Uma Variante Melhorada do Algoritmo Busca Cuco usando uma Estratégia de *Quasi Opposition–Based Learning*

Cácio L. N. A. Bezerra[1], **Fábio G. B. C. Costa**[1], **Gabriel M. Nascimento**[1], **Pedro V. M. Carvalho**[1] **e Fábio A. P. Paiva**[1]

[1] Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte (IFRN)
Campus Parnamirim – RN – Brasil

{cacio.lucas, fabio.gabriel, gabriel.moraes}@escolar.ifrn.edu.br,

pedro.miranda@escolar.ifrn.edu.br, fabio.procopio@ifrn.edu.br

Abstract. *– Metaheuristics aim to solve optimization problems. Cuckoo Search (CS) is a metaheuristic that is based on the agressive reproduction behavior of the cuckoo birds. This paper combines CS with Quasi Opposition–Based Learning (QOBL) and proposes a variant called CS–QOBL. The variant aims to increase the convergence speed of the original CS and improve its performance. To evaluate CS–QOBL, 10 benchmark functions were used. Commonly they are employed to validate new algorithms. In all experiments, CS–QOBL presented a better result and a higher convergence speed compared to original CS.*

Resumo. *– Meta-heurísticas objetivam resolver problemas de otimização. Cuckoo Search (CS) é uma meta-heurística que é baseada no comportamento agressivo de reprodução dos pássaros cuco. Este trabalho combina CS com a Aprendizagem Baseada em Quase–Oposição (QOBL) e propõe uma variante denominada CS–QOBL. A variante objetiva aumentar a velocidade de convergência do CS original e melhorar sua performance. Para avaliar CS–QOBL, foram usadas 10 funções de referências. Comumente elas são empregadas para validar novos algoritmos. Em todos os experimentos, CS–QOBL apresentou melhores resultados e maior velocidade de convergência comparados ao CS original.*

1. Introdução

Os algoritmos evolucionários são meta-heurísticas que, nos últimos anos, vêm sendo frequentemente empregados na resolução de vários problemas de otimização. Algoritmo genético (GA), programação evolutiva (EP), estratégias de evolução (ES), programação genética (GP) e sistemas classificadores de aprendizagem (LCS) são as cinco técnicas que constituem os algoritmos evolucionários [Eiben et al. 2003]. Recentemente, uma nova meta-heurística denominada Busca Cuco [Yang and Deb 2009] tem se mostrado promissora para otimização global e tem se tornado uma nova área de pesquisa da computação evolucionária [Feng et al. 2014].

Busca Cuco (*Cuckoo Search* – CS) é um algoritmo inspirado no parasitismo das crias de algumas espécies do pássaro cuco que se caracterizam por colocar ovos em ninhos de outras espécies de aves hospedeiras. CS vem sendo empregado no cenário de diversos problemas reais de otimização como sistemas de distribuição [Hung et al. 2014], ajuste de parâmetros [Cheung et al. 2017] e outros.

Quando CS é comparado a outros algoritmos meta-heurísticos tradicionais, como GA e Otimização por Enxame de Partículas (PSO), percebe-se a simplicidade de sua implementação. Porém, apesar de ser simples, uma de suas desvantagens é a baixa taxa de convergência e, como consequência, a facilidade de "cair" em ótimos locais.

Para melhorar o desempenho do CS original, vários trabalhos já foram propostos. A variante *Gaussian Cuckoo Search* (GCS) foi apresentada por [Zheng and Zhou 2012] e é baseada em uma distribuição gaussiana. *Gradient-Based Cuckoo Search* (GBCS) [Fateen and Bonilla-Petriciolet 2014] apresenta uma modificação no CS com base no gradiente da função objetivo. Já a *Double Mutation Cuckoo Search* [Qu and He 2015] usa uma pequena taxa de probabilidade para modificar os ninhos parasitas com os melhores *fitness* e uma alta taxa de probabilidade para modificar os ninhos parasitas precários.

Este trabalho apresenta uma modificação no algoritmo CS original a partir do uso de uma estratégia conhecida como *Quasi Opposition–Based Learning* (QOBL) [Rahnamayan et al. 2007]. O objetivo do trabalho é aumentar a velocidade de convergência do CS a fim de melhorar a qualidade das soluções. Os experimentos computacionais evidenciaram a superioridade da variante CS–QOBL quando comparada ao CS e também à outra variante, também implementada neste trabalho, chamada CS–OBL.

O trabalho está organizado da seguinte forma: na Seção 2, é apresentada uma breve introdução sobre a meta-heurística CS e a estratégia QOBL; na Seção 3, é apresentada a nova variante modificada com a QOBL. Na Seção 4, os experimentos computacionais são apresentados e os resultados são discutidos. Por fim, a Seção 5 apresenta as considerações finais e os trabalhos futuros.

2. Fundamentação Teórica

Esta seção apresenta algumas informações úteis para o entendimento da variante que será apresentada, como o algoritmo Busca Cuco e a *Quasi Opposition–Based Learning*.

2.1. Algoritmo Busca Cuco

Cucos são pássaros fascinantes não apenas pelo seu belo canto, mas por apresentar uma estratégia de reprodução agressiva. Algumas espécies como *ani* e *Guira* colocam seus ovos em ninhos comunitários. Há também a possibilidade de elas removerem os ovos de aves hospedeiras para aumentar a probabilidade de incubação dos seus próprios ovos. Muitas espécies se engajam no parasitismo da ninhada colocando seus ovos nos ninhos de outros pássaros hospedeiros que, em geral, são de outras espécies [Payne and Sorensen 2005]. São três tipos básicos de parasitismo: a) intraespecífico de ninhada, b) reprodução cooperativa e, c) ninho por aquisição.

Cuckoo Search é uma meta-heurística inspirada no comportamento parasita dos pássaros cucos desenvolvida por [Yang and Deb 2009]. Ela considera três regras: 1) cada cuco coloca um ovo por vez e o deposita em um ninho escolhido aleatoriamente; 2) os ninhos, com ovos de alta qualidade, serão escolhidos para as próximas gerações; 3) o número de ninhos disponível é fixo e o ovo colocado por um cuco é descoberto pelo pássaro hospedeiro com uma taxa de probabilidade que varia no intervalo [0, 1]. CS usa uma combinação balanceada de caminhada aleatória local (busca local) e caminhada aleatória exploratória global (busca global), controlada por um parâmetro p_a.

A busca local pode ser escrita como segue

$$x_i^{t+1} = x_i^t + \alpha s \oplus H(p_a - \epsilon) \oplus (x_j^t - x_k^t), \tag{1}$$

onde t é a iteração atual, x_j^t e x_k^t são duas soluções diferentes selecionadas aleatoriamente por uma permutação aleatória, α é o fator de escalonamento de s, s é o tamanho do passo, $\oplus$ é o produto interno entre dois vetores, $H(.)$ é a função de Heaviside [Von Karman and Biot 1940] e ϵ é um número gerado aleatoriamente a partir de uma distribuição uniforme. Já a busca global é realizada usando os voos de Lévy, conforme equação a seguir

$$x_i^{t+1} = x_i^t + \alpha L(s, \lambda), \tag{2}$$

onde

$$L \sim \frac{\lambda \Gamma(\lambda) sen(\pi\lambda/2)}{\pi} \frac{1}{s^{1+\lambda}}, (s \gg s_0 > 0). \tag{3}$$

Γ é a função gama padrão, cuja distribuição é válida para grandes passos. No CS original, $\lambda = 1.5$ e $\alpha = 0.01$, conforme [Yang and Deb 2009]. No algoritmo proposto, apresentado na próxima seção, os valores dessas constantes são os mesmos usados no original.

A geração de tamanhos de passos pseudo-aleatórios que obedeçam, corretamente, a Equação 3 não é uma tarefa trivial [Yang 2012]. Existem alguns métodos para geração de números aleatórios e o mais eficiente é o algoritmo de Mantegna [Mantegna 1994]. Nesse algoritmo, o tamanho do passo s pode ser calculado usando duas distribuições gaussianas U e V, conforme equação a seguir

$$s = \frac{U}{|V|^{1/\lambda}}, \tag{4}$$

com

$$U \sim N(0, \sigma^2), \;\; V \sim N(0, 1), \tag{5}$$

onde $U \sim N(0, \sigma^2)$ significa que as amostras são geradas a partir de uma distribuição normal gaussiana, com média zero e variância σ^2, calculada por

$$\sigma^2 = \left[\frac{\Gamma(1+\lambda)}{\lambda \Gamma((1+\lambda)/2)} \cdot \frac{\sin(\pi\lambda/2)}{2^{(\lambda-1)/2}} \right]^{1/\lambda}. \tag{6}$$

Baseado nas três regras definidas anteriormente, os passos básicos do CS podem ser sumarizados no pseudocódigo do Algoritmo 1. Inicialmente, a população de ninhos é gerada aleatoriamente (linha 1). Na linha 3, um cuco é escolhido aleatoriamente por meio do voo de Lévy, conforme Equação 2. Na linha 4, a função objetivo do ninho anterior f_i é avaliada. Na linha seguinte, escolhe-se aleatoriamente um ninho j. Na sequência, na linha 6, é verificado se o valor f_i é melhor que f_j (valor calculado na função objetivo f para o ninho j), em caso positivo, o ninho j é substituído pelo novo ninho i (linha 7). Na linha 9, uma fração (representada pelo parâmetro p_a) dos piores ninhos é abandonada. Na linha 10, as soluções são avaliadas e as melhores são mantidas. Por fim, na linha 11, as soluções são classificadas e a melhor delas é selecionada.

Algoritmo 1 Pseudocódigo do algoritmo Busca Cuco original

Gere a população inicial de n ninhos hospedeiros $x_i = (i = 1, 2, ..., n)$
enquanto critério de parada não for atingido **faça**
    Obtenha um cuco aleatoriamente usando o voo de Lévy
    Avalie a função objetivo f_i
    Escolha um ninho aleatório j
    se $f_i > f_j$ **então**
        Substitua j pela nova solução
    fim se
    Abandone uma fração p_a dos piores ninhos e construa novos
    Mantenha as melhores soluções
    Classifique as soluções e encontre a melhor delas
fim enquanto

2.2. *Quasi Opposition–Based Learning*

Os algoritmos meta-heurísticos são iniciados com soluções aleatórias e, ao longo do tempo, eles seguem em direção da solução ótima. Quando as soluções iniciais estão próximas do ponto ótimo, a convergência é rápida, porém se estão distantes, a convergência se torna mais lenta. O pior caso ocorre quando elas estão posicionadas do lado oposto da solução ótima, o que pode demandar muito tempo ou, definitivamente, impraticável de localizá-la.

Uma solução para esse problema é buscar, simultaneamente, em todas as direções ou, simplesmente, na direção oposta. Para lidar com esta situação, a Aprendizagem Baseada em Oposição (OBL – *Opposition-Based Learning*) [Tizhoosh 2005] pode ser usada. A seguir, são definidos Número Oposto e Ponto Oposto.

***Definição 1*:** Dado $x \in \mathbb{R}$, no intervalo $x \in [a, b]$, o número oposto $\tilde{x}$ é definido na equação a seguir.

$$\tilde{x} = a + b - x. \tag{7}$$

***Definição 2*:** Dado $P = (x_1, x_2, ..., x_n)$ como sendo um ponto no espaço n-dimensional, com x_1, x_2, ..., $x_n \in \mathbb{R}$ e $x_i \in [a_i, b_i]$, $\forall i \in \{1, 2, ..., n\}$. Assim, o ponto oposto $\tilde{P} = (\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_n)$ é definido pelos seus componentes:

$$\tilde{x_i} = a_i + b_i - x_i. \tag{8}$$

Para melhorar o desempenho da OBL, diversas variantes foram apresentadas. Algumas delas são *Quasi Opposition–Based Learning* (QOBL) [Rahnamayan et al. 2007], *Generalized Opposition–Based Learning* (GOBL) [Wang et al. 2009], *Elite Opposition–Based Learning* (EOBL) [Zhou et al. 2012] e outros. A seguir, QOBL é definida.

***Definição 3*:** Dado $x \in \mathbb{R}$, no intervalo $x \in [a, b]$, seu ponto *quasi opposition–based learning*, x_{qo}, é definido por

$$x_{qo} = rand(c, \tilde{x_i}), \tag{9}$$

onde c é o centro do intervalo $[a, b]$, calculado por $c = (a + b)/2$ e $rand(c, \tilde{x_i})$ é um número aleatório uniformemente distribuído entre c e $\tilde{x_i}$.

3. Algoritmo Busca Cuco Modificado com *Quasi Opposition–Based Learning*

Neste trabalho, algumas variantes da OBL – como GOBL, QOBL e EOBL – foram combinadas com CS a fim de avaliar os seus resultados. Depois da avaliação dessas combinações, aquela que apresentou os resultados mais satisfatórios foi CS e QOBL e, por esse motivo, a variante CS apresentada propõe a combinação de CS e QOBL.

A variante proposta tem como objetivo adicionar um mecanismo que possibilite o algoritmo original acelerar a sua velocidade de convergência. A contribuição do trabalho é apresentada na linha 11, quando a função QOBL, definida nas linhas 1 – 6, é invocada. Como parâmetro, a função QOBL recebe um $ninho$. Na linha 2, a função usa a Equação (8) para calcular o oposto de $ninho$. Em seguida, para cada dimensão de $ninho$, QOBL é calculada, conforme linhas 3 – 5, usando Equação (9).

Algoritmo 2 Pseudocódigo do Algoritmo Busca Cuco usando QOBL

1: **função** QOBL($ninho^t$)
2: Calcule $ninho^t_o$, conforme Equação (8)
3: **para** $k \leftarrow 1$ **até** dim_{ninho^t} **faça**
4: Calcule $ninho^t_{qo}$, conforme Equação (9)
5: **fim para**
6: **fim função**
7: Gere a população inicial de n ninhos hospedeiros $x_i = (i = 1, 2, ..., n)$
8: **enquanto** critério de convergência não for atingido **faça**
9: **para** $i \leftarrow 1$ **até** n **faça**
10: Obtenha um cuco aleatoriamente usando o voo de Lévy
11: $ninho^t \leftarrow$ QOBL($ninho^t$)
12: Idem às linhas 4–10 do Algoritmo 1
13: **fim para**
14: Classifique os ninhos e encontre o melhor
15: **fim enquanto**

4. Experimentos Numéricos

Esta seção apresenta as funções de referência usadas para validar o algoritmo proposto e, no final, as simulações e os resultados encontrados são discutidos.

4.1. Funções de Referência

É comum usar funções de referência com a hipótese de que a dificuldade apresentada por elas corresponde àquelas encontradas em aplicações reais. Assim, elas são frequentemente usadas para validar e comparar os algoritmos de otimização, bem como para validar novas abordagens de otimização global [Dieterich and Hartke 2012].

Para realizar os experimentos, foram escolhidas 10 funções de referência que são aplicadas em problemas de minimização e utilizadas em alguns estudos de CS [Fateen and Bonilla-Petriciolet 2014, Qu and He 2015]. Para cada uma delas, a fórmula, o espaço de busca e a solução ótima são apresentados na Tabela 1. Essas funções são classificadas em dois grupos:

1. Grupo 1 – formado por funções unimodais que, geralmente são usadas para testar a capacidade do algoritmo em buscas locais. As funções unimodais usadas nos experimentos são: Esfera (f_1), *Step* (f_2), Schumer Steiglitz (f_3), *Powell Sum* (f_4) e Cigar (f_5).
2. Grupo 2 – formado por funções multimodais que possuem vários mínimos locais. Elas são usadas para verificar a habilidade do algoritmo para "escapar" de ótimos locais. São elas: Ackley (f_6), Rastrigin (f_7), Griewank (f_8), Salomon (f_9) e Alpine (f_{10}).

Tabela 1. Funções de referência utilizadas para avaliar CS, CS–OBL e CS–QOBL.

Fórmula	Espaço de Busca	Ótima
$f_1(x) = \sum_{i=1}^{d} x_i^2$	$-100 \leq x_i \leq 100$	$f(x^*) = 0$
$f_2(x) = \sum_{i=1}^{d-1} (\lvert x_i \rvert + 0.5)^2$	$-100 \leq x_i \leq 100$	$f(x^*) = 0$
$f_3(x) = \sum_{i=1}^{d} x_i^4$	$-100 \leq x_i \leq 100$	$f(x^*) = 0$
$f_4(x) = \sum_{i=1}^{d} \lvert x_i \rvert^{i+1}$	$-500 \leq x_i \leq 500$	$f(x^*) = 0$
$f_5(x) = x_i^2 + \sum_{i=2}^{d} x_i^2$	$-10 \leq x_i \leq 10$	$f(x^*) = 0$
$f_6(x) = -20\ exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right) + 20 + exp(1)$	$-32 \leq x_i \leq 32$	$f(x^*) = 0$
$f_7(x) = \sum_{i=1}^{d} [x_i^2 - 10cos(2\pi x_i) + 10]$	$-5.12 \leq x_i \leq 5.12$	$f(x^*) = 0$
$f_8(x) = \frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$-600 \leq x_i \leq 600$	$f(x^*) = 0$
$f_9(x) = 1 - cos\left(2\pi\sqrt{\sum_{i=1}^{d} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{d} x_i^2}$	$-100 \leq x_i \leq 100$	$f(x^*) = 0$
$f_{10}(x) = \sum_{i=1}^{d} \lvert x_i sen(x_i) + 0.1x_i \rvert$	$-10 \leq x_i \leq 10$	$f(x^*) = 0$

4.2. Resultados e Discussões

Todas as rotinas foram implementadas em MATLAB R2014b. Os experimentos foram executados em um computador que usa um processador Intel Core i7, com 2,4 GHz de freqüência, 8 GB de memória RAM e *Windows 10 Home Single Language* (64 bits).

A variante CS–QOBL é comparada ao CS e a outra variante, também implementada neste trabalho, chamada CS–OBL. Em todos os experimentos, são realizadas 30 execuções independentes, usando 25 ninhos, taxa de probabilidade $p_a = 0.25$, número de dimensões igual a 50, ao longo de 3.000 iterações.

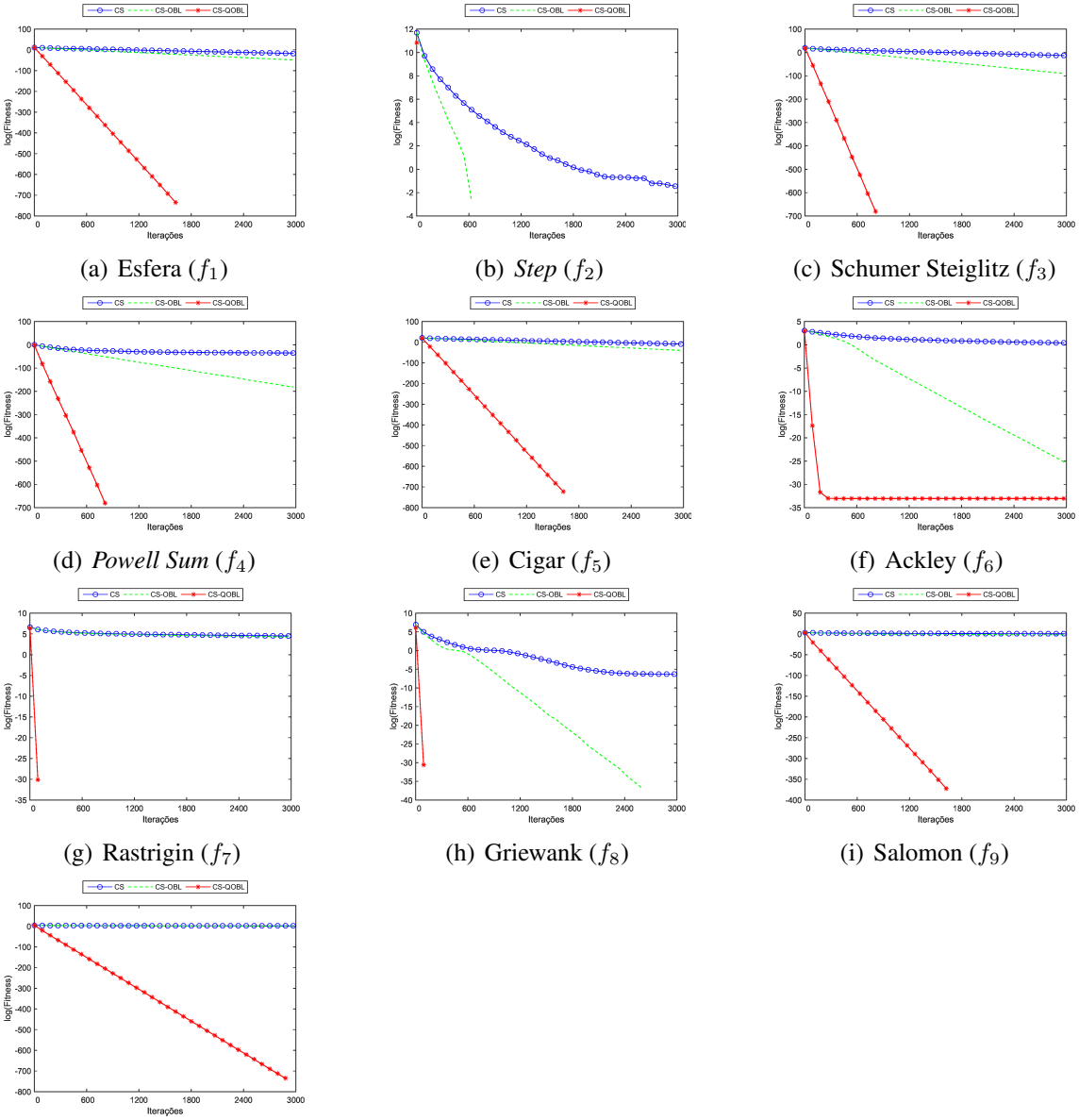
As subfiguras 1(a) – 1(e) mostram os resultados dos experimentos para as funções unimodais. Em 1(a), CS–QOBL apresenta uma superioridade em relação ao CS e ao

CS–OBL, ao encontrar o ponto ótimo, aproximadamente na iteração 1.650, enquanto CS e CS–OBL apresentam comportamento parecido nas primeiras iterações. A subfigura 1(b) mostra que CS–OBL e CS–QOBL encontram a solução ótima da função e, por isso, não há significância estatística nos resultados encontrados, apesar de sua velocidade de convergência ter sido superior. Entretanto observa-se um excelente desempenho da CS–QOBL ao encontrar o ótimo global no início do processo (entre as iterações 45 – 50). Na subfigura 1(c), CS–QOBL encontra a solução ótima da função Schumer Steiglitz perto da iteração 800, enquanto CS estagna nas iterações inicias. CS–OBL, tem um comportamento mais ativo que CS, porém inferior a CS–QOBL. A subfigura 1(d) mostra que o CS–QOBL encontra a solução ótima próximo da iteração 880, ao passo que CS e CS–OBL não encontram. O mesmo ocorre na subfigura 1(e), onde CS–QOBL encontra o ponto ótimo próximo da iteração 1.670.

As subfiguras 1(f) – (g) mostram os resultados dos experimentos para as funções multimodais. Na subfigura 1(f), CS–QOBL ficou estagnada da iteração 170 até a 3.000, aproximadamente, embora seu desempenho tenha superado CS e CS–OBL. Na subfigura 1(g), CS–QOBL encontrou o ótimo global nas primeiras iterações, enquanto CS e CS–QOBL demonstram um desempenho bastante inferior. A subfigura 1(h) mostra que tanto CS–QOBL quanto CS–OBL encontram os ótimos globais, entretanto a velocidade de convergência da CS–QOBL é maior. A subfigura 1(i) encontra o ótimo global próximo da iteração 1.700. Por outro lado, CS e CS–OBL apresentam um comportamento parecido com baixa velocidade de convergência. Por fim, a subfigura 1(j), apresenta convergência bastante ativa da CS–QOBL, enquanto CS e CS–OBL ficaram praticamente estagnados.

Wilcoxon é um teste estatístico não-paramétrico utilizado para comparar duas amostras independentes. Neste trabalho, ele é usado para verificar se os resultados da variante proposta são estatisticamente significativos quando comparados aos da variante CS–OBL, isto é, se o *p-value* é menor que o nível de significância determinado. Nos experimentos, o nível de significância foi definido em 0.05. O teste não comparou a variante proposta com CS original, uma vez que, de acordo com as subfiguras, a superioridade da variante é bastante evidente.

A Tabela 2 apresenta os resultados numéricos da comparação entre CS, CS–OBL e CS–QOBL. Para cada função, é mostrada a melhor e a pior solução, a média das soluções, o desvio padrão e o *p-value* resultante do teste de Wilcoxon. Os valores em negrito representam a melhor média e o desvio padrão dos experimentos. Em relação aos valores médios encontrados, CS–QOBL obtém resultados melhores que CS–OBL em 80% dos experimentos e, em relação ao CS, 100% deles. O desvio padrão, que mede a dispersão em torno da média, mostra que a variante proposta obteve valores bastante satisfatórios quando comparada ao CS original e à variante CS–OBL. Isso comprova a estabilidade das soluções.



(a) Esfera (f_1) (b) *Step* (f_2) (c) Schumer Steiglitz (f_3)

(d) *Powell Sum* (f_4) (e) Cigar (f_5) (f) Ackley (f_6)

(g) Rastrigin (f_7) (h) Griewank (f_8) (i) Salomon (f_9)

(j) Alpine (f_{10})

Figura 1. Convergência média dos algoritmos nas funções avaliadas.

Tabela 2. Comparação do desempenho entre CS, CS–OBL e CS–QOBL.

F	CS		CS–OBL		CS-QOBL		p-Value
	Melhor	Média (Desvio)	Melhor	Média (Desvio)	Melhor	Média (Desvio)	
f_1	3.07e-09	1.97e-08 (2.25e-08)	1.43e-23	4.44e-22 (4.17e-22)	0	**0 (0)**	1.86e-09
f_2	0	6.67e-02 (2.54e-01)	0	**0 (0)**	0	**0 (0)**	1
f_3	6.57e-09	1.49e-06 (2.76e-06)	1.02e-41	4.52e-40 (1.25e-39)	0	**0 (0)**	1.86e-09
f_4	7.52e-25	4.62e-16 (2.53e-15)	1.01e-84	2.37e-80 (5.84e-80)	0	**0 (0)**	1.86e-09
f_5	1.21e-05	9.74e-05 (5.89e-05)	1.51e-19	3.45e-18 (2.29e-18)	0	**0 (0)**	1.86e-09
f_6	9.50e-04	1.48 (0.60)	7.33e-13	8.69e-12 (0.61e-11)	4.44e-15	**4.44e-15 (0)**	1.86e-09
f_7	67.15	90.77 (16.72)	26.9	72.8 (23.9)	0	**0 (0)**	1.86e-09
f_8	3.06e-09	1.81e-03 (3.77e-03)	0	**0 (0)**	0	**0 (0)**	1
f_9	1.20	1.52 (0.20)	9.98e-02	0.13 (4.98e-02)	0	**0 (0)**	1.86e-09
f_{10}	3.66	6.95 (2.02)	1.41e-07	1.23 (2.43)	0	**0 (0)**	1.86e-09

5. Conclusões

Neste trabalho, foi apresentada uma nova variante da meta-heurística *Cuckoo Search* que consiste na sua combinação com a aprendizagem baseada em quase–oposição. A proposta tem como objetivo gerar diversidade de soluções para aumentar a velocidade de convergência do algoritmo original.

Os experimentos computacionais compararam a performance do CS original, da variante CS–OBL e da variante CS–QOBL. Para isso, foram usadas 10 funções de referência, com características de unimodalidade e multimodalidade. Em 100% dos experimentos, CS–QOBL superou CS original e quando CS–OBL não encontrou o ponto ótimo da função, também foi superada por CS–QOBL. Em 90% dos experimentos, CS–QOBL encontrou a solução ótima da função. A variante proposta não encontrou a solução ótima apenas da função Ackley (f_6). Nas primeiras iterações, a velocidade de convergência da variante foi alta, porém próximo da iteração 170, o algoritmo estagnou em um ótimo local cujo valor de *fitness* é 4.44e-15, um valor que, dependendo da precisão do problema, pode ser considerado um bom resultado.

Como trabalhos futuros, pretende-se implementar *Quasi Opposition–Based Learning* (QOBL) em outras meta-heurísticas bioinspiradas com o intuito de minimizar a perda da capacidade de gerar novas soluções e aumentar a velocidade de convergência do algoritmo original.

Referências

Cheung, N. J., Ding, X.-M., and Shen, H.-B. (2017). A nonhomogeneous cuckoo search algorithm based on quantum mechanism for real parameter optimization. *IEEE transactions on cybernetics*, 47(2):391–402.

Dieterich, J. M. and Hartke, B. (2012). Empirical review of standard benchmark functions using evolutionary global optimization. *arXiv preprint arXiv:1207.4318*.

Eiben, A. E., Smith, J. E., et al. (2003). *Introduction to evolutionary computing*, volume 53. Springer.

Fateen, S.-E. K. and Bonilla-Petriciolet, A. (2014). Gradient-based cuckoo search for global optimization. *Mathematical Problems in Engineering*, 2014.

Feng, Y., Jia, K., and He, Y. (2014). An improved hybrid encoding cuckoo search algorithm for 0-1 knapsack problems. *Comp. intelligence and neuroscience*, 2014:1.

Hung, D. Q., Mithulananthan, N., and Bansal, R. C. (2014). An optimal investment planning framework for multiple distributed generation units in industrial distribution systems. *Applied Energy*, 124:62–72.

Mantegna, R. N. (1994). Fast, accurate algorithm for numerical simulation of levy stable stochastic processes. *Physical Review E*, 49(5):4677.

Payne, R. B. and Sorensen, M. D. (2005). *The cuckoos*, volume 15. Oxford Univ. Press.

Qu, C. and He, W. (2015). A double mutation cuckoo search algorithm for solving systems of nonlinear equations. *Int. J. of Hybrid Information Technology*, 8(12):433–448.

Rahnamayan, S., Tizhoosh, H. R., and Salama, M. M. (2007). Quasi-oppositional differential evolution. In *IEEE Congress on Evolutionary Computation*, pages 2229–2236.

Tizhoosh, H. R. (2005). Opposition-based learning: a new scheme for machine intelligence. In *Int. Conf. on Computational intelligence for modelling, control and automation and Int. Conf. on intelligent agents, web technologies and internet commerce*, volume 1, pages 695–701.

Von Karman, T. and Biot, M. A. (1940). *Mathematical methods in engineering*. McGraw Hill.

Wang, H., Wu, Z., Liu, Y., Wang, J., Jiang, D., and Chen, L. (2009). Space transformation search: a new evolutionary technique. In *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pages 537–544. ACM.

Yang, X.-S. (2012). Flower pollination algorithm for global optimization. In *Int. Conf. on Unconventional Computing and Natural Computation*, pages 240–249. Springer.

Yang, X.-S. and Deb, S. (2009). Cuckoo search via lévy flights. In *World Congress on Nature & Biologically Inspired Computing.*, pages 210–214.

Zheng, H. and Zhou, Y. (2012). A novel cuckoo search optimization algorithm based on gauss distribution. *Journal of Computational Information Systems*, 8(10):4193–4200.

Zhou, X., Wu, Z., and Wang, H. (2012). Elite opposition-based differential evolution for solving large-scale optimization problems and its implementation on gpu. In *13th Int. Conf. on Parallel and Distributed Computing, Applications and Technologies*, pages 727–732.