Trabalho de Final da Disciplina de Otimização Multiobjetivo
Universidade Federal de Minas Gerais
Programa de Pós-Graduação em Engenharia Elétrica – 2021/01

ALGORITMO GENÉTICO APLICADO AO PROBLEMA DO CAIXEIRO VIAJANTE MULTIOBJETIVO

Lucas Geovani Nardo
Universidade Federal de Minas Gerais

Lucas Jorge Caldeira Carvalho
Universidade Federal de Minas Gerais

Roberto Moura Passos Júnior
Universidade Federal de Minas Gerais

Resumo — Apesar de fácil descrição, o Problema do Caixeiro Viajante apresenta um elevado grau de complexidade computacional. Devido à grande aplicabilidade dos conceitos envolvidos neste problema, algoritmos vêm sendo desenvolvidos com o intuito de solucioná-lo. Neste trabalho é desenvolvida uma metodologia de otimização multiobjetivo baseada no algoritmo genético NSGA-II, visando otimizar o problema. Posteriormente, é feita uma análise com o uso de indicadores para medir a qualidade destas soluções.

***Palavras-chave* — Problema do Caixeiro Viajante, algoritmo genético, NSGA II, otimização multiobjetivo.**

I. INTRODUÇÃO

O problema matemático relacionado para o Problema do Caixeiro Viajante (PCV) foi tratado em 1857, pelo matemático irlandês Sir Willian Rowan Hamilton, que propôs um jogo montado sobre um dodecaedro, onde cada vértice estava associado a uma cidade importante da época, que denominou *Around the World*. O desafio proposto pelo jogo consistia em encontrar uma rota através dos vértices do dodecaedro que iniciasse e terminasse em uma mesma cidade sem nunca repetir uma visita. Uma solução, para jogo de Hamilton, passou a ser denominada de ciclo hamiltoniano, em sua homenagem [1].

O PCV tem sido muito utilizado no experimento de diversos métodos de otimização por ser, principalmente, um problema de fácil descrição e compreensão, grande dificuldade de solução, uma vez que é NP-Árduo, e larga aplicabilidade [2], com grande aplicação em problemas de transporte e logística [3], perfuração de placas de circuito impresso, revisão de motores de turbina a gás, cristalografia de raio-X, fiação de computadores, problema da ordem de coleta em armazéns, roteamento de veículos, dentre outros [4].

II. OBJETIVO

Modelar matematicamente o Problema do Caixeiro Viajante e resolvê-lo desenvolvendo um algoritmo genético NSGA-II.

Uma análise das soluções encontradas deverá ser feita com o uso de indicadores de qualidade proposto pela literatura, e a ação final executada pelo algoritmo com o uso de métodos de auxílio à tomada de decisão.

III. MODELAGEM DO PCV

Matematicamente, o PCV é descrito como um grafo G = (V,E), onde V = {1,...,n} é o conjunto dos vértices do grafo (cada um representando uma cidade) e E = {(i,j)| i, j = 1,...,n} é um conjunto de arcos ligando esses vértices (representando um caminho entre pares de cidades). Associado a cada arco existe um custo c_{ij} (distância entre as cidades), tal que $c_{ii}=\infty$. A variável binária x_{ij} somente receberá o valor 1 se a aresta (i,j) $\in$ E for escolhida para pertencer ao *tour*, e 0 caso contrário O problema consiste na determinação de um caminho hamiltoniano de custo mínimo sobre G [2].

Podemos modelar o PVC como

$$min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \quad (1)$$

sujeito a

$$\sum_{j \in V} x_{ij} = 1, \; i \in V \quad (2)$$

$$\sum_{i \in V} x_{ij} = 1, \; j \in V \quad (3)$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \forall S \subset V, S \neq \emptyset \quad (4)$$

$$x_{ij} \in \{0,1\}; \; i, j \in V \quad (5)$$

O PCV multiobjetivo consiste em um grafo G = (V, E, w), onde V é o conjunto de vértices, E o conjunto de arestas e w é uma função que atribui a cada aresta (e_{ij} $\in$ E) um vetor (w_{ij}^1 ,..., w_{ij}^M). Cada elemento w_{ij}^m corresponde a um determinado peso, de métricas diferentes, por exemplo, distância e o tempo para uma aresta de um problema biobjetivo [1].

O PVC proposto por este trabalho consistem encontrar o ciclo hamiltoniano que minimiza o tempo e a distância gastos para, saindo de uma cidade x, visitar outras 249 cidades e voltar a x. Uma formulação é vista abaixo.

$$min(f_1 e f_2) \quad (6)$$

sujeito a

$$f_1(C) = \sum_{e \in C} w^m(e) \quad (7)$$

$$f_2(C) = \sum_{e \in C} w^m(e) \quad (8)$$

$$C \in F \quad (8)$$

Sendo F um conjunto dos ciclos hamiltonianos, C um ciclo hamiltoniano, $w_{ij}^m \geq 0$ um valor atribuído pelo

critério (m) a aresta e_{ij} , $f_1(C)$ o tempo e $f_2(C)$ a distância.

IV. MÉTODO DE SOLUÇÃO DO PROBLEMA

Dado um número n de cidades a serem visitadas, o total de rotas possíveis cobrindo todas as cidades é dado por [4]:

$$\frac{(n-1)!}{2} \tag{9}$$

Para um número de cidades igual a 250, proposta deste trabalho, a quantidade combinacional é praticamente infinita.

Algoritmos Genéticos são amplamente utilizados em diversos problemas. Embora não apresentem garantia de encontrar soluções ótimas, normalmente são utilizados quando o custo para encontrar tais soluções é muito elevado ou mesmo desconhecido [6], justificando sua ampla aplicação no PCV.

A escolha do NSGA-II (*Non-dominated Sorting Genetic Algorithm*) foi feita não só por ele está entre os mais tradicionais algoritmos genéticos multiobjetivos, mas também por apresentar bons resultados em *benckmarks* e por usar uma abordagem elitista que ajuda em uma boa convergência para a fronteira Pareto [7]. Além disso, o NSGA-II é considerado um dos algoritmos mais eficazes para resolução de problemas multiobjetivos [8].

V. NSGA-II

O algoritmo proposto por este trabalho foi desenvolvido usando software Matlab®.

A. GERAÇÃO DA POPULAÇÃO INICIAL

Um ponto chave do algoritmo é a escolha da representação dos indivíduos que irão compor a população (conjunto de indivíduos).

Um indivíduo é uma possível solução para o problema. Em outras palavras, um indivíduo, no PCV, é um ciclo hamiltoniano.

Devido à restrição que diz que o caixeiro não pode visitar a mesma cidade duas vezes, o indivíduo foi representado como um vetor com n+1 posições, sendo n o número de cidades a serem visitadas. Este vetor, é uma permutação dos números inteiros de 1 a n, e a posição n+1 é igual à primeira posição, representando a volta o caixeiro à cidade de origem (Figura 1).

Para isso, foi criada uma classe chamada *Indivíduo* que possui todos os atributos referentes a esta entidade que serão necessários para o desenvolvimento do programa:

1. *representação*: genótipo do indivíduo, ou seja, como ele será representado computacionalmente. Para este trabalho, conforme descrito, será um vetor permutação de n inteiros, com tamanho n+1, cuja posição n+1 tem o mesmo valor da posição 1;
2. *fitness*: avalia o valor do indivíduo em relação às funções objetivo.
3. *rank*: armazena o valor do rank daquele indivíduo, indicando em qual fronteira ele se encontra.
4. *conjuntoDominante*: conjunto de todos os indivíduos da população que aquele indivíduo domina.
5. *contadorDominado*: valor que mostra quantos indivíduos da população dominam aquele indivíduo.
6. *crowdingDistance:* valor da distância de multidão do indivíduo.

A classe *Indivíduo* possui um único método: o método construtor.

Este método é chamado sempre que uma objeto daquela classe é instanciado, e é nele que a representação do indivíduo está sendo gerada.

A população inicial é gerada de forma aleatória.

A Figura 1 mostra a representação de um indivíduo gerado de forma aleatória para um número de cidades igual a 4. Nele, o caixeiro parte da cidade 4 e visita a cidade 1; da cidade 1 ele visitar a cidade 3; da cidade 3 ele visitar a cidade 2; após isso, ele retorna à cidade 4.

4	1	3	2	4

Figura 1. Exemplo da representação de um indivíduo para n igual a 4

```
function [populacaoInicial] = gerar_populacao(Parametros)

    numeroCidades = Parametros.numeroCidades;
    tamanhoPopulacao = Parametros.tamanhoPopulacao;

    %Geração aleatória através de uma permutação
    for i = 1:tamanhoPopulacao
        populacaoInicial(i) = Individuo(numeroCidades);
    end
end
```

Figura 2. Função para gerar a população inicial

```
classdef Individuo

    properties
        representacao;
        fitness;
        rank;
        conjuntoDominante;
        contadorDominado;
        crowdingDistance;
    end

    methods
        function obj = Individuo(numeroCidades)
            obj.representacao=randperm(numeroCidades, numeroCidades);
            obj.representacao(numeroCidades + 1) = obj.representacao(1); %Volta para a primeira cidade
            obj.fitness = [];
            obj.rank = [];
            obj.conjuntoDominante=[];
            obj.contadorDominado = [];
            obj.crowdingDistance = [];
        end
    end
end
```

Figura 3. Classe Indivíduo

B. CÁLCULO DO FITNESS

Para cada indivíduo da população é feito o cálculo do fitness (aptidão), conforme visto na Figura 4.

O fitness foi representado computacionalmente por um vetor de duas dimensões que armazena o tempo e a distância total gasto por aquele ciclo hamiltoniano representado pelo indivíduo.

```
%Cálculo da aptidão de cada indivíduo
for i = 1:Parametros.tamanhoPopulacao
    populacao(i).fitness = fitness(populacao(i),Parametros,Datasets);
end
```

Figura 4. Cálculo do fitness de cada indivíduo da população

```
function aptidao = fitness(individuo,Parametros,Datasets)

    distanciaTotal = 0;
    tempoTotal = 0;
    for i = 1:(Parametros.numeroCidades)
        j = i + 1;
        pos_i = individuo.representacao(i);
        pos_j = individuo.representacao(j);
        tempoTotal = tempoTotal + Datasets.tempo(pos_i,pos_j);
        distanciaTotal = distanciaTotal + Datasets.distancia(pos_i,pos_j);
    end
    aptidao(1) = tempoTotal;
    aptidao(2) = distanciaTotal;
end
```

Figura 5. Função para o cálculo do fitness

C. CLASSIFICAÇÃO DA POPULAÇÃO EM FRONTEIRAS

A classificação da população em fronteiras é feita utilizando o conceito de dominância Pareto (Figura 6).

Cada indivíduo da população é comparado com os demais em relação à dominância Pareto. Caso o indivíduo i domine o indivíduo j, a posição do indivíduo j no vetor população é armazenado no atributo *conjuntoDominante* do indivíduo i, e soma-se um no atributo *contradorDominado* do indivíduo j.

Depois da verificação de todos os membros da população, os indivíduos cujo atributo *contadorDominado* é igual a zero recebem o valor um no atributo *rank*, formando o conjunto Pareto ótimo estimado até aquele momento. Essa fronteira é chamada de fronteira não-dominada.

Após isso, é feita uma verificação para saber quais indivíduos são dominados apenas pelos indivíduos de *rank* igual a 1. Esses indivíduos recebem o valor de *rank* igual a 2, formando-se a segunda fronteira. Esse procedimento é repetido até a formação da enésima fronteira (Figura 8).

Figura 6. Dominância Pareto

Fonte: Dias, 2016

```
tamanhoPopulacao = numel(populacao);

for i=1:tamanhoPopulacao
    populacao(i).conjuntoDominante=[];
    populacao(i).contadorDominado=0;
end

fronteira{1}=[];

for i=1:tamanhoPopulacao
    for j=i+1:tamanhoPopulacao
        p = populacao(i);
        q = populacao(j);

        if  all(p.fitness <= q.fitness) && any(p.fitness < q.fitness)
            p.conjuntoDominante = [p.conjuntoDominante j];
            q.contadorDominado = q.contadorDominado + 1;
        end

        if  all(q.fitness <= p.fitness) && any(q.fitness < p.fitness)
            q.conjuntoDominante = [q.conjuntoDominante i];
            p.contadorDominado = p.contadorDominado + 1;
        end

        populacao(i) = p;
        populacao(j) = q;
    end

    if populacao(i).contadorDominado == 0
        fronteira{1}=[fronteira{1} i];
        populacao(i).rank = 1;
    end
end
```

Figura 7. Comparação dos indivíduos por dominância Pareto e formação da primeira fronteira

```
k=1;

while true

    Q = [];

    for i = fronteira{k}
        p = populacao(i);

        for j = p.conjuntoDominante
            q = populacao(j);

            q.contadorDominado = q.contadorDominado - 1;

            if q.contadorDominado == 0
                Q = [Q j];
                q.rank = k+1;
            end

            populacao(j) = q;
        end
    end

    if isempty(Q)
        break;
    end

    fronteira{k+1} = Q;

    k=k+1;

end
```

Figura 8. Com exceção da primeira fronteira, formação das demais

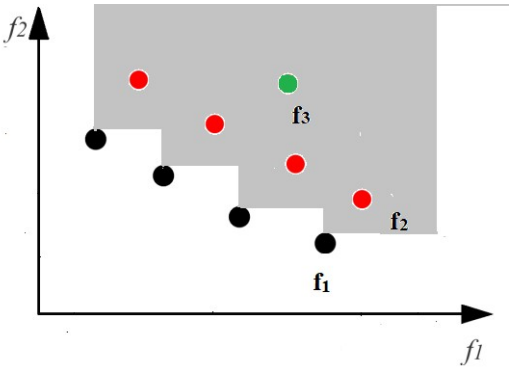

Figura 9: Separação da população em fronteiras

D. CÁLCULO DA DISTÂNCIA DE MULTIDÃO

O valor da métrica *crowding distance* (CD), também chamada de distância de multidão ou distância de aglomeração, de cada solução se baseia na distância em que sua imagem se encontra das imagens vizinhas mais próxima para cada função objetivo dentro de um mesmo posto (rank). Quanto maior o valor de CD, mais distante a solução se encontra de suas vizinhas no espaço dos objetivos, tornando maior sua preferência na seleção para a próxima geração da população. Para manter as soluções candidatas que têm imagem nas extremidades de um posto,é atribuído valor infinito aos seus CDs. Em um problema com duas funções objetivo, CD é o semiperímetro de um retângulo cujos vértices ão as imagens vizinhas mais próximas [10] (Figura 10).

Após o cálculo, o valor de CD é salvo no atributo *crowingDistance* do indivíduo.

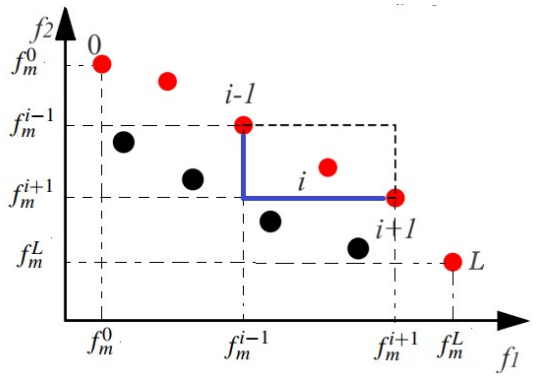

Figura 10: Distância de aglomeração

```
function populacao = distancia_aglomeracao_2(populacao,fronteira)
    quantidadeFronteiras = numel(fronteira);
    %Cálculo para cada fronteira
    for k=1:quantidadeFronteiras
        nIndividuosFronteira = length(fronteira{k});
        if nIndividuosFronteira == 1
            populacao(fronteira{k}).crowdingDistance = inf;
            continue;
        elseif nIndividuosFronteira == 2
            populacao(fronteira{k}(1)).crowdingDistance = inf;
            populacao(fronteira{k}(2)).crowdingDistance = inf;
            continue;
        else
            individuosFronteira = populacao(fronteira{k});
            fitness = [];
            for i = 1:nIndividuosFronteira
                fitness(i) = individuosFronteira(i).fitness(1);
            end
            %Ordenamento por fitness
            [~,index] = sort(fitness);
            individuosFronteira = individuosFronteira(index);
            for j = 1:nIndividuosFronteira
                if j == 1 || j == nIndividuosFronteira
                    d(j) = inf;
                else
                    soma = 0;
                    for i = 1:2
                        numerador = abs(individuosFronteira(j+1).fitness(i)+individuosFronteira(j-1).fitness(i));
                        denominador = abs(individuosFronteira(1).fitness(i)+individuosFronteira(nIndividuosFronteira).fitness(i));
                        soma = soma + numerador/denominador;
                    end
                    d(j) = soma;
                end
                populacao(fronteira{k}(index(j))).crowdingDistance = d(j);
            end
        end
    end
end
```

Figura 11. Cálculo da distância de multidão

E. CLASSIFICAÇÃO DA POPULAÇÃO POR RANK E POR DISTÂNCIA DE MULTIDÃO

A população é classificada em ordem crescente de rank e em ordem decrescente de distância de multidão.

Desta forma, criasse uma hierarquização de preferência entre as duas métricas: os melhores indivíduos são aqueles com melhor *rank*. Caso os indivíduos possuam o mesmo *rank*, aquele que tiver a maior distância de multidão será considerado o mais apto para cruzamento e para fazer parte da próxima geração da população.

Representação					Rank	CD
4	1	3	2	4	1	∞
1	4	3	2	1	2	∞
3	4	2	1	3	2	∞
4	2	3	1	4	2	4
2	4	3	1	2	2	2
3	2	4	1	3	3	∞

Figura 12: Exemplo de uma população classificada

```
function [populacao, fronteira]=classificacao_populacao(populacao)

    % Classificação baseada na crowding distance
    [~, cdDescendente] = sort([populacao.crowdingDistance],'descend');
    populacao = populacao(cdDescendente);

    % Classificação baseada em rank
    [~, rankAscendente]=sort([populacao.rank]);
    populacao=populacao(rankAscendente);

    % Atualizar fronteiras
    ranks = [populacao.rank];
    maxRank = max(ranks);
    fronteira = cell(maxRank,1);

    for k=1:maxRank
        fronteira{k} = find(ranks==k);
    end

end
```

Figura 13. Classificação dos indivíduos da população

F. TORNEIO

A seleção dos indivíduos pais que farão parte do cruzamento para gerar os indivíduos filhos foi feita por meio do método denominado torneio.

São gerados dois números aleatórios diferentes entre um e o tamanho da população. Estes números representam as posições, em relação ao vetor população, dos dois indivíduos que disputarão o torneio.

Os números gerados aleatoriamente são colocados em ordem crescente. O indivíduo vencedor é aquele que tem a posição do menor número aleatório.

Isso só pode ser feito porque no passo anterior do algoritmo houve a classificação da população.

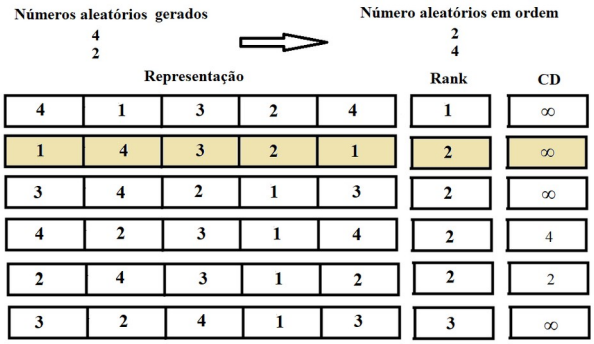

Figura 14: Exemplo de torneio

```
function [campeoes] = torneio(populacao)

    tamanhoPopulacao = numel(populacao);

    for k = 1:tamanhoPopulacao

        %Seleciona n indivíduos para o torneio
        aleatorio = randperm(tamanhoPopulacao,2);
        aleatorioCrescente = sort(aleatorio);

        %Seleciona o de menor índice (mais apto)
        campeoes(k) = populacao(aleatorioCrescente(1));
    end
end
```

Figura 15. Seleção por torneio

G. CRUZAMENTO

No cruzamento, os filhos gerados herdam parte do cromossomo dos pais. Ou seja, a representação dos filhos é uma mistura da representação dos pais (Figura 16).

Os pais selecionados para o torneio são cruzados em pares.

Um número aleatório entre dois e o tamanho do cromossomo menos um é gerado. Esse número representa a posição em que o cromossomo dos pais serão recombinados para gerar os filhos.

Os filhos gerados por esse processo, no caso do PCV, podem ser inválidos (não serem permutações), conforme pode ser visto na Figura 12. Então, é feito um procedimento para substituir os valores repetidos dos cromossomos.

Por último, repetimos o primeiro elemento da representação na posição n+1, para que o caixeiro retorne à cidade de origem.

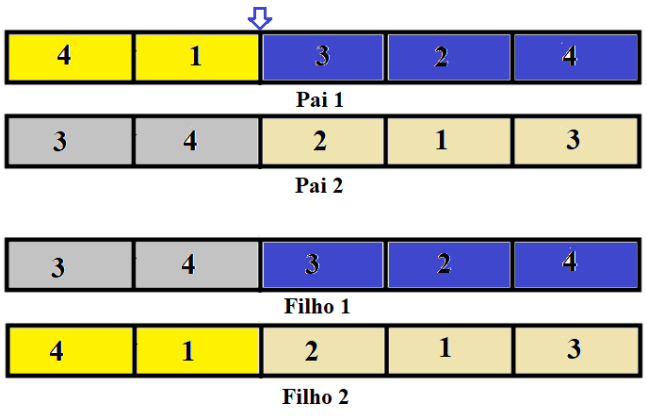

Figura 16. Cruzamento entre dois pais para gerar dois filhos

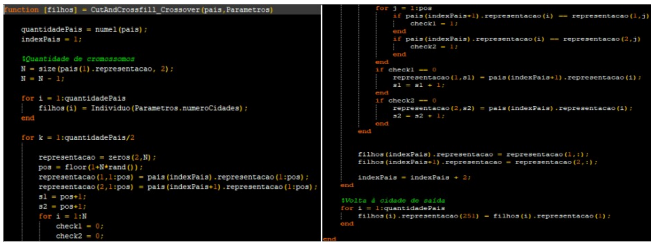

Figura 17. Cruzamento

H. MUTAÇÃO

A mutação acontece para que seja inserida na população características que não poderiam ser obtidas apenas pelo cruzamento.

Existe um parâmetro no algoritmo, escolhido pelo operador, que armazena a probabilidade de uma mutação acontecer.

Para cada indivíduo da população é gerado um número aleatório. Caso esse número seja menor ou igual à probabilidade de mutação, seleciona-se dois genes daquele indivíduo e troca-os de lugar. Isso faz com que o indivíduo resultante seja válido.

De modo a preservar a característica do indivíduo de ser uma permutação, são gerados dois números aleatórios inteiros distintos que representam a posição dos genes que serão invertidos, conforme mostrado na Figura 18.

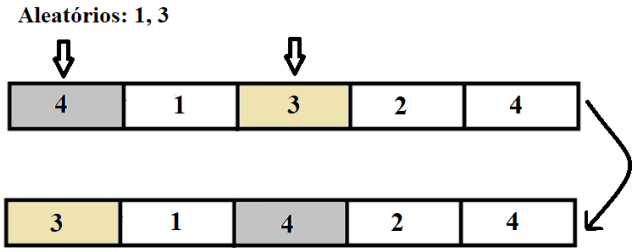

Figura 18: Exemplo de mutação

```
function [filhos] = mutacao(filhos,Parametros)

    tamanhoPopulacao = numel(filhos);
    nCidades = Parametros.numeroCidades;

    for k = 1:tamanhoPopulacao
        aleatorio = rand;
        if aleatorio <= Parametros.probabilidadeMutacao
            indexTroca = randperm(nCidades, 2);
            gene_1 = filhos(k).representacao(indexTroca(1));
            gene_2 = filhos(k).representacao(indexTroca(2));
            filhos(k).representacao(indexTroca(1)) = gene_2;
            filhos(k).representacao(indexTroca(2)) = gene_1;
            filhos(k).representacao(nCidades+1) = filhos(k).representacao(1);
        end
    end
end
```

Figura 19. Mutação

I. INDIVÍDUOS QUE COMPÕEM AS PRÓXIMAS GERAÇÕES

Após a mutação, é calculado o fitness dos filhos, eles são inseridos na população e são repetidos os passos de C a E.

Os N piores indivíduos são excluídos da população, onde N é o tamanho da população.

```
%Cálculo da aptidão dos filhos
quantidadeFilhos = numel(filhos);
for i = 1:quantidadeFilhos
    filhos(i).fitness = fitness(filhos(i),Parametros,Datasets);
end

%Inserir os filhos na população
populacao = [populacao filhos];

%Classificação da população em ranks
[populacao, fronteira] = classificacao_dominancia(populacao);

% Cálculo da distância de aglomeração
populacao = distancia_aglomeracao_2(populacao,fronteira);

%Ordenamento da população pelo rank e distância de aglomeração,
%respectivamente
[populacao, fronteira] = classificacao_populacao(populacao);

%Excluir os N piores indivíduos da população
tamanhoPopulacao = numel(populacao);
populacao = populacao(1:tamanhoPopulacao/2);
```

Figura 20. Próxima geração da população

VI. INDICADORES DE QUALIDADE

Com a popularização dos algoritmos evolutivos, a necessidade de comparar suas respostas se tornou algo tão relevante quanto seu desenvolvimento. Para avaliar estas respostas, utilizamos o que chamamos de Indicadores de Desempenho. Estes indicadores também são conhecidos como medidas de desempenho, ou métricas, e tem como objetivo quantificar a qualidade de um dado conjunto, ou comparar dois conjuntos [11].

Segundo [11], podemos dividir os indicadores de qualidade em 4 classes:

1. Indicadores de capacidade: medem o número de vetores em um dado conjunto ou a razão entre a quantidade de elementos da fronteira Pareto Estimada (FA) e a fronteira Pareto Ótima (FR).
2. Indicadores de convergência: avaliam a proximidade entre os conjuntos FA e FR.
3. Indicadores de diversidade: São divididos em dois grupos: 1) Distribuição: mede o quão uniformemente distribuídos estão os pontos de FA, e 2) Cobertura: indica o quanto os vetores em FA se aproximam dos extremos de FR.
4. Indicadores de convergência-diversidade: Avaliam tanto a convergência quanto a diversidade em uma única escala.

A. INDICADOR Δ

O indicador Δ, ou indicador SPREAD, é um indicador de diversidade que mede a extensão do espalhamento obtido pelas soluções encontradas [12].

Além disso, esse indicador tem a capacidade de medir a convergência das soluções extremas da FA.

Primeiramente, calculamos a distância Euclidiana d_i entre soluções consecutivas do grupo de soluções não dominadas. Então, calculamos a média $\bar{d}$ dessas distâncias. Depois disso, do grupo de soluções não dominadas, calculamos as soluções extremas, que não fazem parte das soluções encontradas, ajustando uma curva paralela à da Fronteira Ótima de Pareto [12].

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} |d_i - \bar{d}|}{d_f + d_l + (N-1)\bar{d}} \tag{10}$$

Onde, N é o número de soluções existentes no grupo de soluções não dominadas.

Quanto mais perto de zero o indicador estiver melhor a solução encontrada pelo algoritmo, pois, nesse caso, a distância média entre as soluções seria a mesma de uma solução a outra, configurando uma distribuição uniforme, e e a distância entre as extremidades da FA e da FR seriam muito próximas.

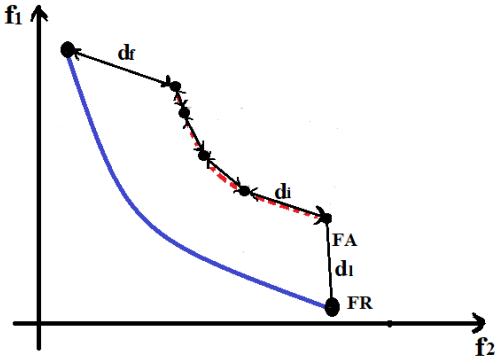

Figura 21. Representação gráfica do indicador Δ

```
function [delta] = indicador_delta(A,Pontos)
    N = numel(A);
    if N ~= 1
        aTempo = [];
        aDistancia = [];
        A_aux = A;
        for i = 1:N
            aTempo(i) = A(i).fitness(1);
            aDistancia(i) = A(i).fitness(2);
        end
        % Classificação baseado em f1
        [~, f1Crescente] = sort(aTempo);
        A = A_aux(f1Crescente);
        dle = 0;
        d2e = 0;
        di = 0;
        for i = 1:N
            if i == 1
                dle = dle + sum(sqrt((A(i).fitness-Pontos.extremoSuperior).^2));
                di = di + sum(sqrt((A(i).fitness-A(i+1).fitness).^2));
            elseif i == N
                d2e = d2e + sum(sqrt((A(i).fitness-Pontos.extremoInferior).^2));
                di = di + sum(sqrt((A(i).fitness-A(i-1).fitness).^2));
            else
                distancia(1) = sum(sqrt((A(i).fitness-A(i-1).fitness).^2));
                distancia(2) = sum(sqrt((A(i).fitness-A(i+1).fitness).^2));
                di = di + min(distancia);
            end
        end
        media = di/N;
        delta = (dle+d2e+abs(di-media))/(dle+d2e+N*media);
    else
        delta = 1;
    end
end
```

Figura 22. Função que calcula o indicador Δ

B. INDICADOR HIPERVOLUME S-METRIC

O indicador hipervolume s-metric mede tanto a diversidade da fronteira quanto a convergência.

É calculado, para o caso de duas funções objetivo, a área entre os pontos que compõem a fronteira estimada e um ponto de referência. Um alto valor para este indicador significa que FA está muito próxima à FR, e maior a diversidade da população. Sua interpretação geométrica é encontrada na Figura 16.

Esse indicador é calculado da seguinte forma:

$$HV(FA) := \{U_i area_i : x_i \in FA\} \quad (11)$$

Antes de realizar o cálculo do indicador, foi feita uma verificação para ver quais pontos da fronteira não-dominada (seria a FA de cada geração) dominavam o ponto de referência. Somente para esses pontos foram usados para o cálculo.

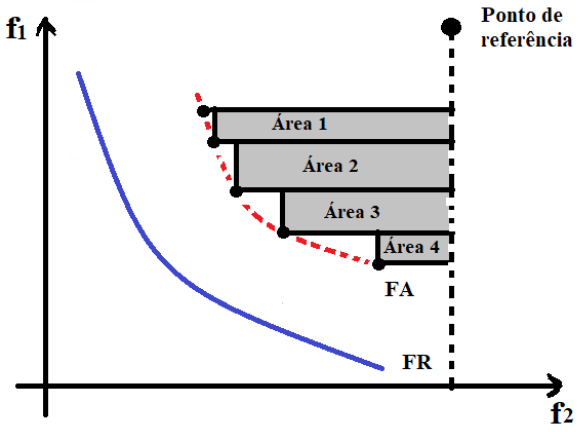

Figura 23. Indicador hipervolume

```
function [hipervolume] = indicador_hipervolume(fronteira,Pontos)

    N = numel(fronteira);
    vetorReferencia = 1.1*Pontos.antiutopico;
    hipervolume = 0;

    if N ~= 1
        fronteiraTempo = [];
        fronteira_aux = fronteira;

        for i = 1:N
            fronteiraTempo(i) = fronteira(i).fitness(1);
        end


        % Classificação baseado em f1
        [~, flCrescente] = sort(fronteiraTempo);
        fronteira = fronteira_aux(flCrescente);

        dominantes = NaN;
        %Pontos da fronteira não-dominada que dominam o vetor de referência
        for i = 1:N
            if all(fronteira(i).fitness <= vetorReferencia) && any(fronteira(i).fitness < vetorReferencia)
                dominantes(i) = fronteira(i);
            end
        end
        if isnan(dominantes) == false
            for i = 1:numel(dominantes)
                base = dominantes(i).fitness(1) - vetorReferencia(1);
                altura = dominantes(i).fitness(2) - vetorReferencia(2);
                hipervolume = hipervolume + base*altura;
            end
        end
    end
end
```

Figura 24. Cálculo do indicador hipervolume s-metric

VII. INDICADORES DE QUALIDADE

O processo de decisão em ambiente complexo dificulta a tomada de decisão, pois pode envolver dados imprecisos ou incompletos, múltiplos critérios e inúmeros agentes de decisão. Além disso, os problemas de decisão podem também ter vários objetivos, que acabam sendo conflitantes entre si. Sendo assim, a tomada de decisão deve buscar uma opção que apresente o melhor desempenho, a melhor avaliação, ou o melhor acordo entre as expectativas do decisor, considerando a relação entre os elementos. Podemos então, definir a decisão como um processo de análise e escolha entre várias alternativas disponíveis do curso de ação que a pessoa deverá seguir [13].

Para utilização dos indicadores, além das duas funções objetivo, um terceiro critério foi criado: dificuldade do percurso entre as cidades.

A dificuldade no percurso é uma matriz de 250 x 250 gerada usando números aleatórios inteiros de um a 10. Quanto maior este número, maior a dificuldade do percurso para ir da cidade i à cidade j.

A. AHP

O método AHP se baseia em três etapas [13]:

1. Construção de hierarquia: no método AHP o problema é estruturado em níveis hierárquicos, o que facilita a melhor compreensão e avaliação do mesmo (Figura 20).
2. Definição de prioridades: fundamenta-se na habilidade do ser humano de perceber o relacionamento entre objetos e situações observadas, comparando pares, à luz de um determinado foco, critério ou julgamentos paritários. Nessa etapa, é usada a escala numérica de Saaty para realizar a comparação (Figura 21).
3. Consistência lógica: o ser humano tem a habilidade de estabelecer relações entre objetos ou ideias de forma que elas sejam coerentes, tal que estas se relacionem bem entre sie suas relações apresentem consistência. Assim o método AHP se propõe a calcular a Razão de Consistência dos julgamentos, denotada por RC = IC/IR, onde IR é o Índice de Consistência Randômico obtido para uma matriz recíproca de ordem n, com elementos não-negativos e gerada randomicamente.

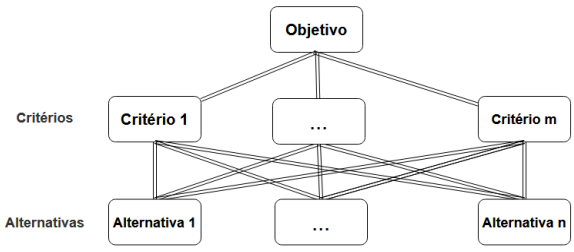

Figura 25. Hierarquização do problema

Escala numérica	Escala Verbal	Explicação
1	Ambos elemento são de igual importância.	Ambos elementos contribuem com a propriedade de igual forma.
3	Moderada importância de um elemento sobre o outro.	A experiência e a opinião favorecem um elemento sobre o outro.
5	Forte importância de um elemento sobre o outro.	Um elemento é fortemente favorecido.
7	Importância muito forte de um elemento sobre o outro.	Um elemento é muito fortemente favorecido sobre o outro.
9	Extrema importância de um elemento sobre o outro.	Um elemento é favorecido pelo menos com uma ordem de magnitude de diferença.
2, 4, 6, 8	Valores intermediários entre as opiniões adjacentes.	Usados como valores de consenso entre as opiniões.
Incremento 0.1	Valores intermediários na graduação mais fina de 0.1.	Usados para graduações mais finas das opiniões.

Figura 26. Escala numérica de Saaty

Fonte: MARTINS, SOUZA, BARROS, 2009

B. PROMETHEE II

O cálculo do PROMETHEE II se divide em quatro momentos distintos [14]:

1. O primeiro passo consiste em calcular a diferença de desempenho da alternativa x_i com a alternativa x_k relativa a critério j e a função de preferência relativa de cada critério j, que será escolhida pelo decisor e é dada pela equação:

$$P(x_i, x_k) = P_j(u_j(x_i) - u_j(x_k)) = P_j(\delta_{ik}) \quad (12)$$

2. Calcular o índice de preferência da alternativa x_i comparada a todas as demais alternativas x_k. Este índice é dado pela equação:

$$S_{ik} = \frac{\sum_j w_j P(\delta_{ik})}{\sum_j w_j} \quad (13)$$

3. Cálculo do fluxo positivo, que representa a intensidade de preferência de uma alternativa sobre todas as demais alternativas, e do fluxo negativo, representa a intensidade de preferência de todas as alternativas sobre uma das alternativas.

$$\Phi_i^+ = \sum_k S_{ik} \quad (14)$$

$$\Phi_i^- = \sum_k S_{ki} \quad (15)$$

4. Ordenação geral das alternativas, utilizando o fluxo de superação neto ou fluxo líquido, que representa o balanço entre o poder e a fraqueza da alternativa. Cabe ressaltar que quanto maior o fluxo líquido melhor é o desempenho da alternativa, a equação para este fluxo é dada por:

$$\Phi(a) = \Phi^+(a) - \Phi^-(a) \quad (16)$$

```
function [Q] = promethee(fronteira,w)

    n = numel(fronteira);
    m = 3;
    tempo = zeros(n,1);
    distancia = zeros(n,1);
    dificuldade = zeros(n,1);
    for i=1:n
        tempo(i) = fronteira(i).fitness(1);
        distancia(i) = fronteira(i).fitness(2);
        dificuldade(i) = fronteira(i).dificuldade;
    end
    D = [tempo distancia dificuldade];
    Vl=zeros(m,n,n);
    V=zeros(n,n);
    Fplus=zeros(n,1);
    Fminus=zeros(1,n);
    for j=1:m
        for i=1:n
            for s=1:n
                d=D(i,j)-D(s,j);
                H=0;
                if d>0
                    H=1;
                end
                Vl(j,i,s)=w(j)*H;
            end
        end
    end
    V(:,:)=sum(Vl,1);
    Fplus =sum(V,2)-diag(V);
    Fminus=sum(V,1)-diag(V)';
    Q=Fplus-Fminus';
end
```

Figura 27. Código para o tomador de decisão Promethee II

VIII. RESULTADOS E DISCUSSÕES

Para usar os indicadores, foi necessário encontrar o ponto antiutópico e as extremidades da fronteira Pareto Ótima estimada.

Foram realizadas cinco simulações do algoritmo com os mesmos parâmetros que foram usados posteriormente para obtenção dos resultados, conforme observado na Tabela 1. O ponto antiutópico (PA) foi obtido pegando-se os piores valores de f_1 e f_2 do melhor conjunto de resultados dentre as simulações. Esse conjunto foi obtido na simulação 5.

$$PA = (103.1, 6361.4) \quad (17)$$

As extremidades da fronteira Pareto ótima estimada (FR) foi obtida pegando-se os melhores valores da extremidade da fronteira Pareto estimada dentre cinco execuções do algoritmo (simulação 5).

Para este trabalho o ponto de referência considerado no indicador hipervolume é igual a 110% do ponto antiutópico.

Tabela 1. Extremidade da fronteira Pareto estimada

Execução	Extremidade esquerda	Extremidade direita
1	(104.9, 6958.2)	(110.0, 6588.8)
2	(101.4, 7113.7)	(109.4, 6502.8)
3	(106.6, 7220.6)	(118.4, 6570.0)
4	(107.9, 6805.5)	(116.5, 6317.1)
5	(97.5, 6361.4)	(103.1, 6026.4)

A Figura 27 mostra as fronteiras não dominadas ao longo das mil gerações.

Percebe-se um afunilamento da distribuição dos membros desta fronteira, indicando que o algoritmo não teve um bom espalhamento. Isso pode ser visto pelo alto valor do indicador Δ médio, que ficou bem próximo a um (0.99858).

Essa característica da convergência do algoritmo refletiu no indicador hipervolume. As fronteiras não dominadas e, consequentemente, a fronteira Pareto Ótima estimada não tiveram uma boa distribuição espacial (espalhamento). Isso fez com que o ponto utópico e antiutópico ficassem muito próximo. Como esse indicador só é calculado para valores que dominam o vetor de referência (calculado usando o ponto antiutópico) demorou para que o indicador saísse do valor zero.

Podemos notar que após isso, o indicador hipervolume atingiu altos valores, tendo uma média de 16131.05. Isso se deve a três fatores: os altos valores de distância, todos os indivíduos da população terem convergido para a fronteira Pareto estimada, e pela fronteira Pareto estimada ter ficado próxima a fronteira Pareto Ótima estimada. Isso pode ser deduzido de forma lógica, já que o ponto utópico e antiutópico ficaram muito próximos, ou seja, qualquer valor entre eles ficará próximo à fronteira Pareto Ótima. Além disso, os crescentes valores deste indicador mostram que a fronteira não dominada aproximou-se cada vez mais da fronteira Pareto Ótima.

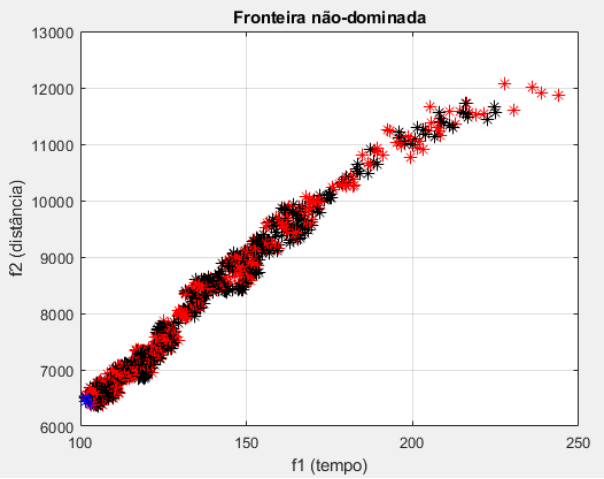

Figura 28: Fronteiras não dominadas (vermelho e preto) e fronteira Pareto Estimada (azul)

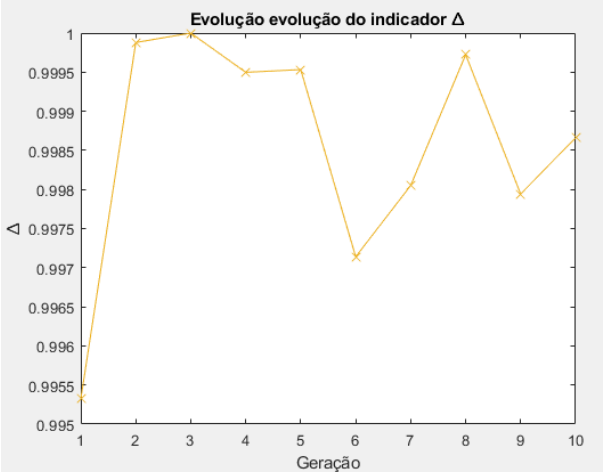

Figura 29: Evolução do indicador Δ

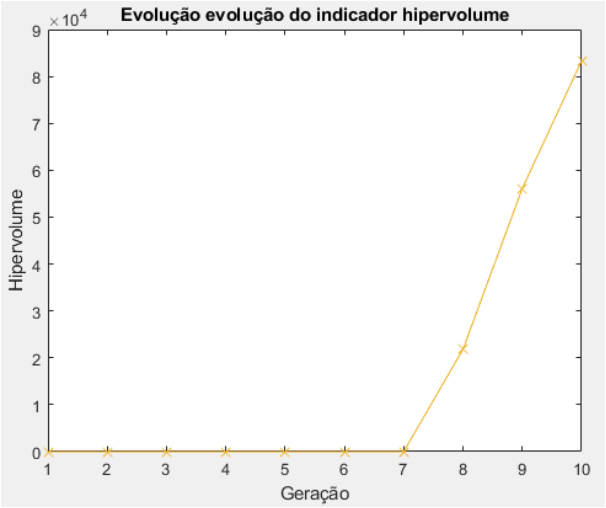

Figura 30: Evolução do indicador hipervolume

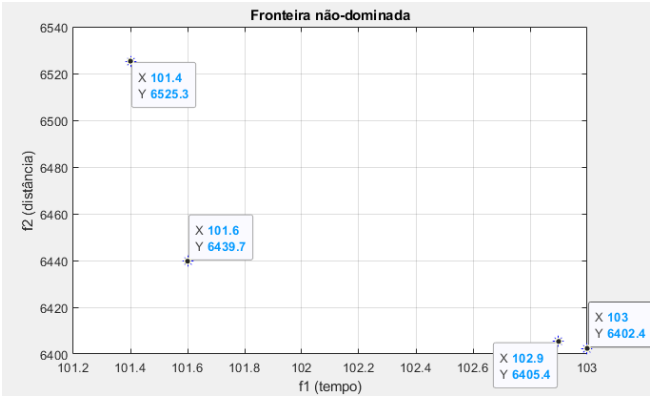

Figura 31: Fronteira Pareto estimada

O vetor de pesos usado no Promethee foi obtido usando o método AHP.

A matriz apresentada na Tabela 2 foi obtida usando a escala numérica de Saaty. Nela f_1 é o critério relacionado ao tempo, f_2 relacionado à distância e f_3 o critério dificuldade.

Neste trabalho, consideramos que o PVC está relacionado a uma prova de orientação. Nesse esporte, cada competidor recebe um mapa com pontos onde devem passar. Vence o competidor que passar por todos os pontos mais rapidamente.

A partir da tabela 2, o vetor de pesos (w) é calculado normalizando o autovetor associado ao maior autovalor da matriz. Sendo assim, obtêm-se um vetor de pesos igual a:

$$w = \begin{bmatrix} 0.7306 \\ 0.0810 \\ 0.1884 \end{bmatrix} \quad (18)$$

Tabela 2. Relação par a par empregando a escala Saaty

	f_1	f_2	f_3
f_1	1	7	5

f_2	$\frac{1}{7}$	1	$\frac{1}{3}$
f_3	$\frac{1}{5}$	3	1

Para sabermos se houve consistência na escolha dos valores da matriz, calculamos o quociente de consistência (QC).

$$QC = \frac{\frac{\lambda_{max} - n}{n-1}}{0.52} \tag{19}$$

Onde n é o número de linhas da matriz, λ_{max} é o valor do maior autovalor da matriz e 0.52 é o índice de consistência aleatório para uma matriz de ordem 3 encontrado por Saaty.

Encontrou-se um valor de QC igual a 0.0642. O valor próximo a zero obtido mostra que houve consistência na escolha dos valores da matriz.

Usando os valores de w, a melhor solução dentro da fronteira Pareto Estimada para o nosso problema foi o vetor

$$solução = \begin{bmatrix} 103 \\ 6402.4 \\ 5.6840 \end{bmatrix} \tag{20}$$

REFERÊNCIAS

[1] FARIAS, M. S. R. *Algoritmos Evolucionários Aplicados ao Problema do Caixeiro Viajante Multiobjetivo.* Dissertação apresentada ao Programa de Pós-Graduação em Modelagem Computacional de Conhecimento da Universidade Federal de Alagoas. Maceió, 2008.

[2] PRESTES, A. N. *Uma Análise Experimental de Abordagens Heurísticas Aplicadas ao Problema do Caixeiro Viajante*. Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Sistemas e Computação da Universidade Federal do Rio Grande do Norte. Natal, 2006.

[3] MORAES, D. H. *Uma Nova Abordagem Baseada em Algoritmos Evolutivos Multiobjetivo Aplicado ao Problema do Caixeiro Viajante Biobjetivo*. Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2017.

[4] MATAI, R.; SINGH, S.P.; MITTAL, M. L. *Traveling Salesman Problem; An Overview of Applications, Formulations and Solution Approaches.* Traveling Salesman Problem, Theory and Applications. Intechopen, p. 1-24, 2010.

[5] RODRIGUES, M. A. P. *Problema do Caixeiro Viajante: um Algoritmo para a Resolução de Problemas de Grande Porte Baseado em Busca Local Dirigida*. Dissertação apresentada à Universidade Federal de Santa Catarina, Florianópolis. Santa Catarina, 2000.

[6] SILVA, T. G. N. *Proposta de Avaliação de um Método para Gerar a População Inicial de Algoritmos Genéticos Multiobjetivo*. Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual do Ceará. Fortaleza, 2015.

[7] TAKAHASHI, F. C. *Estudo sobre Métodos Evolutivos Multiobjetivos Voltados para a Robustez e Diversidade no Espaço de Decisão*. Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais. Belo Horizonte, 2015.

[8] FILHO, H. L. J. *Incorporando Preferências do Usuário um uma Abordagem de Teste de Linha de Produtos de Software Baseada em Otimização Multiobjetivo*. Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Paraná. Curitiba, 2018.

[9] DIAS, T. F. *Otimização Multiobjetivo de uma Máquina Pentafásica Utilizando NSGA-II*. Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Rio Grande do Sul. Porto Alegre, 2016.

[10] VARGAS, D. E. C. *Um Estudo dos Parâmetros do Algoritmo NSGA-II com o Operador SBX em Problemas de Otimização Estrutural Multiobjetivo*. Proceeding Series of the Brazilian Society of Computional and Applied Mathematics, v. 6, n. 2, 2008.

[11] SILVA, J. B. A.; SANTOS, T. F.; XAVIER, S. M. *Estado da Arte de Medidas de Desempenho de Algoritmos de Otimização Multiobjetivo*. Revista de Matemática de Ouro Preto, v. 1, pp 66-100, 2017.

[12] CAMARGO, F. H. F. *Aplicação de Meta Heurística na Otimização Multiobjetivo de Sistemas Hidrotérmicos*. Dissertação apresentada ao Programa de Pós-Graduação da Universidade Federal de Goiás. Goiânia, 2017.

[13] MARINS, C. S.; SOUZA, D. O.; BARROS, M. S. *O uso de Métodos de Análise Hierárquica (AHP) na Tomada de Decisões Gerenciais – Um Estudo de Casos*. XLI SBPO, Pesquisa Operacional na Gestão do Conhecimento, p. 1778-1788, 2009.

[14] CENSA, E. P. P. S.; COSTA, H. G.; HORA, H. R. M. *O Método PROMETHEE II como ferramenta para Ordenação de Municípios: O Estudo de Casos do Rio de Janeiro*. Proceedings 7º Congresso Luso-Moçambicano de Engenharia, p. 18-18, 2014.