

Dado um conjunto de n pessoas, cada uma possuindo apenas uma habilidade, e um

conjunto de m projetos, que demandam uma quantidade específica de pessoas por habilidade, o PFMT consiste em formar m times, de forma que satisfaçam as demandas de habilidades dos projetos. Além disso, este problema busca maximizar a quantidade de relações sociais positivas entre os membros alocados a um mesmo projeto, uma vez que a produtividade de um time está diretamente relacionada com essas relações positivas e, portanto, com o sucesso do projeto (BALLESTEROS-PÉREZ *et al.*, 2012). Note que neste problema, cada indivíduo pode possuir apenas uma habilidade, ou seja, o PFMT não trata questões de pessoas com múltiplas habilidades.

Para resolução deste problema, Gutiérrez *et al.* (2016) utilizaram três algoritmos: uma abordagem de Programação por Restrição, uma heurística de Busca Local e a meta-heurística *Variable Neighborhood Search (VNS)*, onde, dentre as três, o VNS obteve melhores resultados. Em (FIGUEIREDO; CAMPÊLO, 2018), também foi apresentado um algoritmo para resolução do PFMT, a meta-heurística *Simulated Annealing (SA)*. No Capítulo 5, o VNS de Gutiérrez *et al.* (2016) e o SA de Figueiredo e Campêlo (2018) serão comparados com o algoritmo proposto neste trabalho. Por esta razão, os detalhes das duas implementações são apresentados nas subseções seguintes.

3.2.1 *Variable Neighborhood Search*

Para entendimento do algoritmo VNS de Gutiérrez *et al.* (2016), faz-se necessário conceituar a busca de vizinhança utilizada neste algoritmo, denominada *vizinhança k -habilidade* e denotada por N^k . Esta busca consiste em, dada uma solução viável X , obtida através da aplicação de programação por restrição considerando o modelo quadrático apresentado por Gutiérrez *et al.* (2016), seleciona-se k habilidades aleatórias ($1 \leq k \leq f$), executando novamente o modelo matemático, onde, neste caso, não serão permitidos que pessoas que estejam exercendo alguma das k habilidades sorteadas possam ser alocadas nesta nova solução. Desta forma, são alocadas novas pessoas a estas k habilidades e, portanto, gerada uma nova solução viável X' , vizinha de X .

Além da busca de vizinhança, o algoritmo VNS também utiliza o algoritmo Local Search (LS), apresentado no Algoritmo 8. Este algoritmo recebe como entrada um número máximo de iterações M , um conjunto Q de listas de pessoas $\{Q_1, Q_2, \dots, Q_f\}$ em H que compartilham a mesma habilidade $k \in K$ e o restante dos parâmetros são iguais aos definidos neste trabalho para o $PFMT_{MH}$. A saída deste algoritmo é uma solução ótima local X e sua eficiência $E(X)$. A

eficiência E de uma solução é definida na Subseção 4.4.1.2. A partir de uma solução inicial X_0 , será utilizada a vizinhança 1 -habilidade N^1 para encontrar uma solução vizinha X' . Se a solução X' for melhor que a solução atual ($E(X') > E(X)$), ela é tomada como a nova solução corrente.

Algoritmo 8: $LS(N^1, X)$

Entrada: M, Q, D, R, S, W

Saída: Um ótimo local X e sua eficiência $E(X)$.

```

1 início
2    $X_0 \leftarrow$  solução inicial;
3    $E(X_0) \leftarrow$  eficiência de  $X_0$ ;
4    $i \leftarrow 0, X \leftarrow X_0$ ;
5   enquanto  $i < M$  faça
6      $X' \leftarrow N^1(X)$ ;
7     se  $E(X') > E(X)$  então
8        $X \leftarrow X', E(X) \leftarrow E(X')$ ;
9     fim
10     $i = i + 1$ ;
11  fim
12 fim
```

Por fim, a meta-heurística VNS é ilustrada no Algoritmo 9. Inicialmente, o VNS realiza uma busca local usando a vizinhança N^1 até alcançar uma solução ótima local X . Deste ponto em diante, o algoritmo explora sequencialmente outras vizinhanças k maiores, eventualmente voltando a vizinhança N^1 quando a vizinhança N^k com $k > 1$ habilidades encontra uma solução X' melhor que X ($E(X') > E(X)$).

Algoritmo 9: Variabel Neighborhood Search k-habilidade

Entrada: M', Q, D, R, S, W
Saída: Um ótimo local X e sua eficiência $E(X)$.

```

1 início
2    $X_0 \leftarrow$  solução inicial;
3    $E(X_0) \leftarrow$  eficiência de  $X_0$ ;
4    $i \leftarrow 0, X \leftarrow X_0$ ;
5   enquanto  $i < M'$  faça
6      $k \leftarrow 1, j \leftarrow 1$ ;
7     enquanto  $j \leq f$  faça
8        $X' \leftarrow LS(N^k, X)$ ;
9       se  $E(X') > E(X)$  então
10         $X \leftarrow X', E(X) \leftarrow E(X'), k \leftarrow 1$ ;
11      fim
12      senão
13         $k \leftarrow k + 1$ ;
14      fim
15       $j \leftarrow j + 1$ ;
16    fim
17     $i \leftarrow i + 1$ ;
18  fim
19 fim

```

3.2.2 Simulated Annealing

O Algoritmo 10 apresenta a meta-heurística SA de Figueiredo e Campêlo (2018) para resolução do PFMT. Primeiramente, é criada uma solução inicial X a partir de um algoritmo de fluxo máximo, tal qual descrito na Seção 4.2. A partir da solução X , é chamado o método $buscaVizinhanca(X, k)$. Este método seleciona, aleatoriamente, um subconjunto K' de k habilidades. Então, para cada $k_a \in K'$, se a pessoa h_i exerce a habilidade k_a no projeto p_l , o arco (v_i, v_{la}) pode ser desconsiderado da rede de fluxo com probabilidade p , escolhida aleatoriamente no intervalo $[0.5, 1]$. Se isto resultar em uma solução X' inviável, ela é descartada e o método $buscaVizinhanca(X, k)$ irá retornar X em vez de X' .

Em (FIGUEIREDO; CAMPÊLO, 2018), a temperatura é inicializada em 100. A cada iteração, ela decreta em 99% de seu valor atual. Quando a temperatura atinge um valor

Algoritmo 10: Simulated Annealing

Saída: Uma solução X^* .

```

1 início
2    $X \leftarrow$  solução inicial,  $X^* \leftarrow X$ ,  $temp \leftarrow 100$ ,  $i \leftarrow 0$ ;
3   enquanto  $i < 1000$  faça
4      $temp \leftarrow temp \times 0,99$ ;
5      $k \leftarrow 1$ ;
6     se  $temp < 0.01$  então
7        $temp \leftarrow 100$ ;
8     fim
9     enquanto  $k < f$  faça
10       $X' \leftarrow buscaVizinhanca(X, k)$ ,  $\Delta \leftarrow E(X') - E(X)$ ;
11      se  $\Delta < 0 \parallel numeroAleatorio() < e^{-\Delta/temp}$  então
12         $X \leftarrow X'$ ;
13        se  $\Delta < 0$  então
14           $X^* \leftarrow X$ ,  $k \leftarrow 1$ ;
15        fim
16      fim
17       $k \leftarrow k + 1$ ;
18    fim
19     $i \leftarrow i + 1$ ;
20  fim
21  retorna  $X^*$ ;
22 fim

```

menor que 0.01, o sistema é reaquecido para 100 unidades. Com este processo de reaquecimento, o algoritmo ganha novas oportunidades para buscar por outras regiões no espaço de busca. O critério de parada deste algoritmo é um número máximo de iterações, que, neste caso, é 1000.

3.3 Problema de Formação de Times Sociotécnicos

O Problema de Formação de Times Sociotécnicos (PFTS) proposto por Campêlo *et al.* (2018) também generaliza o PFT (LAPPAS *et al.*, 2009). Como no PFMT (GUTIÉRREZ *et al.*, 2016), neste problema são considerados múltiplos projetos, porém considera-se que as pessoas possuem não apenas uma habilidade, mas um conjunto delas. Outra diferença é que o

PFMT possui frações de dedicação de pessoas, enquanto que no PFTS uma pessoa só pode estar em um time, com 100% de seu tempo dedicado a ele.

Sendo assim, dado um conjunto de pessoas, com diferentes conjuntos de habilidades, e uma rede social que captura a afinidade entre elas, o PFTS consiste em encontrar um conjunto de times disjuntos, respeitando as restrições de demanda de habilidades e maximizando a afinidade entre os membros de um mesmo time, uma vez que as relações sociais positivas em um time impactam diretamente no sucesso do projeto (BALLESTEROS-PÉREZ *et al.*, 2012). Embora o PFTS trate a questão de pessoas com múltiplas habilidades, o fracionamento de dedicação de tempo não é abordado.

4 O PROBLEMA DE FORMAÇÃO DE MÚLTIPLOS TIMES COM MÚLTIPLAS HABILIDADES

Neste capítulo, é apresentado o Problema de Formação de Múltiplos Times com Múltiplas Habilidades ($PFMT_{MH}$), foco de estudo deste trabalho. A definição formal do problema é apresentada na Seção 4.1. Na Seção 4.2, é apresentada a maneira como é gerada uma solução viável para este problema e, na Seção 4.3, é ilustrada como uma solução é representada neste trabalho. Por fim, na Seção 4.4, é apresentado o algoritmo proposto para resolução do $PFMT_{MH}$.

4.1 Definição do problema

O $PFMT_{MH}$ é uma união do PFMT (GUTIÉRREZ *et al.*, 2016) e do PFTS (CAM-PÊLO *et al.*, 2018), contando, então, com as seguintes características: múltiplos times, pessoas com múltiplas habilidades e frações de dedicação de pessoas. De forma semelhante ao PFTS, neste problema as pessoas possuem um conjunto de habilidades. No entanto, também considera-se que as mesmas podem ter frações de dedicação alocadas a vários times simultaneamente, desde que não exceda 100% de seu tempo de trabalho, como no PFMT. Dito isso, o objetivo do $PFMT_{MH}$ é formar times que respeitem as restrições de demanda de habilidades dos projetos, além de maximizar a afinidade entre os seus membros, uma vez que a saúde das relações sociais em um time está diretamente ligada à produtividade do mesmo e, portanto, ao sucesso do projeto (BALLESTEROS-PÉREZ *et al.*, 2012). Mais formalmente, temos:

Problema 4.1.1 *O Problema de Formação de Múltiplos Times com Múltiplas Habilidades (PFMT_{MH})*

Entrada: uma tupla (P, H, K, D, q, R, S, W) , onde:

- *P é um conjunto de projetos aos quais as pessoas são alocados, onde $|P| = m$;*
- *H é um conjunto de pessoas disponíveis que podem ser alocadas a P , onde $|H| = n$;*
- *K é um conjunto de habilidades que as pessoas em H possuem, onde $|K| = f$. Todas as habilidades em K são requeridas por pelo menos um projeto em P ;*
- *D é um conjunto de particionamentos de tempos possíveis para uma pessoa em H , onde $|D| = t$. Por exemplo, $\{0, 1\}$ (alocação de período completo) e $\{0; 0, 5; 1\}$ (alocação de meio período). Denominamos α a diferença entre os elementos em D ;*
- *Função $q : H \rightarrow 2^K \setminus \emptyset$ que atribui a cada pessoa $h_i \in H$ um subconjunto não-vazio $q(h_i) \subseteq K$ de habilidades;*
- *$R_{m \times f}$ é a matriz de demandas do projeto que especifica quantas pessoas (ou frações de pessoas, se permitido) com a mesma habilidade $k_a \in K$ são necessárias para cada projeto particular $p_l \in P$, onde cada elemento r_{la} é sempre um número real não-negativo;*
- *$S_{n \times n}$ é uma matriz sociométrica, com elementos denotados como s_{ij} , que contém a predisposição de cada pessoa $h_i \in H$ para trabalhar com a pessoa $h_j \in H$. Esses elementos da matriz podem assumir três possíveis valores, $s_{ij} = \{-1, 0, +1\}$, onde o valor $+1$ é escolhido se h_i está disposto a trabalhar com h_j , o valor -1 se h_i prefere não trabalhar com h_j , e o valor 0 se h_i é neutro em relação a trabalhar com h_j . Além disso, os elementos da diagonal s_{ii} são sempre iguais a $+1$;*
- *W é uma lista de m números que descreve a prioridade dos projetos. Por razões de manter a função objetivo no intervalo $[0, 1]$, a soma de todos os elementos é igual a 1 .*

Questão: *Determinar $|P|$ times, onde:*

1. *cada pessoa $h_i \in H$ possa ser alocada a vários times em P , com no mínimo uma habilidade $f(h_i) \in q(h_i)$ em cada time alocado, onde sua alocação de tempo total não exceda 100%;*
2. *cada time p_l tenha especificamente r_{la} frações de dedicação de pessoas, para cada habilidade $k_a \in K$;*
3. *a harmonia dos times seja maximizada.*

Ou a verificação de que esses times não podem ser formados.

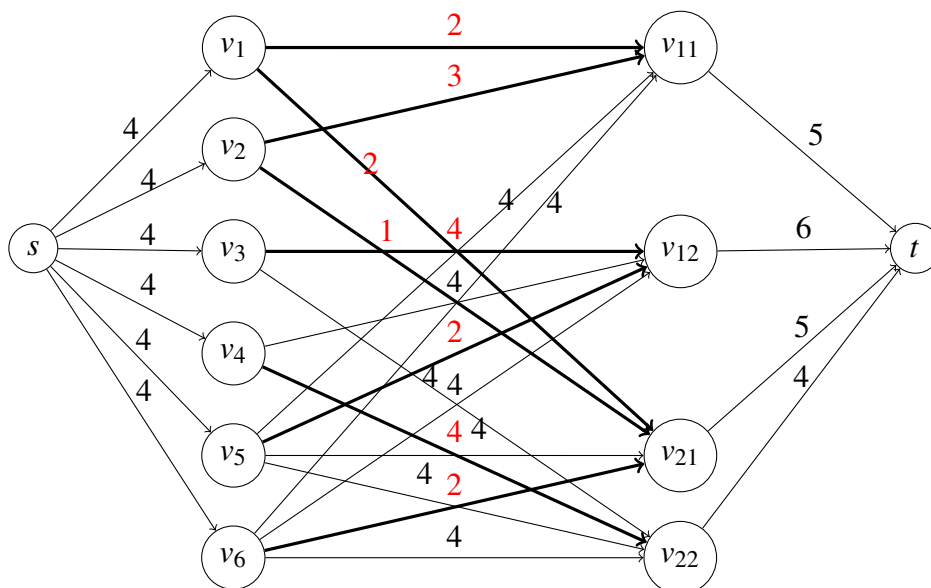
4.2 Geração de Solução Viável

Para a geração de uma solução viável, foi realizada uma relaxação do $PFMT_{MH}$, onde foram desconsideradas as restrições de afinidade entre as pessoas. Logo em seguida, foi realizada uma redução deste problema relaxado para o problema de fluxo máximo pois, como este é um problema polinomial (CORMEN *et al.*, 2012), esta é uma das formas de gerar uma solução inicial viável em tempo polinomial. Esta redução é apresentada a seguir. À medida que a redução é descrita, utiliza-se a seguinte instância I como ilustração:

- $P = \{p_1, p_2\}$;
- $H = \{h_1, h_2, h_3, h_4, h_5, h_6\}$;
- $K = \{k_1, k_2\}$;
- $D = \{0; 0,25; 0,5; 0,75; 1\}$ e $\alpha = 0,25$;
- $q(h_1) = \{k_1\}$, $q(h_2) = \{k_1\}$, $q(h_3) = \{k_2\}$, $q(h_4) = \{k_2\}$, $q(h_5) = \{k_1, k_2\}$ e $q(h_6) = \{k_1, k_2\}$;
- $r_{11} = 1,25$, $r_{12} = 1,5$, $r_{21} = 1,25$ e $r_{22} = 1,0$.

A Figura 1 apresenta a rede $G(I)$ gerada a partir da instância I acima. Uma solução viável é descrita pelos arcos mais escuros, onde o valor em vermelho representa o fluxo que passa pelo arco.

Figura 1 – Rede $G(I)$ gerada a partir de uma instância I do $PFMT_{MH}$. Solução destacada pelos arcos mais escuros, onde o valor em vermelho representa o fluxo que passa pelo arco.



Fonte: Elaborado pela autora (2018).

Em geral, dada uma instância I , constrói-se a rede $G(I)$ da seguinte maneira. Inici-

almente, são criados os vértices fonte e sumidouro, denotados por s e t , respectivamente. Para cada pessoa $h_i \in H$ é criado um vértice v_i e incluído um arco (s, v_i) com $c_{sv_i} = 1/\alpha$; a instância I utilizada no exemplo possui 6 pessoas e, então, são criados os vértices v_1, v_2, v_3, v_4, v_5 e v_6 e, como $\alpha = 0,25$, $c_{sv_i} = 4$. Para cada projeto $p_l \in P$ e cada habilidade $k_a \in K$, cria-se um vértice, denotado por v_{la} , se, e somente se, $r_{la} \neq 0$; na instância I utilizada como exemplo, o projeto 1 necessita de 1,25 e 1,5 de frações de tempos de pessoas com as habilidades k_1 e k_2 , respectivamente, ($r_{11} = 1,25$ e $r_{12} = 1,5$), e o projeto 2 requer 1,25 e 1 de frações de dedicação de tempo das pessoas com as habilidades k_1 e k_2 , respectivamente ($r_{21} = 1,25$ e $r_{22} = 1$). Por isso, são criados os vértices v_{11}, v_{12}, v_{21} e v_{22} . São incluídos arcos (v_i, v_{la}) com $c_{v_i v_{la}} = 1/\alpha$ se, e somente se, $k_a \in q(h_i)$, ou seja, se a pessoa h_i possui a habilidade k_a ; na instância I utilizada como exemplo, as pessoas h_1 e h_2 possuem a habilidade k_1 , as pessoas h_3 e h_4 possuem apenas a habilidade k_2 e as pessoas h_5 e h_6 possuem ambas as habilidades k_1 e k_2 . Finalmente, para cada vértice v_{la} , é incluído um arco (v_{la}, t) com $c_{v_{la}t} = r_{la}/\alpha$.

Depois de construída a rede de fluxo, foi utilizado o algoritmo 6 (*relabel-to-front*), descrito na Seção 2.2, para gerar uma solução inicial. Perceba que, se possível, o fluxo máximo irá saturar todos os arcos (v_{la}, t) , significando que cada demanda da habilidade $k_a \in K$ no projeto $p_l \in P$ foi atendida e, assim, temos uma solução viável. Caso contrário, a solução é dita inviável e é descartada. Dessa forma, mesmo que as afinidades entre as pessoas não estejam sendo consideradas, a solução gerada é viável. A meta-heurística tentará melhorar essa solução considerando as relações sociais entre as pessoas.

4.3 Representação da Solução

Neste trabalho, uma solução é representada da seguinte forma. Os projetos possuem subgrupos de habilidades e estas, por sua vez, possuem subgrupos de frações de tempo. Então, dentro de cada subgrupo de fração de tempo, há um vetor de pessoas.

A Figura 2 ilustra a representação de uma solução viável, dada a instância I apresentada na Subseção 4.2. A pessoa h_1 está exercendo a habilidade k_1 no projeto p_1 com uma dedicação de tempo de 0,5, ou seja, 50% de seu tempo de trabalho, assim como a pessoa h_2 , mas esta está dedicando 0,75 de seu tempo. Estas pessoas também estão alocadas ao projeto p_2 com a habilidade k_1 , onde h_1 e h_2 estão dedicando, respectivamente, 0,5 e 0,25 de seus tempos de trabalho. A pessoa h_3 está exercendo, com uma fração de tempo 1,0, a habilidade k_2 no projeto p_1 . Já a pessoa h_4 está dedicando 100% de seu tempo de trabalho ao projeto p_2 , com a habilidade

k_2 . A pessoa h_5 está no projeto p_1 , dedicando 0,5 de seu tempo com a habilidade k_2 e a pessoa h_6 está exercendo a habilidade k_1 em p_2 , também com 0,5 de dedicação de tempo.

Figura 2 – Representação de uma solução viável X , de acordo com a instância I apresentada na Seção 4.2.

p_1								p_2							
k_1				k_2				k_1				k_2			
0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0
	h1														
		h2			h5			h2					h1		
									h6						
															h4

Fonte: Elaborado pela autora (2018).

4.4 Limite Superior

Nesta seção, é descrito um algoritmo meta-heurístico que fornece um bom limite superior para o $PFMT_{MH}$. Este algoritmo baseia-se na meta-heurística Algoritmos Genéticos, onde, a partir de um conjunto inicial de soluções viáveis (população), a cada iteração obtém-se novas soluções que podem ser selecionadas para a próxima população, respeitando sempre as características de viabilidade das soluções futuras.

4.4.1 Heurística baseada em Algoritmos Genéticos

O algoritmo proposto neste trabalho é baseado na meta-heurística Algoritmos Genéticos, apresentada na Subseção 2.4.1.1. Uma descrição geral pode ser visualizada no Algoritmo 11. As demais subseções apresentarão detalhes do seu funcionamento.

4.4.1.1 Geração da População Inicial

Primeiramente, o método $geracaoPopulacaoInicial(\beta)$ gera um conjunto de soluções iniciais, denominado população inicial, de tamanho β . O algoritmo para geração da população inicial foi criado a partir da redução apresentada na Seção 4.2, mais detalhes serão descritos a seguir.

Dada uma rede de fluxo G para o $PFMT_{MH}$, é sorteado um arco (v_i, v_{la}) aleatório de G . Em seguida, a capacidade desse arco é diminuída em $d \in D$ unidades, onde d é sorteado

Algoritmo 11: Algoritmo Genético

Entrada: $(P, H, K, D, q, R, S, W, \beta, probOperador)$
Saída: Melhor solução encontrada para o $PFMT_{MH}$.

```

1 início
2    $populacao \leftarrow geracaoPopulacaoInicial(\beta);$ 
3    $probIndividuos \leftarrow calculoAptidaoPopulacao(populacao);$ 
4   enquanto critério de parada não for atendido faça
5      $operador \leftarrow selecaoOperadorGenetico(probOperador);$ 
6      $individuos \leftarrow selecaoIndividuosReproducao(probIndividuos);$ 
7      $populacao \leftarrow populacao \cup reproducao(operador, individuos);$ 
8      $probIndividuos \leftarrow calculoAptidaoPopulacao(populacao);$ 
9      $populacao \leftarrow selecaoNovaPopulacao(probIndividuos, \beta);$ 
10  fim
11  retorna melhor solução encontrada.
12 fim

```

aleatoriamente, e o algoritmo de fluxo máximo é chamado novamente. Como a capacidade do arco (v_i, v_{la}) foi diminuída, obrigatoriamente a demanda r_{la} precisará ser atendida por outra(s) pessoa(s). Sendo assim, é gerada uma solução diferente da anterior. Este procedimento se repete até que a população tenha sido inteiramente construída, ou seja, até que sejam geradas β soluções. Neste trabalho, definimos $\beta = 50$.

4.4.1.2 Aptidão de uma Solução

A aptidão (eficiência) de uma solução é medida tal qual em (GUTIÉRREZ *et al.*, 2016). A *Eficiência do Projeto* é definida como a soma das relações sociais de todos os pares de pessoas trabalhando no mesmo projeto. Dessa forma, a eficiência de um projeto $l(e_l)$ é dada por:

$$e_l = \frac{1}{2} \left(1 + \frac{\sum_{h_i, h_j \in H} S_{ij} x_{il} x_{jl}}{(\sum_{k_a \in K} r_{al})^2} \right), \quad (4.1)$$

onde os termos $\frac{1}{2}$ e 1 alteram o intervalo de variação de $[-1, 1]$ para $[0, 1]$.

$X = [x_{il}]$ é o conjunto das variáveis de decisão que modelam a matriz de alocação $(n \times m)$, onde cada elemento $x_{il} \in D$ especifica a fração de tempo em que cada pessoa $h_i \in H$ está alocada a cada projeto $p_l \in P$.

Na equação (4.1), o numerador da fração representa o total de relações sociais entre cada par de pessoas h_i e h_j , ponderado pelas frações de tempo ocupadas por elas. Note que,

quando todos os valores $s_{ij} = +1$, $\forall h_i, h_j \in H$, o numerador coincide com o denominador, resultando em 1, sendo esta a eficiência máxima possível para um projeto.

Sendo assim, a *Eficiência Global* de uma solução X , $E(X)$, é dada pelo somatório das eficiências de todos os projetos, ponderadas pelos valores $w_l \in W$, ou seja:

$$E(X) = \sum_{p_l \in P} w_l e_l. \quad (4.2)$$

Desta forma, o método *calculaAptidaoPopulacao(populacao)* recebe como parâmetro uma população de indivíduos, isto é, um conjunto de soluções e calcula a aptidão de cada uma, de acordo com a equação (4.2). Em seguida, é criado um vetor de probabilidades, denominado *vetorProbIndividuos*, que contém a probabilidade de cada indivíduo ser selecionado para reprodução. Para cada indivíduo X , esta probabilidade é dada por:

$$probIndividuo(X) = \frac{E(X)}{\sum_{X' \in populacao} E(X')}. \quad (4.3)$$

Note que *probIndividuo(X)* é diretamente proporcional à aptidão de X , isto é, quanto maior a aptidão de um indivíduo, maior é a probabilidade deste ser selecionado para reprodução. Esta forma de seleção é conhecida como Método da Roleta, onde cada indivíduo da população é representado na roleta proporcionalmente à sua aptidão e, assim, esta é girada para selecionar os indivíduos. Finalmente, o vetor de probabilidades é definido como segue:

$$vetorProbIndividuos = [probIndividuo(X_1), probIndividuo(X_2), \dots, probIndividuo(X_\beta)]. \quad (4.4)$$

4.4.1.3 Critérios de Parada

Foram utilizados dois critérios de parada neste algoritmo: número máximo de gerações (γ) e convergência, isto é, quando não há melhora na solução durante um número θ de gerações. No algoritmo proposto, utilizou-se $\gamma = 1000$ e $\theta = 200$.

4.4.1.4 Operadores Genéticos

Após a obtenção de um conjunto de soluções iniciais viáveis, foram implementados quatro tipos de operadores genéticos para obtenção de novas soluções, sendo dois deles baseados no operador *swap*, definido no trabalho de Eiben e Ruttkay (1996). Este operador escolhe dois genes aleatoriamente em um cromossomo e permuta suas posições. Além das duas variações do operador *swap*, também foi implementada uma mutação e um *crossover*, onde dois cromossomos

são escolhidos para reprodução. Para ilustração destes operadores genéticos, é utilizada a instância I definida na Seção 4.2 e a solução X apresentada na Figura 2.

4.4.1.4.1 *Swap* 1

Neste *swap*, são selecionadas duas pessoas que estão em projetos diferentes, exercendo a mesma habilidade na mesma fração de tempo e, então, essas pessoas são trocadas de projetos. Para isso, primeiramente é criado um vetor, denominado *vetorProbHabilidades*, de tamanho f , que contém a probabilidade de cada habilidade $k_a \in K$ ser sorteada. Desta forma, para cada habilidade k_a , esta probabilidade é dada por:

$$probHabilidade(k_a) = \begin{cases} \frac{\sum_{p \in P} r_{pk_a}}{\sum_{p \in P, k \in K} r_{pk}}, & \text{se } k_a \text{ é demandada em pelo menos dois projetos} \\ 0, & \text{caso contrário.} \end{cases} \quad (4.5)$$

Note que $probHabilidade(k_a)$ é diretamente proporcional à demanda de k_a nos projetos, ou seja, quanto maior a demanda de uma habilidade k_a , maior será a probabilidade dela ser selecionada. Mais ainda, se uma habilidade k_a for demandada em apenas um projeto, sua probabilidade será 0 em *vetorProbHabilidades*, visto que, para este *swap*, precisa-se de duas pessoas em projetos diferentes exercendo a mesma habilidade. Desta maneira, de acordo com o vetor de probabilidades *vetorProbHabilidades*, uma habilidade k_a é selecionada; suponha que, na instância I , a habilidade k_2 foi selecionada.

Logo em seguida, é criado um vetor, denominado *vetorProbProjetos*, de tamanho m , que contém a probabilidade de cada projeto $p_l \in P$ ser sorteado. Então, dada a habilidade k_a selecionada, a probabilidade de um projeto p_l ser sorteado é definida por:

$$probProjeto(p_l) = \frac{r_{p_l k_a}}{\sum_{p \in P} r_{pk_a}}. \quad (4.6)$$

Perceba que $probProjeto(p_l)$ baseia-se nas demandas de cada projeto p_l pela habilidade k_a selecionada, ou seja, quanto maior a demanda da habilidade k_a em um projeto p_l , maior será a probabilidade dele ser selecionado. Sendo assim, dois projetos p_i e p_j são selecionados; na instância I , são selecionados p_1 e p_2 .

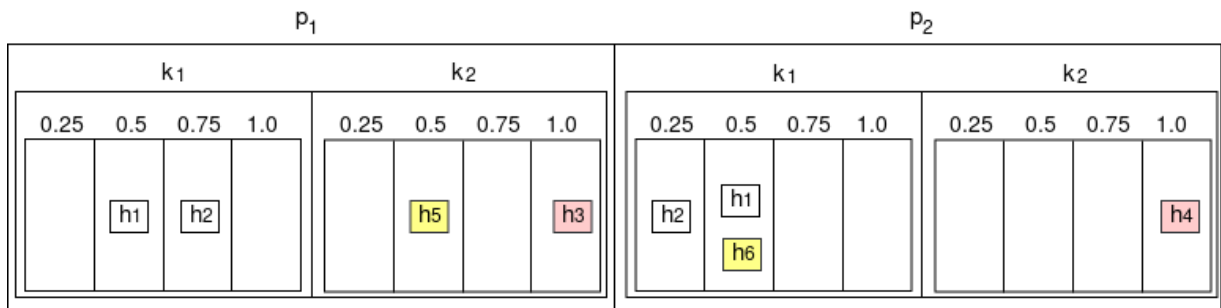
Após a seleção da habilidade k_a e dos projetos p_i e p_j , deve-se selecionar uma fração de tempo $d_i \in D$. Para isto, é criado um vetor, denominado *vetorProbTempos*, de tamanho t , que contém a probabilidade de cada fração de tempo $d_i \in D$ ser sorteada. Então, dada uma solução

X , dois projetos p_i e p_j e uma habilidade k_a , a probabilidade de uma fração d_i ser selecionada é dada por:

$$probTempo(d_i) = \begin{cases} \frac{|X[p_i][k_a][d_i]| + |X[p_j][k_a][d_i]|}{\sum_{d \in D} |X[p_i][k_a][d]| + |X[p_j][k_a][d]|}, & \text{se } |X[p_i][k_a][d_i]| \neq 0 \text{ e } |X[p_j][k_a][d_i]| \neq 0 \\ 0, & \text{caso contrário.} \end{cases} \quad (4.7)$$

Note que $probTempo(d_i)$ está relacionada com a quantidade de pessoas exercendo a habilidade k_a em ambos os projetos p_i e p_j na fração de tempo d_i . Se, em pelo menos um dos projetos selecionados, não houver pessoas trabalhando na fração de tempo d_i , então a probabilidade de d_i será 0 em $vetorProbTempos$, uma vez que precisa-se, para este $swap$, de duas pessoas em projetos diferentes com a mesma fração de tempo. De acordo com o vetor de probabilidades $vetorProbTempos$, uma fração de tempo d_i é selecionada; na instância I , é selecionada a fração de tempo 1,0.

Figura 3 – Representação dos $swaps$ 1 e 2, de acordo com a instância I definida na Seção 4.2 e a solução X apresentada na Figura 2. As pessoas marcadas com as cores vermelha e amarela foram as escolhidas para os $swaps$ 1 e 2, respectivamente.



Fonte: Elaborado pela autora (2018).

A seguir, são selecionadas, aleatoriamente, duas pessoas $h_i, h_j \in H$ na solução que estão, respectivamente, nos projetos p_i e p_j , exercendo a mesma habilidade k_a na mesma fração de dedicação de tempo d_i . Logo em seguida, estas pessoas são trocadas de projetos, isto é, a pessoa h_i é alocada ao projeto p_j e a pessoa h_j é alocada ao projeto p_i . Já que ambos os projetos continuam com a habilidade k_a sendo exercida na fração de tempo d_i , a demanda de k_a nestes projetos é mantida e, como cada pessoa ainda irá trabalhar com a mesma fração de tempo d_i , a restrição de dedicação de tempo não será violada. Logo, obtém-se uma nova solução viável. Na instância I , foram selecionadas as pessoas h_3 e h_4 , marcadas em vermelho na Figura 3, e a nova

solução possuiria h_3 no projeto p_2 e h_4 no projeto p_1 , ambas dedicando 100% de tempo com a habilidade k_2 .

É importante destacar que, após uma troca, uma pessoa pode ter sido alocada à uma equipe em que já estava trabalhando com aquela habilidade. Nos casos em que isso acontece, pode-se optar por somar as frações de tempo desta pessoa, com uma probabilidade de 50%.

4.4.1.4.2 Swap 2

No *swap 2*, são selecionadas duas pessoas que estão em projetos diferentes, exercendo habilidades diferentes, sendo que cada uma possui a habilidade que está sendo exercida pela outra, na mesma fração de tempo. Assim, primeiramente, cria-se um vetor, denominado *vetorProbHabilidades2*, de tamanho f , contendo a probabilidade de cada habilidade $k_a \in K$ ser sorteada. No entanto, a probabilidade de uma habilidade k_a ($probHabilidade2(k_a)$) ser selecionada é definida de maneira diferente neste *swap*. Esta probabilidade é definida da seguinte forma:

$$probHabilidade2(k_a) = \frac{\sum_{p \in P} r_{pk_a}}{\sum_{p \in P, k \in K} r_{pk}}. \quad (4.8)$$

Note que a única diferença entre $probHabilidade(k_a)$, definido na equação 4.5, e $probHabilidade2(k_a)$ é que, se a habilidade k_a for demandada em apenas um projeto, a probabilidade desta ser selecionada no *swap 1* é 0 ($probHabilidade(k_a) = 0$), mas não no *swap 2*, pois neste *swap* não é necessário que as habilidades escolhidas sejam demandadas em mais de um projeto. Depois de criado este vetor de probabilidades, são selecionadas duas habilidades $k_a, k_b \in K$; no exemplo ilustrado pela instância I , suponha que são selecionadas as habilidades k_1 e k_2 .

Logo após, é criado um vetor, denominado *vetorProbProjetos2*, de tamanho m , contendo a probabilidade de cada projeto $p_l \in P$ ser sorteado. Portanto, dadas duas habilidades k_a e k_b selecionadas, a probabilidade de um projeto p_l ser sorteado é dada por:

$$probProjeto2(p_l) = \frac{r_{p_l k_a} + r_{p_l k_b}}{\sum_{p \in P} (r_{pk_a} + r_{pk_b})}. \quad (4.9)$$

Note que $probProjeto2(p_l)$ é baseada nas demandas das habilidades k_a e k_b no projeto p_l . Sendo assim, quanto maior for a demanda das duas habilidades selecionadas em um projeto, maior será a probabilidade dele ser sorteado. Assim, são selecionados dois projetos $p_i, p_j \in P$; no exemplo ilustrado pela instância I , suponha que foram selecionados os projetos p_1 e p_2 .

Depois de selecionados os dois projetos e as duas habilidades, é selecionada uma fração de dedicação de tempo $d_i \in D$. Para isto, cria-se um vetor, denominado *vetorProbTempos2*, de tamanho t , contendo a probabilidade de cada fração de tempo d_i ser sorteada. Então, dada uma solução X , dois projetos p_i e p_j e duas habilidades k_a e k_b , a probabilidade de uma fração d_i ser selecionada é dada por:

$$probTempo2(d_i) = \frac{|X[p_i][k_a][d_i]| + |X[p_i][k_b][d_i]| + |X[p_j][k_a][d_i]| + |X[p_j][k_b][d_i]|}{\sum_{d \in D} |X[p_i][k_a][d]| + |X[p_i][k_b][d]| + |X[p_j][k_a][d]| + |X[p_j][k_b][d]|} \quad (4.10)$$

Note que $probTempo2(d_i)$ é baseada na quantidade de pessoas que estão nos projetos p_i e p_j , exercendo as habilidades k_a ou k_b , na fração de tempo d_i . Sendo assim, quanto maior a quantidade de pessoas que estão trabalhando em uma fração de tempo, maior será a probabilidade dela ser selecionada. Assim, uma fração de tempo d_i é selecionada; no exemplo ilustrado pela instância I , assumamos que a fração de tempo selecionada foi 0,5.

Portanto, são selecionadas duas pessoas $h_i, h_j \in H$ que estão nos projetos p_i e p_j , respectivamente, onde h_i esteja exercendo, na fração de tempo d_i , a habilidade k_a e possua k_b e h_j esteja exercendo k_b , também na fração de tempo d_i , e possua k_a . Assim, troca-se as duas pessoas na solução, resultando na pessoa h_i alocada ao projeto p_j , exercendo a habilidade k_b e a pessoa h_j alocada ao projeto p_i , exercendo a habilidade k_a , ambas na mesma fração de tempo d_i . No exemplo ilustrado pela instância I , suponhamos que foram escolhidas as pessoas h_5 e h_6 , marcadas em amarelo na Figura 3, e a nova solução possuiria h_5 no projeto p_2 exercendo a habilidade k_1 e h_6 no projeto p_1 exercendo a habilidade k_2 , ambas na fração de tempo 0,5.

Note que é possível realizar esta troca, uma vez que as duas pessoas possuem as duas habilidades k_a e k_b . Já que a fração de tempo d_i é mantida para ambas as pessoas e em ambos os projetos, as restrições de dedicação de tempo e de demanda de habilidades não são violadas. Sendo assim, a nova solução gerada por este *swap* é viável.

4.4.1.4.3 Crossover

Dadas duas soluções, o *crossover* é realizado da seguinte maneira: escolhe-se um ponto de troca aleatório entre os projetos e são geradas (até) duas novas soluções. Então, verifica-se se há alguma pessoa inviável nesta nova solução, ou seja, se está com mais de 100% de tempo alocado. Para cada pessoa inviável, é realizada uma troca entre essa pessoa e uma outra pessoa disponível, que tenha tempo livre e a mesma habilidade, ou seja, é realizada uma mutação,

descrita na Subseção 4.4.1.4.4, para cada pessoa inviável. Neste trabalho, não foi utilizado uma taxa de *crossover*. Logo, se os indivíduos são selecionados para *crossover*, este será realizado.

Dadas duas soluções X e X' , é selecionado um ponto de *crossover* aleatório entre os projetos. A nova solução Y é composta pela primeira parte (do início até o ponto de *crossover*) de X e a segunda parte (do ponto de *crossover* + 1 até o final) de X' , e a nova solução Y' é composta pela primeira parte de X' e a segunda parte de X . Desta maneira, após gerar as duas novas soluções, verifica-se se há alguma pessoa inviável em cada solução e, se houver, é realizada uma mutação para tornar a solução viável. Em alguns casos, pode ocorrer de não ser possível realizar uma mutação, por não haver uma pessoa com a mesma habilidade da pessoa inviável e com tempo disponível. Sendo assim, a partir deste *crossover*, são geradas até duas novas soluções viáveis.

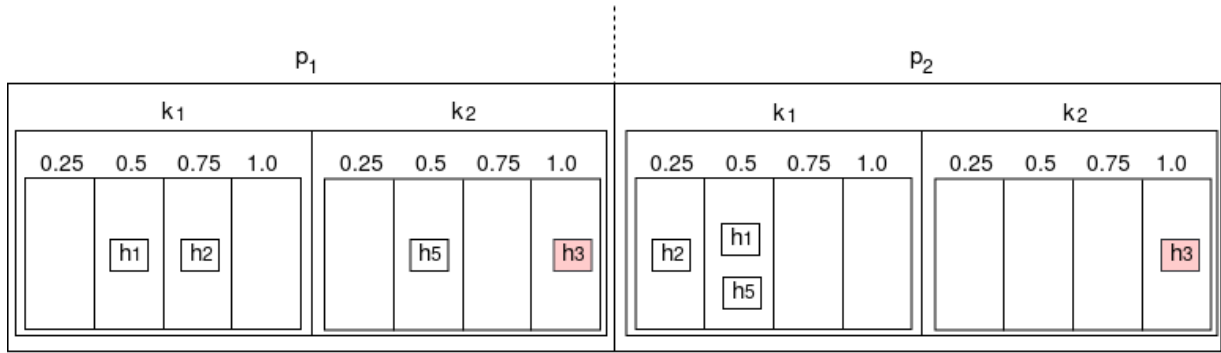
Como ilustração, considere as soluções X e X' , definidas nas figuras 2 e 4, respectivamente. Há apenas um único ponto de *crossover* possível, aquele entre os projetos p_1 e p_2 . Sendo assim, são geradas as duas novas soluções Y e Y' , representadas nas Figuras 5 e 6, respectivamente. As pessoas marcadas em vermelho são aquelas que ficaram inviáveis na nova solução, ou seja, estão excedendo 100% de tempo de trabalho. Na Subseção 4.4.1.4.4, será ilustrado como tornar viável uma destas soluções.

Figura 4 – Representação de uma solução viável X' , dada a instância I apresentada na Seção 4.2.

p_1								p_2							
k_1				k_2				k_1				k_2			
0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0
	h1	h2			h5		h4	h2	h1						h3

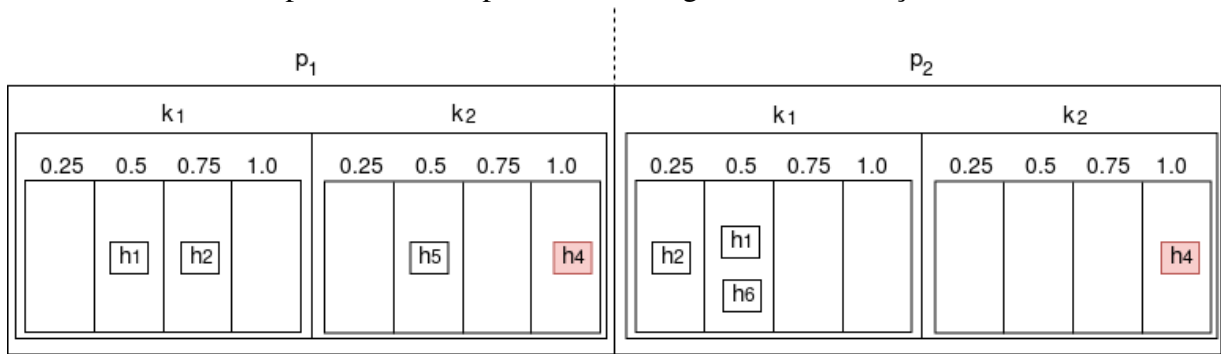
Fonte: Elaborado pela autora (2018).

Figura 5 – Representação da solução Y gerada através de um *crossover* entre as soluções X e X' , definidas nas figuras 2 e 4, respectivamente. A pessoa h_3 , marcada com a cor vermelha, está excedendo 100% de seu tempo de trabalho. Portanto, esta solução está inviável e, possivelmente, poderá ser corrigida com a mutação.



Fonte: Elaborado pela autora (2018).

Figura 6 – Representação da solução Y' gerada através de um *crossover* entre as soluções X e X' , definidas nas figuras 2 e 4, respectivamente. A pessoa h_4 , marcada com a cor vermelha, está excedendo 100% de seu tempo de trabalho. Portanto, esta solução está inviável e, possivelmente, poderá ser corrigida com a mutação.



Fonte: Elaborado pela autora (2018).

4.4.1.4.4 Mutação

Dada uma solução X para o $PFMT_{MH}$, o formato da operação de mutação é definida a seguir. É criado um vetor, de tamanho m , onde cada posição representa a probabilidade de um projeto $p_l \in P$ ser selecionado para mutação, onde esta probabilidade é dada por:

$$probMutacao(p_l) = \frac{1 - e_{p_l}}{\sum_{p \in P} (1 - e_p)}. \quad (4.11)$$

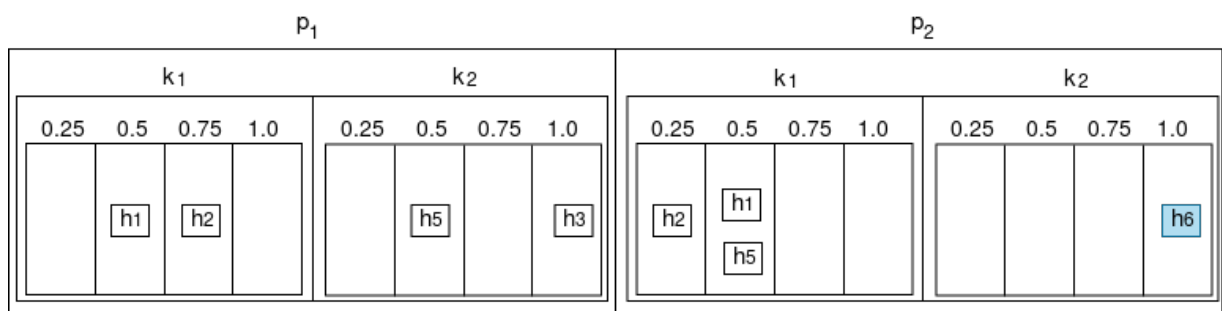
Note que esta probabilidade é inversamente proporcional à eficiência de cada projeto p_l (e_{p_l}), definida na Subseção 4.4.1.2, ou seja, quanto menor a eficiência de um projeto, maior a probabilidade dele ser selecionado para mutação. É importante ressaltar que, de início, escolhia-se um projeto aleatório para sofrer mutação. Depois da alteração para selecionar os projetos de

acordo com esta probabilidade, os GAPs (diferença entre o valor da solução heurística e o valor ótimo) reduziram significativamente.

Depois de selecionar um projeto p_l , são sorteados uma habilidade $k_a \in K$ e uma fração de tempo $d_i \in D$ aleatórios. Logo em seguida, é sorteada uma pessoa $h_i \in H$ aleatória, que esteja exercendo a habilidade k_a no projeto p_l na fração de tempo d_i e, então, é escolhida uma outra pessoa $h_j \in H$ que possua a habilidade k_a e tenha tempo livre, no mínimo, igual a d_i . Assim, a pessoa h_j está apta a substituir a pessoa h_i , sem inviabilizar a solução. Então, é gerada uma nova solução onde h_j está exercendo a habilidade k_a no projeto p_l e na fração de tempo d_i . Para a realização da mutação, são computadas, previamente, as frações de tempos livres de cada pessoa na solução.

Como foi mencionado na Subseção 4.4.1.4.3, se um *crossover* gerar uma solução inviável, a mutação será utilizada para corrigi-la e torná-la viável, se possível. Então, considere a solução Y , ilustrada na Figura 5. A pessoa h_3 , marcada em vermelho, está excedendo 100% de seu tempo de trabalho. Então, foi selecionada a pessoa h_6 , marcada em azul na Figura 7, para substituir uma ocorrência de h_3 na solução Y , uma vez que h_6 possui tempo disponível e a habilidade k_2 . Note que, neste caso, h_6 pode substituir h_3 em qualquer um dos projetos. Então, foi selecionado aleatoriamente o projeto p_2 e, agora, a solução Y é viável.

Figura 7 – Representação de uma mutação na solução Y , definida na Figura 5. A pessoa h_6 , marcada em azul, foi inserida no projeto p_2 , substituindo h_3 , que agora não está mais excedendo 100% de seu tempo. Assim, a solução Y torna-se viável.



Fonte: Elaborado pela autora (2018).

Além de ser utilizada para corrigir as soluções inviáveis que o *crossover* pode gerar, a mutação também pode ocorrer, com uma probabilidade p , depois de realizado algum dos outros três operadores genéticos. Neste trabalho, a probabilidade p é definida da seguinte maneira: (i) se não houver pessoas disponíveis, atribui-se 0 à p , uma vez que não é possível realizar uma mutação neste caso; (ii) se o número de pessoas disponíveis for maior que 10, atribui-se 20%

à p ; (iii) caso o número de pessoas disponíveis seja menor que 10, é atribuído 10% à p ; e (iv) para as instâncias em que $D = [0, 1]$, atribui-se mais 10% à p , uma vez que, de acordo com os experimentos realizados, uma maior ocorrência de mutações é melhor nestas instâncias.

4.4.1.5 Seleção de Operador Genético

À cada iteração do algoritmo, um dos três operadores genéticos (*swaps* 1 e 2, e *crossover*) é selecionado. O Algoritmo 12 apresenta o passo a passo realizado para atribuir a probabilidade de selecionar cada operador. Estas probabilidades são definidas de acordo com as características da instância I recebida como entrada do algoritmo. Para isto, é criado um vetor de tamanho 3, que possui a probabilidade de cada um destes três operadores ser selecionado, onde as posições 0, 1 e 2 referem-se às probabilidades dos *swaps* 1 e 2, e *crossover*, respectivamente. Primeiramente, é atribuído 1 à probabilidade do *swap* 1 ($prob[swap1]$) e 0 às demais. Em seguida, são chamadas as funções de cálculo do *swap* 2 e do *crossover*, apresentadas nos algoritmos 13 e 14, respectivamente.

O Algoritmo 13 define a probabilidade do *swap* 2 da seguinte maneira. Primeiramente, verifica-se se a instância I é particular do PFMT (GUTIÉRREZ *et al.*, 2016), onde as pessoas possuem apenas uma habilidade. Neste caso, o *swap* 2 não é aplicável, uma vez que, para o mesmo, precisa-se de pessoas com, no mínimo, duas habilidades. Caso I seja uma instância do $PFMT_{MH}$, onde as pessoas possuem múltiplas habilidades, este *swap* é aplicável. Então, para estas instâncias, verifica-se se as habilidades estão bem distribuídas entre as pessoas. Dada uma habilidade $k_a \in K$, a sua distribuição entre as pessoas é definida como segue:

$$dist(k_a) = \frac{qtdPessoas(k_a)}{\sum_{k_i \in K} qtdPessoas(k_i)}, \quad (4.12)$$

onde $qtdPessoas(k_a)$ é a quantidade de pessoas que possuem a habilidade k_a . Foi estabelecido que, se para alguma habilidade k_a , $dist(k_a) \geq 0,7$, então as habilidades não estão bem distribuídas entre as pessoas e, neste caso, é atribuída uma baixa probabilidade ($w = 0,1$) a este *swap* e o valor 1 à variável auxiliar s . Caso contrário, as habilidades estão bem distribuídas e uma nova verificação é feita. É verificado se as demandas das duas habilidades que as pessoas mais possuem estão bem distribuídas entre os projetos. Este cálculo é realizado tal qual está descrito na equação (4.9). De igual forma, se algum projeto $p_l \in P$ possui distribuição maior ou igual à 0,7, significa que as demandas destas habilidades não estão bem distribuídas e atribui-se uma probabilidade de 0,3 à este *swap*. Caso contrário, atribui-se uma probabilidade maior

($w = 0,4$). Finalmente, a mesma probabilidade w que é atribuída ao *swap* 2, é decrementada do *swap* 1, uma vez que o somatório das probabilidades deve permanecer igual à 1.

Algoritmo 12: Seleção de Operador Genético

Entrada: Uma instância I .

Saída: Vetor *probOperadores* com as probabilidades dos operadores genéticos.

1 **início**

```

2   |    $prob[swap1] \leftarrow 1; prob[swap2] \leftarrow 0; prob[crossover] \leftarrow 0;$ 
3   |    $probOperadores \leftarrow [prob[swap1], prob[swap2], prob[crossover]];$ 
4   |    $s \leftarrow 0;$ 
5   |    $probOperadores \leftarrow calculaProbSwap2(probOperadores, s);$ 
6   |    $probOperadores \leftarrow calculaProbCrossover(probOperadores, s);$ 
7   |   retorna probOperadores.

```

8 **fim**

O Algoritmo 14 é utilizado para calcular a probabilidade de selecionar o *crossover*. Primeiro, verifica-se se as demandas dos projetos possuem muito tempo fracionado. Define-se *qtdFrac* e *qtdInt* a quantidade de projetos que possuem demandas fracionadas e inteiras, respectivamente. Então, é verificado se $qtdFrac / (qtdFrac + qtdInt) < 0,4$. Neste caso, considera-se que os projetos possuem poucas demandas fracionadas e, assim, é atribuída uma probabilidade maior do que no caso contrário, pois, no *crossover*, quanto menos demandas fracionadas melhor, uma vez que serão necessárias menos correções nas soluções geradas. Logo em seguida, verifica-se se a instância I é de múltiplas habilidades e, se for, significa que $prob[swap2] \neq 0$. Neste caso, verifica-se se a variável auxiliar $s = 0$, isto é, se $prob[swap2] > 0,1$, para garantir que a probabilidade do *swap* 2 não se tornará negativa ao serem atribuídos mais 0,3 de probabilidade ao *crossover* e, conseqüentemente, decrementados 0,15 de $prob[swap2]$. Este valor também é decrementado de $prob[swap1]$. Já se a instância não é de múltiplas habilidades, só é possível decrementar de $prob[swap1]$ e, neste cenário, de acordo com os diferentes conjuntos D , são atribuídas probabilidades distintas ao *crossover*.

Algoritmo 13: *calculaProbSwap2(probOperadores, s)*

Entrada: Vetor *probOperadores* com as probabilidades dos operadores genéticos.

Saída: Probabilidade do *swap 2*.

```

1 início
2   // Definindo a probabilidade do swap 2;
3    $w \leftarrow 0$ ;
4   se a instância I é de múltiplas habilidades então
5     se as habilidades não estão bem distribuídas entre as pessoas então
6       |  $w \leftarrow 0,1; s \leftarrow 1$ ;
7     fim
8     senão
9       se as demandas das habilidades estão bem distribuídas entre os projetos então
10        |  $w \leftarrow 0,4$ ;
11      fim
12      senão
13        |  $w \leftarrow 0,3$ ;
14      fim
15    fim
16     $probOperadores[0] -= w; probOperadores[1] += w$ ;
17  fim
18 fim

```

4.4.1.6 Seleção de Indivíduos para Reprodução

O método *selecaoIndividuosReproducao(probIndividuos)* seleciona os indivíduos (soluções) para reprodução de acordo com as probabilidades de cada um, definidas no parâmetro *probIndividuos*. Este vetor de probabilidades é criado no método *calculoAptidaoPopulacao(populacao)*, onde dada uma população como entrada, é calculada a probabilidade de cada indivíduo ser selecionado para reprodução. Esta probabilidade está definida na equação (4.3) e é diretamente proporcional à aptidão de cada indivíduo.

Algoritmo 14: *calculaProbCrossover(probOperadores, s)*

Entrada: Vetor *probOperadores* com as probabilidades dos operadores genéticos e uma variável auxiliar *s*.

Saída: Probabilidade do *crossover*.

```

1 início
2   // Definindo a probabilidade do crossover;
3    $w \leftarrow 0$ ;
4   se as demandas dos projetos possuem pouco tempo fracionado então
5     |  $w \leftarrow 0,05$ ;
6   fim
7   senão
8     |  $w \leftarrow 0,015$ ;
9   fim
10  se a instância I não é de múltiplas habilidades então
11    | se  $D = [0, 1]$  então
12      |  $w += 0,3$ ;
13    | fim
14    | se  $D = [0; 0,5; 1]$  então
15      |  $w += 0,385$ ;
16    | fim
17    | se  $D = [0; 0,25; 0,5; 0,75; 1]$  então
18      |  $w += 0,485$ ;
19    | fim
20    |  $probOperadores[0] -= 2w$ ;
21  | fim
22  | senão
23    | se  $s = 0$  então
24      |  $w += 0,15$ ;
25    | fim
26    |  $probOperadores[0] -= w; probOperadores[1] -= w$ ;
27  | fim
28  |  $probOperadores[3] += 2w$ ;
29 fim

```

4.4.1.7 Seleção da Nova População

Ao final de cada geração (iteração) do algoritmo, uma nova população deve ser selecionada. Os *swaps* 1 e 2 podem gerar, no máximo, uma nova solução. Já o *crossover* pode criar, no máximo, duas novas soluções. Estas novas soluções são inseridas na população atual, e como esta possui tamanho 50, ficará com, no máximo, 52 soluções. Então, estas soluções são ordenadas de acordo com as suas eficiências e são escolhidas, para a nova população, as 45 melhores soluções e as 5 piores, com o intuito de deixar a população um pouco diversificada.

5 RESULTADOS

Neste capítulo, são comparadas três implementações para o PFMT, problema em que as pessoas possuem apenas uma habilidade: as meta-heurísticas VNS (GUTIÉRREZ *et al.*, 2016), SA (FIGUEIREDO; CAMPÊLO, 2018) e o Algoritmo Genético proposto neste trabalho. Para o $PFMT_{MH}$, onde as pessoas possuem múltiplas habilidades, são apresentados apenas os resultados obtidos através do Algoritmo Genético, uma vez que, por se tratar de um problema novo, não são conhecidos resultados para ele na literatura. Na Seção 5.1, é apresentado o ambiente computacional utilizado para realização dos testes. Na Seção 5.2, são apresentadas as principais características das instâncias utilizadas na realização dos testes computacionais. Por fim, é realizado, na Seção 5.3, um estudo comparativo entre as implementações da literatura e o algoritmo proposto.

5.1 Configuração do Ambiente Computacional

A implementação da meta-heurística Algoritmos Genéticos deste trabalho foi realizada na linguagem de programação *Python*, versão 2.7. Foi utilizado o software matemático SageMath, versão 8.4 (STEIN, 2018), que possui recursos de Teoria dos Grafos. Os experimentos realizados foram executados utilizando uma máquina com processador Intel Pentium i7, 8×3.60 GHz, 16 GB de memória RAM e sistema operacional Linux Ubuntu 14.05.

5.2 Instâncias

As três implementações para o PFMT foram avaliadas através de testes computacionais utilizando as instâncias pseudo-aleatórias recriadas por (FIGUEIREDO; CAMPÊLO, 2018) e geradas como em (GUTIÉRREZ *et al.*, 2016). Além disto, foram criadas novas instâncias pseudo-aleatórias, obtidas por um gerador de instâncias desenvolvido neste trabalho. Estas novas instâncias abrangem a questão das múltiplas habilidades por pessoa e são aplicáveis ao $PFMT_{MH}$.

5.2.1 Instâncias do PFMT

Em (FIGUEIREDO; CAMPÊLO, 2018), foram recriadas o conjunto de 90 instâncias, geradas da mesma maneira que em (GUTIÉRREZ *et al.*, 2016). Estas instâncias estão divididas

em três grupos, de acordo com a porcentagem de relações positivas na matriz sociométrica: 30% para as instâncias do grupo I, 50% para as instâncias do grupo II e 70% para as instâncias do grupo III. Dentro de cada um dos grupos, foram criadas seis classes de instâncias, variando os valores de outros parâmetros (número de projetos, número de pessoas disponíveis, número de habilidades, frações de alocação de tempo), de acordo com a Tabela 1. Por fim, para cada uma das seis classes dentro de cada um dos três grupos, foram geradas aleatoriamente cinco instâncias específicas.

Tabela 1 – Valores dos parâmetros para cada classe de instâncias do PFMT

Classes de Instâncias	Projetos (m)	Pessoas Disponíveis (n)	Habilidades (f)	Frações de Alocação de Tempo (t)
1	2	25	10	0.0 - 1.0
2	5	50	5	0.0 - 1.0
3	2	25	10	0.0 - 0,5 - 1,0
4	5	50	5	0.0 - 0,5 - 1,0
5	2	25	10	0.0 - 0,25 - 0,5 - 0,75 - 1,0
6	5	50	5	0.0 - 0,25 - 0,5 - 0,75 - 1,0

Fonte: (FIGUEIREDO; CAMPÊLO, 2018)

5.2.2 Instâncias do $PFMT_{MH}$

Neste trabalho foi criado um gerador de instâncias pseudo-aleatórias para abranger a questão de pessoas com múltiplas habilidades, onde foram criadas mais três categorias de instâncias, de acordo com a porcentagem de pessoas com múltiplas habilidades: 30% para as instâncias da categoria A, 50% para as instâncias da categoria B e 70% para as instâncias da categoria C. Dentro de cada categoria, foram inseridas as instâncias de 50 vértices criadas em (FIGUEIREDO; CAMPÊLO, 2018), isto é, as classes 2, 4 e 6, modificando apenas a matriz K de habilidades, gerando, assim, 135 instâncias para o $PFMT_{MH}$.

5.3 Estudo Comparativo entre os Algoritmos

A Tabela 2 apresenta os resultados médios tanto para o GAP (porcentagem entre o valor da solução heurística e o valor da solução ótima) quanto para o tempo de resolução, em segundos, dos três algoritmos heurísticos (Algoritmo Genético (GA), VNS (GUTIÉRREZ *et al.*, 2016) e SA (FIGUEIREDO; CAMPÊLO, 2018)) para as instâncias de 50 vértices do PFMT (GUTIÉRREZ *et al.*, 2016), isto é, as classes 2, 4 e 6. Como é possível observar, os GAPs obtidos pelo GA, proposto neste trabalho, são melhores em comparação às outras duas meta-heurísticas, significando que encontramos soluções mais próximas às soluções ótimas.

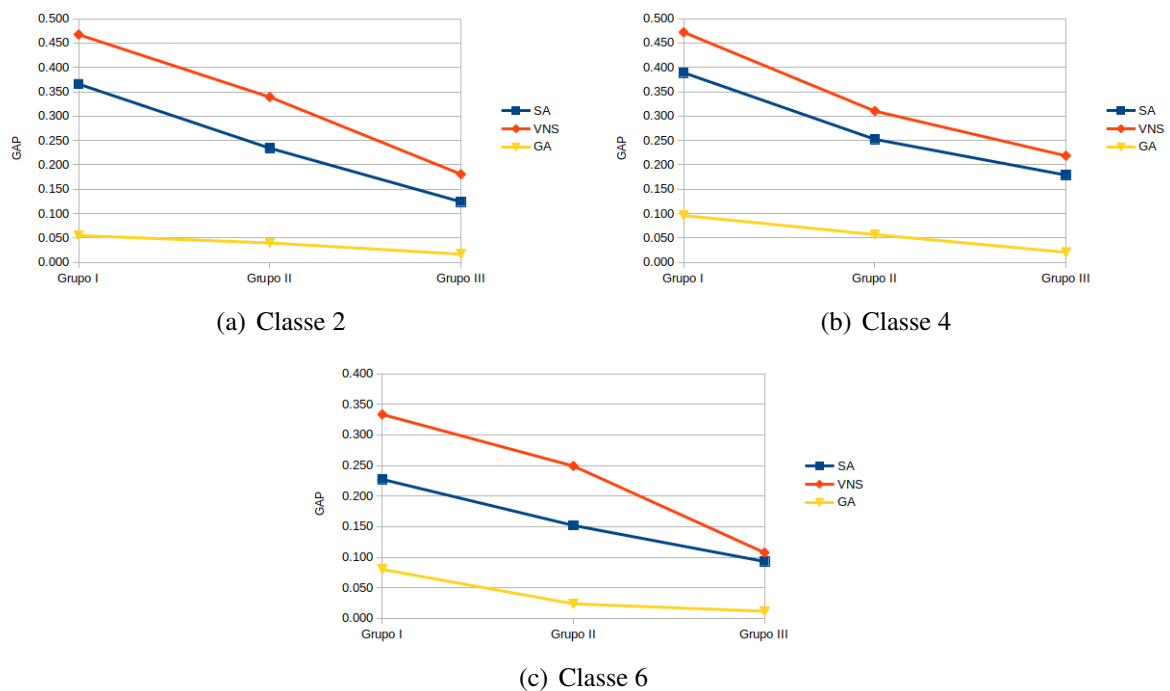
Este resultado se deve, principalmente, a ajustes realizados em operações de mutação realizadas no algoritmo. Através dos experimentos realizados, pode-se notar que o uso do operador de mutação resultou em uma alta redução do GAP. Inicialmente, a escolha do gene que sofreria mutação, ou seja, um projeto, era aleatória, não sendo obtidos resultados satisfatórios. Porém, após a realização de uma atribuição probabilística, relacionando a escolha do gene de forma inversamente proporcional à eficiência de cada projeto, isto é, aqueles menos eficientes possuem maiores chances de serem selecionados, foi possível alcançar resultados com alta qualidade, quando comparado aos algoritmos da literatura. Sobre o tempo de resolução do GA, este é um pouco maior em comparação ao do SA (FIGUEIREDO; CAMPÊLO, 2018) e bem menor em relação ao do VNS (GUTIÉRREZ *et al.*, 2016).

Tabela 2 – Análise comparativa entre as heurísticas, onde o GAP é a diferença entre o valor heurístico e o valor ótimo, e o tempo é medido em segundos.

50 vértices	Classe 2						Classe 4						Classe 6					
	GA		SA		VNS		GA		SA		VNS		GA		SA		VNS	
	Gap	Tempo	Gap	Tempo	Gap	Tempo	Gap	Tempo	Gap	Tempo	Gap	Tempo	Gap	Tempo	Gap	Tempo	Gap	Tempo
Grupo I	0,054	15,713	0,366	6,201	0,467	1118,563	0,096	16,603	0,389	4,005	0,472	948,268	0,08	8,677	0,227	4,235	0,333	1081,382
Grupo II	0,04	17,114	0,234	6,292	0,339	1019,949	0,057	17,316	0,252	6,003	0,310	1293,291	0,024	9,078	0,152	4,382	0,249	912,325
Grupo III	0,017	17,363	0,124	6,309	0,181	1117,204	0,021	17,904	0,179	7,058	0,219	1128,163	0,011	6,843	0,093	6,985	0,107	1208,231

Fonte: Elaborado pela autora (2018).

Figura 8 – GAPs das heurísticas da literatura e do Algoritmo Genético proposto.

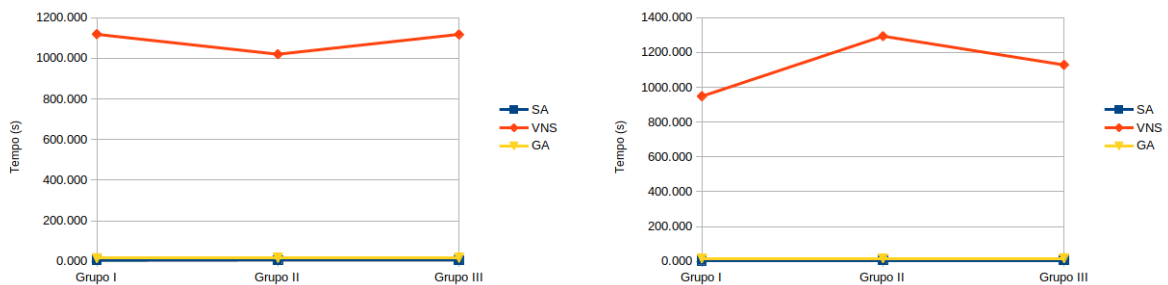


Fonte: Elaborado pela autora (2018).

Os gráficos apresentados na Figura 8 exibem os GAPs (eixo Y) do Algoritmo Genético proposto e das duas heurísticas da literatura, o VNS (GUTIÉRREZ *et al.*, 2016) e SA

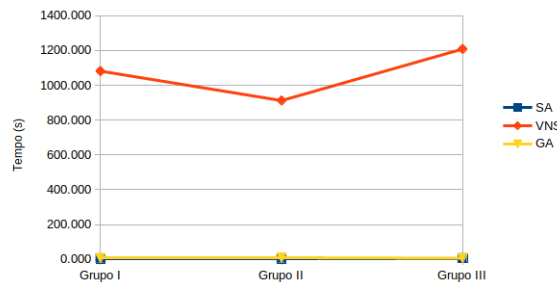
(FIGUEIREDO; CAMPÊLO, 2018), para os grupos I, II e III de instâncias (eixo X) das classes 2, 4 e 6. É possível notar que ambas as heurísticas apresentam um comportamento semelhante, onde os GAPs são maiores para as instâncias do grupo I e menores para as instâncias do grupo III, para ambas as classes. Por este motivo, acredita-se que existe uma relação entre o número de relações positivas na matriz sociométrica e a dificuldade de resolução da instância. Já a Figura 9 apresenta os tempos de execução destes três algoritmos.

Figura 9 – Tempos de execução das heurísticas da literatura e do Algoritmo Genético proposto.



(a) Classe 2

(b) Classe 4



(c) Classe 6

Fonte: Elaborado pela autora (2018).

Em relação às novas instâncias criadas para o $PFMT_{MH}$, onde as pessoas possuem múltiplas habilidades, são apresentados, na Tabela 3, os resultados médios tanto para as eficiências das soluções encontradas com o Algoritmo Genético proposto, quanto para o tempo de resolução, em segundos. Para estas instâncias, como não são conhecidas as soluções ótimas, não é possível apresentar uma análise utilizando o conceito de GAP. No entanto, a eficiência máxima possível de uma solução para este problema é 1, de acordo com a Subseção 4.4.1.2. Logo, pode-se notar que as eficiências encontradas pelo algoritmo proposto são próximas ao máximo valor possível.

Tabela 3 – Análise dos resultados obtidos com o Algoritmo Genético proposto para as novas instâncias geradas. A eficiência é calculada como definido na Subseção 4.4.1.2 e o tempo é medido em segundos.

50 vértices		Classe 2		Classe 4		Classe 6	
		Eficiência	Tempo	Eficiência	Tempo	Eficiência	Tempo
Categoria A	Grupo I	0,874	15,285	0,87	17,964	0,92	11,852
	Grupo II	0,932	16,831	0,94	17,3	0,939	10,06
	Grupo III	0,971	17,045	0,968	17,264	0,98	9,246
Categoria B	Grupo I	0,887	17,748	0,88	18,909	0,909	10,043
	Grupo II	0,932	18,515	0,924	18,145	0,942	12,231
	Grupo III	0,963	18,138	0,959	18,06	0,977	9,708
Categoria C	Grupo I	0,872	19,178	0,885	18,521	0,904	10,621
	Grupo II	0,928	18,986	0,929	18,948	0,954	12,472
	Grupo III	0,972	18,251	0,971	18,234	0,978	11,581

Fonte: Elaborado pela autora (2018).

6 CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho definiu o Problema de Formação de Múltiplos Times com Múltiplas Habilidades ($PFMT_{MH}$) como sendo uma generalização de problemas semelhantes encontrados na literatura (GUTIÉRREZ *et al.*, 2016; CAMPÊLO *et al.*, 2018). Este é um problema do mundo real, aplicável a empresas e organizações, com o objetivo de formar as melhores equipes de trabalho possíveis, considerando as relações sociais entre as pessoas, uma vez que estas são tão importantes e influenciam diretamente no sucesso da equipe (BALLESTEROS-PÉREZ *et al.*, 2012).

Foi apresentado um algoritmo baseado na meta-heurística Algoritmos Genéticos e posteriormente realizados testes computacionais para as instâncias específicas do PFMT, onde as pessoas possuem apenas uma habilidade. Os resultados obtidos foram comparados com as soluções alcançadas com os algoritmos *Variable Neighborhood Search* (VNS) e *Simulated Annealing* (SA) de (GUTIÉRREZ *et al.*, 2016) e (FIGUEIREDO; CAMPÊLO, 2018), respectivamente. O algoritmo proposto neste trabalho apresentou melhores resultados, em termos de GAP, ou seja, apresentamos soluções mais próximas às soluções ótimas, levando um pouco mais de tempo de resolução, no máximo 10 segundos a mais que o algoritmo SA, com o melhor resultado da literatura, proposto por Figueiredo e Campêlo (2018). Também foram realizados experimentos com as novas instâncias geradas para o $PFMT_{MH}$, incluindo pessoas com múltiplas habilidades. Como não há conhecimento sobre os valores das soluções ótimas destas instâncias, não foi apresentado uma análise utilizando o conceito de GAP. No entanto, sabe-se que o valor máximo possível para qualquer solução deste problema é 1 e os resultados obtidos ficaram próximos a este valor. Como trabalhos futuros, no intuito de conhecer os valores ótimos das instâncias geradas neste trabalho, espera-se desenvolver um modelo matemático para o $PFMT_{MH}$, além de realizar testes computacionais com o possível modelo proposto, assim como possíveis aprimoramentos no algoritmo apresentado. Além disso, outras heurísticas e meta-heurísticas podem ser investigadas para este novo problema.

REFERÊNCIAS

- BAILEY, D. E. Manufacturing improvement team programs in the semiconductor industry. **IEEE Transactions on semiconductor manufacturing**, IEEE, v. 10, n. 1, p. 1–10, 1997.
- BALLESTEROS-PÉREZ, P.; GONZÁLEZ-CRUZ, M. C.; FERNÁNDEZ-DIEGO, M. Human resource allocation management in multiple projects using sociometric techniques. **International Journal of Project Management**, Elsevier, v. 30, n. 8, p. 901–913, 2012.
- CAMPÊLO, M.; FIGUEIREDO, T.; SILVA, A. The sociotechnical teams formation problem: a mathematical optimization approach. **Annals of Operations Research**, Springer, p. 1–16, 2018.
- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos: teoria e prática**. Rio de Janeiro: Editora Campus, 2012. v. 3.
- CORTES, O. A. C. Integração entre lógica nebulosa e algoritmos evolutivos. **INFOCOMP**, v. 3, n. 1, p. 36–42, 2004.
- EIBEN, A.; RUTTKAY, Z. Self-adaptivity for constraint satisfaction: Learning penalty functions. In: IEEE. **Evolutionary Computation, 1996., Proceedings of IEEE International Conference**. IEEE, 1996. p. 258–261.
- FIGUEIREDO, T.; CAMPÊLO, M. The multiple team formation problem. **Latin-Iberoamerican Conference on Operations Research**, Lima, Perú, 2018.
- FITZPATRICK, E. L.; ASKIN, R. G. Forming effective worker teams with multi-functional skill requirements. **Computers & Industrial Engineering**, Elsevier, v. 48, n. 3, p. 593–608, 2005.
- GOLDBERG, A. V.; TARJAN, R. E. A new approach to the maximum-flow problem. **Journal of the ACM (JACM)**, ACM, v. 35, n. 4, p. 921–940, 1988.
- GUTIÉRREZ, J. H.; ASTUDILLO, C. A.; BALLESTEROS-PÉREZ, P.; MORA-MELIÀ, D.; CANDIA-VÉJAR, A. The multiple team formation problem using sociometry. **Computers & Operations Research**, Elsevier, v. 75, p. 150–162, 2016.
- HILLIER, F. S.; LIEBERMAN, G. J. **Introdução à pesquisa operacional**. São Paulo: McGraw Hill Brasil, 2013.
- HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. Ann Arbor, Michigan, Estados Unidos: University of Michigan Press, 1975.
- LAPPAS, T.; LIU, K.; TERZI, E. Finding a team of experts in social networks. In: ACM. **Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining**. Paris, França, 2009. p. 467–476.
- LIU, C.; YANG, S. A hybrid genetic algorithm for integrated project task and multi-skilled workforce scheduling. **Journal of Computational Information Systems**, v. 7, n. 6, p. 2187–2194, 2011.
- LUKE, S. **Essentials of Metaheuristics**. second. Morrisville, Carolina do Norte, Estados Unidos: Lulu, 2013. Disponível em <https://cs.gmu.edu/~sean/book/metaheuristics/Essentials.pdf>.

MAURER, I. How to build trust in inter-organizational projects: The impact of project staffing and project rewards on the formation of trust, knowledge acquisition and product innovation. **International journal of project management**, Elsevier, v. 28, n. 7, p. 629–637, 2010.

MENDES, G. Otimização da localização de poços de petróleo com completção seca utilizando algoritmos genéticos. **Pontifícia Universidade Católica do Rio de Janeiro**, 2013.

MORENO, J. L. Foundations of sociometry: An introduction. **Sociometry**, JSTOR, p. 15–35, 1941.

SILVA, E. E. d. **Otimização de estruturas de concreto armado utilizando algoritmos genéticos**. Tese (Doutorado) — Universidade de São Paulo, 2001.

SOUZA, S.; MACEDO, R.; VARGAS, E.; COURY, D.; OLESKOVICZ, M. Estimação de parâmetros de um sistema elétrico de potência utilizando algoritmos genéticos. **IEEE Latin America Transactions**, v. 4, n. 1, p. 47–54, 2006.

STEIN, W. **SageMath**. 2018. Disponível em: <http://www.sagemath.org/>. Acesso em: 10 de novembro de 2018.