Avaliação experimental de algoritmos evolutivos para otimização global

RESUMO

Otimização global é um ramo da matemática aplicada que tem como objetivo encontrar o valor ótimo de uma ou mais funções objetivo sob um determinado conjunto de restrições. Diversos problemas em engenharia, biologia, química e ciência da computação, dentre outras áreas, são modelados como problemas de otimização global. As meta-heurísticas destacam-se dentre algoritmos para otimização global, sendo métodos genéricos e eficientes que podem ser aplicados para resolver uma grande variedade de problemas. Este trabalho estuda e avalia cinco das principais meta-heurísticas da literatura: algoritmo genético, algoritmo da colônia de formigas, algoritmo de seleção clonal, algoritmo de evolução diferencial e algoritmo de enxame de partículas. Como casos de teste, utilizamos um conjunto de funções matemáticas diverso tradicionalmente utilizadas como *benchmark* para a avaliação destes algoritmos. Os resultados obtidos a partir de uma rigorosa e robusta avaliação experimental apontam a melhor destas meta-heurísticas avaliadas quando resolvendo o conjunto de *benchmark* proposto.

PALAVRAS CHAVE. Otimização global, Meta-heurísticas, Avaliação experimental.

Tópicos: Meta-heurísticas, Inteligência Computacional.

ABSTRACT

Global optimization is a branch of applied mathematics with the objective of finding the optimum value of one or more objective functions over a determined set of constraints. Several problems in engineering, biology, chemistry, and computer science, among many other areas, are modeled as global optimization problems. Small- or medium-sized global optimization problems can be solved using mathematical programming techniques. Meta-heuristics stand-out among the algorithms for global optimization, being generic and efficient methods that can be applied to solve a large variety of optimization problems. This work studies and evaluates five of the main meta-heuristics in the literature: the genetic algorithm, the ant colony optimization, the clonal selection algorithm, the differential evolution algorithm, and the particle swarm optimization. As test cases, we employed a diverse set of mathematical functions traditionally employed as a benchmark for evaluating this class of algorithms. The results obtained from a rigorous and robust experimental evaluation point out the best among of the evaluated meta-heuristics in solving the proposed benchmark set.

KEYWORDS. Global optimization, Meta-heuristics, Experimental evaluation

Paper topics: Meta-heuristics, Computational intelligence

1. Introdução

Problemas de otimização global consistem em encontrar o máximo (ou mínimo) de uma função objetivo sob um determinado conjunto de restrições. Podemos descrever um problema de otimização global como

$$\text{maximizar ou minimizar } f(x)$$
$$\text{tal que } x \in S,$$

onde $S \in \mathbb{R}^n$ é o conjunto de soluções viáveis do problema, x é uma solução e $f(x)$ é uma função objetivo que deve ser otimizada [Horst et al., 2000].

A otimização global de funções matemáticas é uma tarefa desafiadora porque elas possuem múltiplos ótimos locais. Uma função matemática muito conhecida é a Rastringin, que pode ser definida como

$$f(x) = \sum_{i=0}^{d} \left(x_i^2 - 10\cos 2\pi x_i\right),$$

onde d é o número de dimensões (ou variáveis) da função objetivo. A Figura 1 mostra esta função para $d = 2$. Pode-se notar a enorme quantidade de mínimos locais, o que dificulta sua otimização.

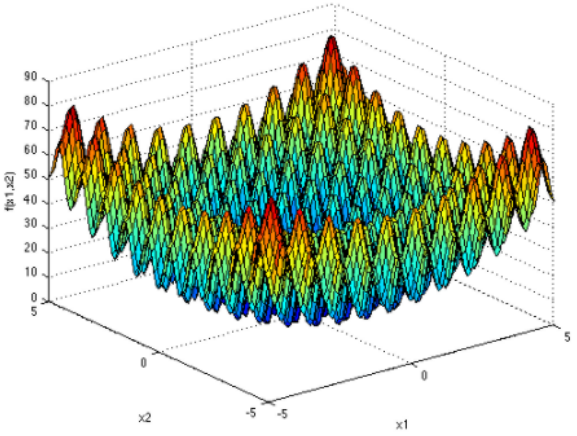

Figura 1: A função Rastringin em duas dimensões

Existem várias abordagens para a otimização global de funções, incluindo métodos determinísticos e estocásticos. Os métodos determinísticos geralmente exploram de forma sistemática o espaço de busca, tentando encontrar o ótimo global através de algoritmos de ramificação e redução, ou através de métodos baseados em derivadas [Horst et al., 2000; Pardalos e Romeijn, 2013]. Já os métodos estocásticos utilizam técnicas baseadas em amostragem aleatória para explorar o espaço de busca de forma probabilística, como a busca aleatória, o recozimento simulado, algoritmos de busca local e algoritmos evolutivos [Zhigljavsky e Zilinskas, 2007].

Neste trabalho estamos interessados em estudar e comparar métodos estocásticos para otimização global. Mais especificamente, este trabalho estuda cinco diferentes algoritmos evolutivos: o algoritmo genético (GA, do inglês *Genetic Algorithm*) [Reeves, 2010], o algoritmo de colônia de formigas (ACO, do inglês *Ant Colony Optimization*) [Socha e Dorigo, 2008], o algoritmo de seleção clonal (CLONAL) [De Castro e Von Zuben, 2000], o algoritmo de evolução diferencial (DE, do inglês *Differential Evolution*) [Storn e Price, 1997] e o algoritmo de enxame de partículas (PSO, do inglês *Particle Swarm Optimization)* [Kennedy e Eberhart, 1995]. Estes algoritmos serão testados

utilizando 15 diferentes funções matemáticas desenvolvidas como *benchmark* em uma competição de otimização global [Liang et al., 2014].

2. Algoritmos

Nesta seção, apresentamos os cinco algoritmos utilizados em nossos experimentos computacionais. Estes cinco métodos foram escolhidos pois cada um deles representa uma diferente classe de algoritmo evolutivo. Apesar disso, todos os métodos utilizam o conceito de uma população de soluções X, sendo que cada solução $X_i \in X$ é representada como um vetor d-dimensional, de tal forma que cada posição do vetor (denominada gene) representa o valor de uma das variáveis da função objetivo a ser otimizada. A Figura 2 mostra para um problema arbitrário, uma representação de um indivíduo com 5 variáveis. Nesta solução, nós denotamos cada gene por x_i, sendo $0 < i \leq d$. Ainda denota-se por *fitness* o valor $f(X_i)$, isto é, o valor da solução X_i quando aplicado como solução para uma função f.

2.1	2.8	0.5	1.3	2.3	1.7

Figura 2: Exemplo de um indivíduo de 5 genes

Além disso, a população inicial de todos os cinco algoritmos são inicializadas exatamente da mesma maneira. Cada gene é inicializado utilizando uma distribuição uniforme contínua $\mathcal{U}(-100, 100)$. Logo após esta inicialização, é dado início ao processo de evolução. Os processos evolutivos dos cinco algoritmos propostos são descritos a seguir.

2.1. Algoritmo Genético

Inicialmente, a população do GA é inicializada como descrito na Seção 2. Logo após, é dado início ao processo evolutivo do algoritmo. O processo evolutivo é iterativo e constitui-se nos operadores de seleção, cruzamento, mutação e evolução, sendo que cada iteração do processo evolutivo é considerado uma geração.

A seleção é realizada utilizando um algoritmo de roleta como descrito em Reeves [2010]. Já o operador de cruzamento escolhido é o *2-point crossover*, onde um novo indivíduo c (denominado filho) é gerado a partir de dois pais $a \in X$ e $b \in X$ previamente selecionados. O indivíduo c é gerado de tal forma que

$$c_i = \begin{cases} a_i, & \text{se} \quad i \leq k \\ b_i, & \text{se} \quad r > k \end{cases}\}$$

onde k é um número pseudo-aleatório gerado no intervalo $[0, d]$.

O operador de mutação utilizado altera o valor de um único gene do indivíduo c gerado de forma aleatória. O gene a ser mutado é escolhido utilizando uma distribuição uniforme inteira $U(1, d)$ e seu novo valor é gerado da mesma forma que na inicialização da população, isto é, utilizando uma distribuição uniforme contínua $\mathcal{U}(-100, 100)$.

Os operadores de seleção, cruzamento e mutação são aplicados $|X|$ vezes, gerando assim um número de filhos igual ao número de indivíduos anteriormente na população. Seja $C = \{C_1, C_2, \ldots, C_{|X|}\}$ o vetor contendo todos os filhos gerados. O operador de evolução é aplicado de tal forma que o vetor C é comparado com o vetor X da população atual e pode ser descrito como

$$X_i = \begin{cases} X_i, & \text{se} \quad f(X_i) < f(C_i) \\ C_i, & \text{caso contrário} \end{cases}.$$

2.2. Evolução Diferencial

A primeira etapa do DE é realizar a inicialização da população, assim como descrito na Seção 2. Após isto, tem-se início o processo evolutivo iterativo que ocorre de forma semelhante ao do GA. A única modificação é que a ordem de aplicação dos operadores de mutação e cruzamento no DE são invertidos, isto é, primeiro realiza-se o processo de mutação para depois aplicar o cruzamento no indivíduo previamente mutado. Os operadores utilizados neste algoritmo são descritos a seguir.

A mutação produz novos indivíduos, denominados indivíduos experimentais, a partir da adição de um indivíduo aleatório γ à diferença entre dois outros indivíduos aleatórios α e β da mesma população. Este processo utiliza um fator de perturbação $F \in [0; 2]$. Seja X_α, X_β, e X_γ três indivíduos distintos que pertencem à população X selecionados aleatoriamente utilizando o mesmo algoritmo de roleta previamente utilizado no GA. Um indivíduo experimental V é gerado como como

$$V = X_\gamma + F(X_\alpha - X_\beta)$$

O operador de cruzamento é aplicado é o *2-point crossover*, também utilizado no GA. Ele é aplicado utilizando o indivíduo V gerado pelo operador de mutação e um outro indivíduo $X_i \in X$, onde $i \neq \alpha \neq \beta \neq \gamma$. Já o operador de evolução aplicado ao DE também é desenvolvido exatamente igual ao do GA.

2.3. Colônia de Formigas

O ACO é um algoritmo baseado no comportamento de uma colônia de formigas real, onde os membros da colônia na busca por alimento depositam feromônio pelo trajeto, marcando assim as melhores rotas encontradas.Esse algoritmo é amplamente utilizado para solucionar problemas complexos em grafos, como o problema do caixeiro-viajante.

Para adaptar o ACO que normalmente é utilizado para solucionar problemas de otimização combinatória discretos afim de solucionar um problema contínuo, foi utilizada uma abordagem com base na distribuição gaussiana para simularmos a distribuição do feromônio.

2.3.1. Atualização do feromônio

Seja $\tau(x_i)$ o feromônio calculado para a formiga X_i na dimensão x_i, a obtenção do valor atual do feromônio naquela região será dada pela função de distribuição gaussiana, como demonstrado na Equação (1) [Socha e Dorigo, 2008]:

$$\tau(x_i) = \frac{1}{\sqrt{2\pi\sigma_i}} \exp\left(-\frac{(x_i^{\text{opt}} - x_i)^2}{2\sigma_i^2}\right) \tag{1}$$

Nesta equação, x_i^{opt} representa a posição da melhor formiga na dimensão $_i$, e σ_i é o desvio padrão calculado na dimensão $_i$ em relação a x_i^{opt}.

2.3.2. Transição de estado das formigas

Após o calculo do feromônio deve-se gerar um conjunto de n pontos candidatos $X^{\text{new}} = \{X_1^{\text{new}}, X_2^{\text{new}}, \ldots, X_n^{\text{new}}\}$ para a nova posição da formiga. Em sequência, é calculada a probabilidade de escolha de cada ponto como descrito na equação (2) [Socha e Dorigo, 2008]:

$$P(X_j^{\text{new}}) = \frac{\tau(X_j^{\text{new}})}{\sum_{j=1}^{m} \tau(X_j^{\text{new}})} \tag{2}$$

Utilizando a probabilidade calculada acima, é sorteado um dos candidatos que se tornará a nova posição da formiga.

2.3.3. Pseudo código

Inicializamos nossa colônia de formigas aleatoriamente, atualizamos as trilhas de feromônio e a variável que irá armazenar a melhor solução do algoritmo até o momento atual de execução (linhas 1 a 3). Sendo assim, até que não ocorra a convergência dos resultados para um ótimo local, ou até que o tempo limite estabelecido seja atingido (linha 4), selecionamos uma nova posição para cada formiga gerando n pontos candidatos e sorteando um deles com base na probabilidade calculada com a fórmula (2)(linha 5).Em seguida calculamos o valor da função para cada formiga (linha 6), atualizamos o valor do feromônio para cada dimensão de cada formiga com a Equação (1) (linha 7), evaporamos o feromônio de cada localização com base em uma taxa pré-estabelecida (linha 8), calculamos o desvio padrão σ_i para cada dimensão i do problema em relação a melhor localização conhecida (linha 9), depois garantimos que a melhor localização encontrada durante toda a execução esteja no conjunto solução (linha 10). Por fim, é retornado o melhor resultado encontrado (linha 12).

Algorithm 1: Algoritmo de Colônia de Formigas

Entrada : Parâmetros do ACO, como o número de formigas, a taxa de evaporação do feromônio, a função objetivo, etc.
Saída : Melhor solução encontrada pelo ACO

1 Inicialize a colônia de formigas aleatoriamente;
2 Inicialize as trilhas de feromônio com valores iniciais;
3 Inicialize a melhor solução global e a melhor solução pessoal de cada formiga;
4 **while** *não houver convergência entre os resultados* ***and*** *não atingir tempo limite* **do**
5 Selecione uma nova posição para a formiga;
6 Calcule o valor da função objetivo para a posição atual da formiga;
7 Atualize as trilhas de feromônio;
8 Evapore o feromônio;
9 Atualize os valores de sigma (desvio padrão) para as dimensões do problema;
10 Capture a melhor formiga e garanta que a mesma esteja na colônia
11 **end**
12 **return** Melhor solução global encontrada;

2.4. Algoritmo de Enxame de Partículas

O PSO é uma meta-heurística e método de otimização estocástica baseado em enxames. O PSO propõe simular o comportamento social de animais, como insetos, passarinhos e peixes. A ideia aqui é fazer com que esses enxames de animais trabalhem por meio de um cooperativismo com o objetivo de encontrar comida, onde cada membro (partícula) desse enxame tem um comportamento de busca de acordo com sua aprendizagem baseada em experiências suas e de outros membros [Wang et al., 2018].

2.4.1. Cálculo da velocidade da partícula

Seja v_{ij} a velocidade da partícula na dimensão i, o cálculo da velocidade da partícula para o momento atual será realizado como

$$v_{ij} = v_{ij} \cdot w + c_1 \cdot r_1 \cdot (p_{best_{ij}} - x_{ij}) + c_2 \cdot r_2 \cdot (g_{best_j} - x_{ij}), \tag{3}$$

sendo x_{ij} a posição atual da partícula i na dimensão j, $r1$ e $r2$ números aleatórios entre 0 e 1, $p_{best_{ij}}$ a melhor posição pessoal da partícula i na dimensão j, g_{best_j} a melhor posição global na dimensão j, w o peso de inércia, responsável por determinar o quão conservativa ou exploratória a busca será, c_1

o fator de aprendizagem cognitiva, que representa a tendência da partícula em se mover em direção à sua própria melhor posição anterior e c_2 o fator de aprendizagem social, responsável por controlar a influencia do g_{best_j} na atualização da velocidade.

2.4.2. Pseudo código

O pseudo-código 2 exemplifica nossa implementação do algoritmo PSO. Neste algoritmo, inicializamos nossa população de partículas aleatoriamente e a variável que irá deter a melhor solução do algoritmo até o momento atual de execução (linhas 1 e 2). Assim, até que não ocorra a convergência dos resultados para um ótimo local, ou até que o tempo limite estabelecido seja atingido (linha 3), para cada partícula do enxame (linha 4), realizamos o calculo do valor atual da partícula (linha 5). Em sequência, verificamos se o novo valor da partícula é melhor que o melhor valor da mesma partícula até o momento (linha 6) e atualizamos ou mantemos o melhor valor (linha 7). Em seguida, verificamos se o valor calculado para a partícula é melhor que o melhor valor global (linha 9). Caso positivo, atualizamos o melhor valor global (linha 10). Por fim, utilizamos a Equação (3) para atualizar a velocidade das partícula (linha 12) e, com estas velocidades atualizadas, calculamos a nova posição de cada partícula (linha 13). Por fim, a melhor solução encontrada é retornada (linha 16).

Algorithm 2: Algoritmo de Enxame de Partículas

Entrada : Parâmetros do PSO, como o número de partículas, a função objetivo, etc.
Saída : Melhor solução encontrada pelo PSO

```
1  Inicialize o enxame de partículas aleatoriamente;
2  Inicialize a melhor solução global e a melhor solução pessoal de cada partícula;
3  while não houver convergência entre os resultados and não atingir tempo limite do
4      for cada partícula no enxame do
5          Calcule o valor da função objetivo para a posição atual da partícula;
6          if a função objetivo é melhor que a melhor solução pessoal da partícula then
7              Atualize a melhor solução pessoal da partícula;
8          end
9          if a função objetivo é melhor que a melhor solução global then
10             Atualize a melhor solução global;
11         end
12         Atualize a velocidade da partícula de acordo com as equações do PSO;
13         Atualize a posição da partícula de acordo com a velocidade;
14     end
15 end
16 return Melhor solução global encontrada;
```

2.5. Algoritmo de Seleção Clonal

O Algoritmo de Seleção Clonal (CLONAL) é uma analogia algorítmica que se baseia no principio de seleção presente no sistema imunológico biológico [De Castro e Von Zuben, 2000]. A ideia principal é gerar uma população principal P_m com n anticorpos, e a partir dos anticorpos a_i, onde $a_i \in P_m$, gerar varias população de clones C_i que serão alvo de mutações.

2.5.1. Inicialização

Como o algoritmo está sendo utilizando para otimização de funções f com a assinatura $f : R^n \rightarrow R$, cada anticorpo a_i será representado por um vetor X d-dimensional, logo será atrelado

a este anticorpo também uma avaliação de afinidade, sendo nada mais do que a função f aplicada ao vetor a_i. É importante ressaltar também que $\forall x \in X, a \leq x \leq b$, onde a e b representam uma restrição de valores para o espaço de busca.

2.5.2. Mutação

Para cada anticorpo a_i uma nova população C_i com k clones é gerada, após termos todas as populações de clones construídas, é aplicado uma função de mutação $M^*(C, \alpha)$ que irá gerar as populações de clones alvo de mutações C_i^*. Logo, podemos afirmar que:

$$C_i^* = M^*(C_i, \alpha_i) \tag{4}$$

A função de mutação $M^*(C, \alpha)$ é definida da seguinte forma

$$M^*(C, \alpha) = \{M^{(\alpha)}(X) \text{ tal que } X \in C\} \tag{5}$$

onde $M(X)$ é a mesma Função **??** de mutação utilizada pelo algoritmo genético e α representa o número de vezes em que a função de mutação singular M deve ser aplicada ao anticorpo representado por X. Sabendo que $0 < \alpha \leq n$, uma boa maneira de definir o valor de α a ser utilizado para cada população de clones, é de acordo com a afinidade do anticorpo que os gerou.

Neste caso, utilizamos a função logarítmica, descrita na Equação (6), que detém essa característica

$$\alpha = l(x) = a + (b - a) \cdot \frac{log(x) - log(c)}{log(d) - log(c)} \tag{6}$$

Nesta equação, o valor de entrada x representa o *rank* do anticorpo na população P_m, definido por meio de ordenação. As constantes a, b, c e d são valores que descrevem o intervalo de domínio e contradomínio da função logarítmica

$$l : [c, d] \rightarrow [a, b].$$

Em nossa implementação, estas contantes assumirão os valores $c = 1$, $d = |P_m|$, $a = 1$ e $b = n$.

2.5.3. Seleção

Neste algoritmo, a decisão de quais anticorpos a_i devem seguir em P_m acontece extraindo o melhor clone c_{best} após as mutações da população C_i^*, se c_{best} for melhor que a_i quanto sua avaliação realizada com base em uma função objetivo, então a_i é substituído por c_{best}. O procedimento descrito ocorre para todos anticorpos a_i que tiveram uma população de clones C_i gerada, e então, após o ranqueamento da nova população P_m gerada realizado com base em ordenação, prosseguimos para a próxima iteração.

2.5.4. Pseudocódigo

O pseudo-código 3 refere-se a nossa implementação do algoritmo CLONAL. Primeiramente, inicializamos uma população de anticorpos aleatórios (linha 1), além de uma variável que irá deter a melhor solução encontrada no decorrer do algoritmo que será retornada no fim da rotina (linha 2). Após as inicializações pertinentes, entramos em um *loop* que irá possibilitar que os anticorpos evoluem a cada nova iteração possível limitado pelos critérios de paradas (linha 3). Deste modo, para garantir evolução a cada iteração, uma sequência de procedimentos é definida. Primeiramente, populações de clones são geradas para cada anticorpo a_i, isso pode ser feito para todos os anticorpos ou através de alguma lógica de seleção (linha 5), após a definição das populações de clones, a função de mutação é aplicada a cada uma delas e logo em seguida é realizado a extração do melhor anticorpo

c_best que este procedimento pode gerar (linha 7). Com o melhor anticorpo em evidência, é verificado se sua afinidade é melhor do que o anticorpo original alvo de clonagem, se for, c_best irá substituir a_i na população principal de anticorpos (linhas 8 à 10). Existe ainda uma segunda condicional, responsável por garantir que a melhor solução global seja salva durante a execução do algoritmo (linhas 11 à 13). O processo descrito até que será realizado para todos os anticorpos elegíveis dentro da população principal, de acordo com a lógica de seleção implementada, e ao fim, é realizado um ranqueamento da população para uso de α nas iterações seguintes.

Algorithm 3: Algoritmo de Seleção Clonal

Entrada : Parâmetros do CLONAL, como o número de anticorpos, número de clones, a função objetivo, etc.
Saída : Melhor solução encontrada pelo CLONAL

1 Inicialize os anticorpos com seus respectivos X aleatoriamente;
2 Inicialize a melhor solução global com o melhor anticorpo gerado aleatoriamente;
3 **while** *não houver convergência entre os resultados* ***and*** *não atingir tempo limite* **do**
4 **for** *cada anticorpo* a_i *na população* **do**
5 C = Gerar populações de k clones do anticorpo a_i;
6 $\alpha = Extrairrankde$ a_i na população principal de anticorpos;
7 $C^* = M(C, \alpha)$;
8 c_{best} = Extrair o melhor anticorpo em C^*;
9 **if** *a avaliação de* c_{best} *for melhor que* a_i **then**
10 Atualize a_i para ser igual à c_{best};
11 **end**
12 **if** c_{best} *é melhor que a melhor solução global* **then**
13 Atualize a melhor solução global;
14 **end**
15 **end**
16 Ordenar anticorpos com base na avaliação de cada um;
17 **end**
18 **return** Melhor solução global encontrada;

3. Experimentos computacionais

Os experimentos computacionais foram realizados em um único núcleo de um processador Intel Core i7-7700, com *clock* de 3.6 GHz e 16 GB de RAM. Todos os algoritmos foram implementados em Ansi-C e compilados utilizando o GNU `gcc 11.3.0`. Além disso, os testes foram realizados no sistema operacional Ubuntu 22.04.1.

Os algoritmos propostos foram avaliados utilizando 15 diferentes funções matemáticas desenvolvidas como *benchmark* em uma competição de otimização global. Uma descrição geral das funções de *benchmark* é apresentada na Tabela 1. A primeira coluna exibe o número da função, enquanto a segunda coluna mostra o seu nome. A terceira coluna define o número de dimensões utilizadas em cada função, isto é, o valor de d. Já a quarta e última coluna define o valor ótimo da função. Note que os experimentos foram realizados sempre com $d = \{10, 30\}$, sendo que quanto maior o valor de d, mais difícil é o problema de otimização. Uma descrição detalhada das funções pode ser encontrada no trabalho de Liang et al. [2014].

Tabela 1: Funções de teste de benchmark CEC 2015.

No.	Functions	Dim	f_{min}
F1	Rotated Bent Cigar Function	$10, 30$	100
F2	Rotated Discus Function	$10, 30$	200
F3	Shifted and Rotated Weierstrass Function	$10, 30$	300
F4	Shifted and Rotated Schwefel's Function	$10, 30$	400
F5	Shifted and Rotated Katsuura Function	$10, 30$	500
F6	Shifted and Rotated HappyCat Function	$10, 30$	600
F7	Shifted and Rotated HGBat Function	$10, 30$	700
F8	Shifted and Rotated Expanded	$10, 30$	800
F9	Shifted and Rotated Expanded Scaer's F6 Function	$10, 30$	900
F10	Hybrid Function 1 (N = 3) Rastrigin	$10, 30$	1000
F11	Hybrid Function 2 (N = 4) Rastrigin	$10, 30$	1100
F12	Hybrid Function 3 (N = 5)	$10, 30$	1200
F13	Composition Function 1 (N = 5)	$10, 30$	1300
F14	Composition Function 2 (N = 3)	$10, 30$	1400
F15	Composition Function 3 (N = 5)	$10, 30$	1500

3.1. Otimização de parâmetros

A otimização de parâmetros foi realizada nas funções matemáticas $F3, F5$ e $F12$. Estas três funções foram escolhidas aleatoriamente e removidas do teste seguinte como forma de evitar um *overfitting* dos algoritmos propostos na etapa de comparação. Esta etapa teve como principal precursor a ferramenta iRace [López-Ibáñez et al., 2016]. Ela foi construída utilizando o método *Iterated Race* disponibilizado pelos autores da ferramenta e implementado na linguagem R. Este experimento ainda considerou um tempo limite de 15 segundos como critério de parada dos algoritmos.

A Tabela 2 exibe o experimento de otimização de parâmetros. A primeira coluna mostra o nome do algoritmo, enquanto a segunda coluna exibe o nome do parâmetro otimizado. A terceira coluna mostra o intervalo de teste, isto é, o menor e maior valor considerados para cada um dos parâmetros. Já a quarta coluna define o resultado da otimização de parâmetros, isto é, o resultado obtido ao aplicarmos o *Iterated Race*.

3.2. Comparação dos algoritmos

A comparação dos algoritmos propostos foi realizada utilizando os parâmetros definidos na Seção 3.1. Assim como no experimento anterior, foi definido um tempo de 15 segundos como critério de parada dos algoritmos. Além disso, foram utilizadas um total de 12 funções matemáticas como *benchmark*, isto é, as 15 funções descritas na Tabela 1 com exceção das utilizadas na otimização de parâmetros.

Os resultados deste experimento são mostrados na Tabela 3. A primeira coluna exibe o número da função matemática. A segunda e terceira coluna exibe os resultados para o GA, sendo que a segunda coluna mostra o valor médio (ME) obtido por 20 diferentes execuções do GA com diferentes sementes para o gerador de números pseudo-aleatórios, enquanto a terceira coluna mostra o desvio padrão (SD) para este mesmo experimento. As demais colunas mostram os resultados para os outros algoritmos, respectivamente o DE, o PSO, o ACO e o CLONAL. Cada linha sumariza os resultados para $d = 10$ e $d = 30$. Além disso, o melhor algoritmo, em relação ao seu desempenho médio, é destacado em negrito em cada linha.

Pode-se notar na Tabela 3 que o DE obteve o melhor resultado para a função $F14$, enquanto o CLONAL obteve o melhor resultado para a função $F4$. Para todas as outras funções matemáticas,

Tabela 2: Ajuste de parâmetros dos algoritmos testados

	Parâmetro	Intervalo de teste	Melhor
GA	Tamanho da População	$[30, 1.00 \times 10^4] \in \mathbb{Z}$	35
	Taxa de Mutação	$[0, 100] \in \mathbb{Z}$	72
	Taxa de Cruzamento	$[0, 100] \in \mathbb{Z}$	63
DE	Tamanho da População	$[30, 1.00 \times 10^4] \in \mathbb{Z}$	218
	Taxa de Cruzamento	$[0, 100] \in \mathbb{Z}$	98
	Taxa de Mutação	$[0, 100] \in \mathbb{Z}$	34
	Fator de pertubação	$[0, 2] \in \mathbb{R}$	0.91205
PSO	Tamanho da população	$[30, 1.00 \times 10^4] \in \mathbb{Z}$	5598
	Componente social 1	$[0.1, 3] \in \mathbb{R}$	1.52300
	Componente social 2	$[0.1, 3] \in \mathbb{R}$	0.77623
ACO	Número de formigas	$[30, 1.00 \times 10^4] \in \mathbb{Z}$	5252
	Número de candidatos	$[3, 30] \in \mathbb{Z}$	9
	Taxa de evaporação	$[0, 1] \in \mathbb{R}$	0.31458
CLONAL	Tamanho da população	$[30, 1.00 \times 10^4] \in \mathbb{Z}$	124
	Número de clones	$[2, 100] \in \mathbb{Z}$	36

Tabela 3: Comparação dos algoritmos propostos

	GA		DE		PSO		ACO		CLONAL	
	ME	SD	ME	SD	ME	SD	ME	SD	ME	SD
F1	3.65×10^7	1.38×10^7	7.75×10^3	1.87×10^3	$\mathbf{9.31 \times 10^3}$	7.02×10^3	1.92×10^9	4.35×10^8	1.69×10^5	1.53×10^5
F2	2.18×10^3	4.99×10^2	1.46×10^3	5.78×10^2	$\mathbf{2.00 \times 10^2}$	1.75×10^{-2}	1.62×10^4	2.87×10^3	5.35×10^3	2.01×10^3
F4	9.83×10^2	1.09×10^2	9.88×10^2	3.17×10^2	6.54×10^2	1.18×10^2	1.57×10^3	8.60×10^1	$\mathbf{4.01 \times 10^2}$	3.66×10^{-1}
F6	6.00×10^2	3.84×10^{-2}	6.00×10^2	4.85×10^{-2}	$\mathbf{6.00 \times 10^2}$	4.90×10^{-2}	6.02×10^2	3.81×10^{-1}	6.00×10^2	1.55×10^{-1}
F7	7.00×10^2	6.32×10^{-2}	7.00×10^2	4.45×10^{-2}	$\mathbf{7.00 \times 10^2}$	3.71×10^{-2}	7.13×10^2	2.36	7.00×10^2	3.06×10^{-1}
F8	8.04×10^2	5.12×10^{-1}	8.02×10^2	6.35×10^{-1}	$\mathbf{8.01 \times 10^2}$	3.78×10^{-1}	1.16×10^3	2.17×10^2	8.04×10^2	1.36
F9	9.03×10^2	1.28×10^{-1}	9.03×10^2	2.14×10^{-1}	$\mathbf{9.02 \times 10^2}$	3.68×10^{-1}	9.03×10^2	1.46×10^{-1}	9.03×10^2	3.37×10^{-1}
F10	3.36×10^3	5.66×10^2	1.63×10^3	2.65×10^2	$\mathbf{1.59 \times 10^3}$	2.12×10^2	1.75×10^4	6.83×10^3	7.23×10^3	4.50×10^3
F11	1.10×10^3	2.62×10^{-1}	1.10×10^3	3.43×10^{-1}	$\mathbf{1.10 \times 10^3}$	8.32×10^{-1}	1.11×10^3	5.66×10^{-1}	1.10×10^3	9.76×10^{-1}
F13	1.62×10^3	5.44×10^{-1}	1.62×10^3	1.16	$\mathbf{1.62 \times 10^3}$	3.49×10^{-1}	1.65×10^3	1.07×10^1	1.62×10^3	7.54×10^{-1}
F14	1.59×10^3	2.21	$\mathbf{1.59 \times 10^3}$	2.37	1.59×10^3	5.15	1.60×10^3	1.87	1.59×10^3	3.85
F15	1.92×10^3	3.31	1.91×10^3	4.33×10^{-1}	$\mathbf{1.90 \times 10^3}$	9.25×10^{-1}	2.01×10^3	1.90×10^1	1.91×10^3	8.56×10^{-1}

o melhor dentre os algoritmos propostos foi o PSO. Desta forma, os resultados indicam que o PSO é o melhor dos cinco algoritmos propostos na otimização as funções de *benchmark* descritas na Tabela 1. Para testar esta observação, nós realizamos uma avaliação experimental de nossos dados seguindo o procedimento estatístico de Garcia e Herrera [2008], que é composto de três passos. Os três passos são descritos abaixo. Eles assumem um nível de significância $\alpha = 0.05$, i.e., a hipótese nula é rejeitada se um p-valor menor ou igual a 0.05 for obtido.

No primeiro passo, um teste de normalidade de Shapiro-Wilk é aplicado para verificar se as médias dos valores obtidos pelos algoritmos seguem uma distribuição normal. Com um p-valor de 0.001, o teste indicou que os dados das cinco heurísticas não provêm de uma distribuição normal. Deste modo, um teste de hipóteses não-paramétrico deve ser utilizado no próximo passo.

No segundo passo foi aplicado um Teste de Friedman para verificar se existe uma diferença significativa entre, ao menos, duas das heurísticas avaliadas. A hipótese nula é de que os cinco algoritmos tem o mesmo valor médio. Os dados foram ranqueados conforme proposto por Carvalho [2019]. Com um p-valor de 0.002, este teste rejeitou a hipótese nula. Deste modo, ele indicou que

existe uma diferença significativa entre as médias obtidas pelos algoritmos propostos neste trabalho.

No terceiro passo, foi aplicado o teste *post-hoc* não-paramétrico de Nemenyi, também conhecido como teste de Nemenyi–Damico–Wolfe–Dunn. Este teste compara os resultados de múltiplos algoritmos e avalia o par de hipóteses

$$\begin{cases} H_0: & \mu_i \leq \mu_j \\ H_1: & \mu_i \neq \mu_j \end{cases}, \qquad \forall (\mu_i, \mu_j) \in W,$$

onde $W = \{\mu_{\text{GA}}, \mu_{\text{DE}}, \mu_{\text{PSO}}, \mu_{\text{ACO}}, \mu_{\text{CLONAL}}\}$, tal que $\mu_{\text{GA}}, \mu_{\text{DE}}, \mu_{\text{PSO}}, \mu_{\text{ACO}}, \mu_{\text{CLONAL}}$ referem-se, respectivamente, aos rankings médios obtidos pelos algoritmos GA, DE, PSO, ACO e CLONAL no segundo passo. A hipótese nula (H_0) afirma que o ranking médio μ_i e μ_j de dois algoritmos não diferem significativamente, implicando que os resultados de uma heurística não são melhores que as de outra. Já a hipótese alternativa (H_1) implica que o resultado obtido pelo algoritmo μ_i é significativamente diferente dos resultados obtidos pelo algoritmo μ_j.

Este teste indicou que existe uma diferença significativa entre o PSO e os demais algoritmos avaliados neste trabalho. Isto é, ele indicou que $\mu_{\text{PSO}} \leq \mu_j : j \in W \setminus \mu_{\text{PSO}}$. Desta forma, podemos concluir que o PSO é o melhor dos cinco algoritmos avaliados ao resolver as funções de otimização global propostas.

4. Conclusões e Trabalhos Futuros

Este artigo apresentou uma avaliação experimental de cinco meta-heurísticas populares para a otimização global: algoritmo genético (GA), algoritmo de evolução diferencial (DE), algoritmo de seleção clonal (CLONAL), algoritmo de colônia de formigas (ACO) e algoritmo de enxame de partículas (PSO). O objetivo principal foi comparar o desempenho desses algoritmos em termos de eficiência e qualidade das soluções encontradas.

Os resultados obtidos mostraram que cada meta-heurística apresenta características distintas e desempenho variável, dependendo do problema de otimização em questão. O GA mostrou-se eficiente na exploração do espaço de busca, permitindo uma rápida convergência, enquanto o DE destacou-se pela sua capacidade de realizar buscas locais precisas. O algoritmo CLONAL demonstrou bom desempenho em problemas com múltiplos ótimos locais, adaptando-se bem às variações do espaço de busca. Por sua vez, o ACO mostrou-se eficaz na exploração de problemas complexos com restrições, utilizando a comunicação entre as formigas para guiar a busca global. Já o PSO, baseado na ideia de simulação de enxame, revelou-se altamente eficiente na busca por soluções ótimas globais, especialmente em problemas de alta dimensionalidade. De fato, uma análise estatística demonstrou que o PSO foi o melhor dos algoritmos avaliados neste trabalho.

Como trabalhos futuros, planeja-se desenvolver um algoritmo co-evolucionário utilizando as cinco meta-heurísticas analisadas e comparadas neste estudo. Tal algoritmo co-evolucionário utilizará um esquema de ilhas (ou subpopulações), de maneira semelhante ao recente artigo de Xavier et al. [2023]. Neste futuro estudo espera-se definir um conjunto de parâmetros, algoritmos e métodos de comunicação entre as subpopulações que seja extremamente efetivo para a otimização global.

Referências

Carvalho, I. A. (2019). On the statistical evaluation of algorithmic's computational experimentation with infeasible solutions. *Information Processing Letters*, 143:24–27.

De Castro, L. N. e Von Zuben, F. J. (2000). The clonal selection algorithm with engineering applications. In *Proceedings of GECCO*, volume 2000, p. 36–39.

Garcia, S. e Herrera, F. (2008). An extension on"statistical comparisons of classifiers over multiple data sets"for all pairwise comparisons. *Journal of Machine Learning Research*, 9(Dec):2677–2694.

Horst, R., Pardalos, P. M., e Van Thoai, N. (2000). *Introduction to global optimization*. Springer Science & Business Media.

Kennedy, J. e Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, p. 1942–1948. IEEE.

Liang, J., Qu, B., Suganthan, P., e Chen, Q. (2014). Problem definitions and evaluation criteria for the cec 2015 competition on learning-based real-parameter single objective optimization. Technical Report Technical Report201411A, Zhengzhou University and Nayang Technologial University.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., e Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Pardalos, P. M. e Romeijn, H. E. (2013). *Handbook of global optimization: Volume 2*, volume 62. Springer Science & Business Media.

Reeves, C. R. (2010). Genetic algorithms. In *Handbook of metaheuristics*, p. 109–139. Springer.

Socha, K. e Dorigo, M. (2008). Ant colony optimization for continuous domains. *European journal of operational research*, 185(3):1155–1173.

Storn, R. e Price, K. (1997). Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341.

Wang, D., Tan, D., e Liu, L. (2018). Particle swarm optimization algorithm: an overview. *Soft computing*, 22:387–408.

Xavier, C. R., Silva, J. G. R., Duarte, G. R., Carvalho, I. A., Vieira, V. d. F., e Goliatt, L. (2023). An island-based hybrid evolutionary algorithm for caloric-restricted diets. *Evolutionary Intelligence*, 16(2):553–564.

Zhigljavsky, A. e Zilinskas, A. (2007). *Stochastic global optimization*, volume 9. Springer Science & Business Media.