Avaliação de um algoritmo genético multi-populacional para otimização global

RESUMO

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. Os algoritmos genéticos são um tipo de meta-heurística inspiradas no processo de evolução natural que têm sido amplamente utilizadas em diversas áreas. Uma abordagem multi-populacional é uma extensão desses algoritmos, onde várias populações independentes são mantidas simultaneamente. Neste estudo, comparamos o desempenho de um algoritmo genético multi-populacional utilizando duas configurações: uma com apenas uma ilha (população única) e outra com diversas ilhas (populações separadas). Como casos de teste, utilizamos um conjunto de funções matemáticas diverso, contendo funções não-separáveis, híbridas e rotacionadas, tradicionalmente utilizadas como *benchmark* para a avaliação deste algoritmo. Os resultados mostram que a abordagem com diversas ilhas foi superior, destacando então a importância da distribuição do processo de busca em múltiplas populações para melhorar a eficiência e eficácia dos algoritmos genéticos.

PALAVRAS CHAVE. Algoritmo genético, Modelo multi-populacional, Otimização global.

Meta-heurísticas, Inteligência computacional

ABSTRACT

Metaheuristics stand out among global optimization algorithms, being generic and efficient methods that can be applied to solve a wide variety of problems. Genetic algorithms are a type of meta-heuristics inspired by the process of natural evolution that have been widely used in several areas. A multi-population approach is an extension of these algorithms, where several independent populations are maintained simultaneously. In this study, we compare the performance of a multi-population genetic algorithm using two configurations: one with only one island (single population) and another with several islands (separate populations). As test cases, we use a diverse set of mathematical functions, containing non-separable, hybrid and rotated functions, traditionally used as *benchmark* for the evaluation of this algorithm. The results show that the multi-populational approach was superior, thus highlighting the importance of distributing the search process across multiple populations to improve the efficiency and effectiveness of genetic algorithms.

KEYWORDS. Genetic algorith, multi-populational model, Global optimization

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.

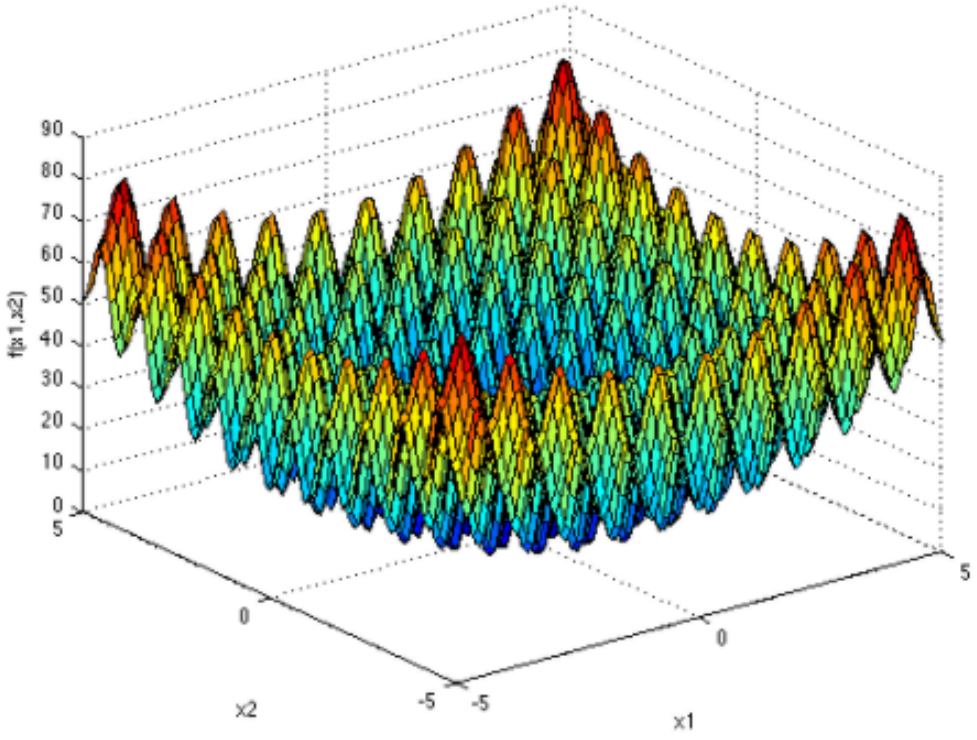

Figura 1: A função Rastringin em duas dimensões

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].

Os algoritmos genéticos (GA, do inglês *Genetic Algorithm*) são algoritmos evolutivos que têm se mostrado eficazes na resolução de problemas complexos de otimização em diversas áreas da pesquisa operacional [Katoch et al., 2021]. Estes são algoritmos genéricos e eficientes que podem ser utilizados para otimizar problemas combinatórios [Xavier et al., 2023], lineares [Das e Pratihar, 2021] ou não-lineares [Reeves, 2010]. Estes algoritmos, conforme proposto por Holland

[1975], baseiam-se no princípio da evolução natural de Charles Darwin para otimizar um conjunto de soluções, denominado população, buscando o ótimo global de um problema.

Existem diversas implementações paralelas e/ou distribuídas de GAs [Tomassini, 2005]. Uma das mais utilizadas é o modelo de ilhas [Li et al., 2008; Xavier et al., 2023], onde diversos conjuntos de soluções evoluem de forma semi-isolada. É dito que cada um desses conjuntos de soluções está localizado em uma ilha. Além disso, as ilhas trocam soluções entre si periodicamente, em um processo denominado *migração*, como forma de acelerar a convergência global do algoritmo.

Neste trabalho estamos interessados em estudar a implementação em ilhas de GAs. Nós realizamos uma análise comparativa de um GA sendo executado em uma única ilha frente a um GA onde existem diversos conjuntos de soluções divididos em múltiplas ilhas. Experimentos computacionais, realizados utilizando 15 diferentes funções matemáticas desenvolvidas como *benchmark* em uma competição de otimização global [Liang et al., 2014], demonstraram que a versão multi-populacional do GA é capaz de melhor aproximar o ótimo global das funções de *benchmark* propostas quando comparado a sua versão com uma única população.

2. Revisão bibliográfica

Diversos estudos sobre algoritmos evolutivos multi-populacionais (AEMP) podem ser encontrados na literatura. Uma referência inicial é o livro de Tomassini [2005]. Este apresentou diversos modelos distribuídos e paralelos para a implementação de algoritmos evolucionários. Além dos AEMP, ele apresentou modelos de rede celulares e modelos co-evolucionários, dentre outros. Foram apresentados e exemplificados diversos parâmetros destes modelos e aplicações destes algoritmos para problemas de otimização.

Lässig e Sudholt [2013] demonstraram o impacto do operador de migração na diversidade de soluções e convergência de AEMPs. Os resultados indicaram que uma alta frequência de migração e um alto número de soluções migrantes em cada aplicação deste operador faz com que as múltiplas populações comportem-se como uma única população, desfazendo todos os efeitos benéficos da estrutura de ilhas. Além disso, foi demonstrado também que uma baixa frequência de migração pode retardar o processo evolutivo, fazendo com que a otimização seja mais lenta do que o esperado.

As ilhas podem ser conectadas de diversas maneiras, cada uma gerando uma diferente topologia de migração. O trabalho de Sekaj [2004] avaliou diversas destas topologias, como a de anel, de vetor, de grade, de estrela e completa. Os autores mostraram que topologias com conexões mais esparsas tem uma maior diversidade de populações. Além disso, eles demonstraram que uma topologia altamente conectada pode se comportar como uma única população, resultado similar ao encontrado por Lässig e Sudholt [2013].

Ruciński et al. [2010] complementou o trabalho de Sekaj [2004] avaliando, além das diferentes topologias, o número de ilhas. Além disso, ele também utilizou dois diferentes algoritmos evolutivos, sendo que cada algoritmo evolutivo era aplicado para evoluir as soluções de diversas ilhas. Os autores estudaram e demonstraram o efeito de diferentes parâmetros no comportamento do AEMP desenvolvido.

3. Algoritmo Genético

O Algoritmo (1) descreve o pseudo-código da implementação de nosso GA. Inicialmente, a linha 1 inicializa a população e a linha 2 faz a avaliação da primeira população. Então, o *loop* das linhas 3–10 é executado até que um critério de parada pré-definido seja atendido. Pode-se afirmar que cada iteração deste *loop* corresponde a uma geração de nosso algoritmo genético. Em cada iteração são aplicados, de forma sequencial, os operadores de seleção, cruzamento, mutação e evolução. Por fim, o melhor indivíduo é retornado na última linha. Os operadores utilizados são explicados detalhadamente a seguir.

Algorithm 1 Algoritmo Genético

```
Inicialização da população
Avaliação da população
while critério de parada não é atendido do
    Seleção dos pais
    Cruzamento
    Mutação
    Avaliação da população intermediária
    Evolução
end while
return o melhor indivíduo encontrado
```

3.1. Representação de uma solução

Seja X uma população constituída por um conjunto de indivíduos $x_i \in X$. Cada indivíduo é representado 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. Denota-se por x_i^j, a j-ésima posição do vetor x_i da solução $x_i \in X$, sendo $0 < j \leq d$, sendo que cada posição do vetor é referenciada como um *gene*. 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 $d = 5$ genes

3.2. Inicialização da população

Cada indivíduo $x_i \in X$ é inicializado de forma aleatória. Para cada gene x_i^j, sorteia-se um valor utilizando uma distribuição uniforme contínua $\mathcal{U}(-100, 100)$ através de um gerador de números pseudo-aleatórios. Este processo é repetido para todos os $|X|$ indivíduos da população inicial.

3.3. Seleção dos pais

A seleção é realizada utilizando um algoritmo de roleta como descrito em Reeves [2010]. Todo indivíduo $x_i \in X$ é associado a uma probabilidade de seleção p_i, sendo que p_i é inversamente proporcional ao valor de *fitness* $f(x_i)$. Desta forma, temos que, quanto melhor é o *fitness* de um indivíduo, maior é o seu valor de p_i.

Utilizando os valores de p_i acima definidos é construído um vetor P tal que $|P| = \sum_{i=0}^{|X|} p_i$. Cada indivíduo $x_i \in X$ recebe um número p_i de chaves neste vetor. Então, o algoritmo de roleta sorteia dois diferentes números pseudo-aleatórios $r1$ e $r2$ utilizando uma distribuição uniforme $\mathcal{U}(0, |P|)$. Os pais selecionados são os indivíduos com chaves associadas as posições $P[r1]$ e $P[r2]$.

Caso o algoritmo acima selecione duas vezes o mesmo indivíduo como pai, um novo valor $r2$ é sorteado. Este procedimento é realizado quantas vezes o necessário até que dois diferentes indivíduos sejam selecionados.

3.4. Cruzamento

O GA utiliza o operador de cruzamento denominado *2-point crossover*. Este operador utiliza dois pais $a, b \in X$ escolhidos pela aplicação do operador de seleção para criar um novo indivíduo c. O i-ésimo gene do indivíduo c é definido como

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

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

3.5. Mutação

O operador de mutação utilizado altera o valor de um único gene do indivíduo c gerado previamente. 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 $\mathcal{U}(-100, 100)$. Esta operação é realizada com uma certa probabilidade regulada por um parâmetro do algoritmo, a taxa de mutação.

3.6. Evolução

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

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

Ou seja, o operador de evolução escolhe o indivíduo de melhor *fitness* entre x_i e c_i, mantendo sempre o tamanho da população a cada geração.

4. Modelo multi-populacional

Um AEMP é uma maneira simples, popular e efetiva de implementação de algoritmos evolucionários, tanto em dispositivos seriais como em computadores paralelos [Whitley et al., 1998]. A maior diferença entre um AEMP e um algoritmo evolutivo clássico é a separação da população em P ilhas, cada uma denominada como uma subpopulação. Cada subpopulação possui $n_i = \frac{N}{X}$ indivíduos, sendo que a população total das $|P|$ ilhas é igual a X. Uma ilha interage com outra utilizando um operador de migração, espalhando soluções locais por toda a população. Sem este operador de migração, um AEMP não é nada mais que diversas execuções separadas de um algoritmo evolutivo clássico [Skolicki e De Jong, 2005].

Além dos parâmetros utilizados por algoritmos evolutivos tradicionais, um AEMP possui a taxa de migração r, o tamanho da migração m e o número de ilhas P, sendo que r representa o intervalo entre dois processos de migração e m denota o número de indivíduos que migram a cada r gerações. Todos os três parâmetros adicionais tem grande influência na convergência de um AEMP [Skolicki e De Jong, 2005; Tomassini, 2005]. Portanto, eles devem ser cuidadosamente calibrados em conjunto com os parâmetros dos algoritmos evolutivos implementados.

O Algoritmo 2 mostra o pseudo-código de um AEMP geral. Primeiro ele inicializa as ilhas $p_i \in P$ (Linhas 1 a 3). Em sequência, cada ilha evolui um total de r gerações (Linhas 5 a 9). Então, o operador de migração é aplicado (Linha 10). O algoritmo executa até que um critério de parada seja atendido (*loop* das linhas 4 a 11). Finalmente, a melhor solução dentre todas as subpopulações é selecionada e retornada (Linha 12) e o algoritmo é encerrado. Um AEMP pode facilmente ser implementado de forma distribuída ao realizar o processo evolutivo (linhas 5 a 9) de forma paralela. Detalhes desta implementação distribuída podem ser encontrados em Alba [2005].

Algorithm 2 Algoritmo Evolutivo Multi-Populacional

```
1: for i ← 0 to P do
2:     Inicializa população p_i
3: end for
4: while critério de parada não é atendido do
5:     for j ← 0 to r do
6:         for i ← 0 to P do
7:             Evolui uma geração da população p_i
8:         end for
9:     end for
10:    Migração
11: end while
12: return melhor indivíduo de todas as ilhas
```

4.1. Migração

A migração nas ilhas de um modelo de algoritmo genético é um processo em que indivíduos são trocados entre as subpopulações (ilhas) de a cada m_r. Essa troca permite a transferência de informação genética e diversidade entre as ilhas, promovendo a evolução global da população. O processo de migração pode ser realizado de diferentes maneiras [Sekaj, 2004; Tomassini, 2005; Ruciński et al., 2010]. A aordagem implementada em nosso trabalho é exemplificada na Figura 3. Nela podemos ver que todas as ilhas são conectadas entre si. A cada r gerações ocorre a migração, onde um total de m indivíduos migram entre as diferentes ilhas. Estes indivíduos são selecionados em igual quantidade de cada uma das ilhas, sendo priorizados aqueles com melhor valor de *fitness*.

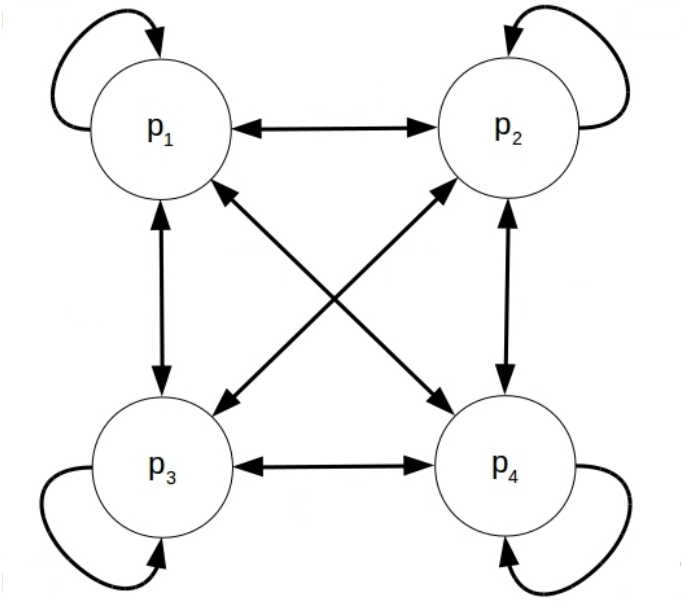

Figura 3: Operador de migração implementado

5. 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.	**Função**	**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

5.1. Otimização de parâmetros

A otimização de parâmetros foi realizada nas funções matemáticas $F2, F5$ 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. O principal precursor foi a ferramenta iRace [López-Ibáñez et al., 2016], que aplica o o método *Iterated Race* disponibilizado pelos autores da ferramenta e implementado na linguagem R. Este experimento ainda considerou um tempo limite de 10 segundos e 1.490.400 avaliações como critério de parada dos algoritmos.

A Tabela 2 exibe o experimento de otimização de parâmetros. Nesta tabela GA refere-se ao GA clássico implementado utilizando uma única população enquanto GAM refere-se ao GA multi-populacional implementado utilizando o modelo de ilhas. 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*.

5.2. Comparação dos algoritmos

A comparação dos algoritmos propostos foi realizada utilizando os parâmetros definidos na Seção 5.1. Assim como no experimento anterior, foi definido um tempo de 10 segundos

Tabela 2: Ajuste de parâmetros dos algoritmos testados

	Parâmetro	Intervalo de teste	Melhor
GA	Tamanho da População	$[30, 1.00 \times 10^4] \in \mathbb{Z}$	4777
	Taxa de Mutação	$[0, 100] \in \mathbb{Z}$	25
GAM	Tamanho da População	$[30, 1.00 \times 10^4] \in \mathbb{Z}$	108
	Taxa de Mutação	$[0, 100] \in \mathbb{Z}$	47
	P	$[1, 10] \in \mathbb{Z}$	5
	r	$[10, 300] \in \mathbb{Z}$	184
	m	$[1, 29] \in \mathbb{Z}$	29

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 o GAM. 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.

Tabela 3: Comparação do Algoritmo Genético Simples e Algoritmo Genético Multipopulacional

	GA		GAM	
	ME	SD	ME	SD
F1	1.22×10^9	2.57×10^8	$\mathbf{1.06 \times 10^6}$	6.61×10^5
F3	3.09×10^2	5.82×10^{-1}	$\mathbf{3.06 \times 10^2}$	4.36×10^{-1}
F4	1.73×10^3	1.39×10^2	$\mathbf{1.15 \times 10^3}$	1.05×10^2
F6	6.02×10^2	2.38×10^{-1}	$\mathbf{6.00 \times 10^2}$	2.47×10^{-2}
F7	7.08×10^2	1.92	$\mathbf{7.00 \times 10^2}$	2.55×10^{-2}
F8	8.55×10^2	2.59×10^1	$\mathbf{8.02 \times 10^2}$	2.85×10^{-1}
F9	9.04×10^2	1.43×10^{-1}	$\mathbf{9.03 \times 10^2}$	1.02×10^{-1}
F10	3.1×10^4	1.31×10^4	$\mathbf{1.67 \times 10^4}$	1.52×10^4
F11	$\mathbf{1.11 \times 10^3}$	1.08	$\mathbf{1.11 \times 10^3}$	1.13
F13	1.64×10^3	7.51	$\mathbf{1.62 \times 10^3}$	1.50
F14	1.60×10^3	1.82	$\mathbf{1.59 \times 10^3}$	2.07
F15	1.98×10^3	1.25×10^1	$\mathbf{1.92 \times 10^3}$	1.21×10^1

Como pode ser observado na Tabela 3, o GAM oteve resultados superiores ou tão bom quantos aos do GA em todas as funções avaliadas. A única função onde ambos os algoritmos obtiveram resultados semelhantes foi a F11. A grande vantagem do GAM sobre o GA foi a maior diversidade de sua população distribuída. Apesar da grande população otimizada para o GA, este ficou preso em ótimos locais com frequência, o que não possibilitou uma melhor convergência do método. Já as múltiplas populações do GAM possibilitou que o algoritmo não ficasse preso em ótimos locais e conseguisse explorar de forma mais efetiva o espaço de soluções das funções avaliadas.

6. Conclusões e trabalhos futuros

Este trabalho avaliou duas versões de um Algoritmo Genético. A primeira versão, denominada GA, é sua implementação clássica da literatura, enquanto a segunda versão, denominada GAM, implementa um Algoritmo Genético cuja população é subdividida em diversas ilhas. Experimentos computacionais mostraram que o GAM teve uma performance tão boa quanto ou melhor que a do GA em todas as funções de *benchmark* propostas.

Como trabalhos futuros, planeja-se desenvolver um algoritmo multi-populacional co evolucionário utilizando diversas meta-heurísticas populacionais, como o algoritmo de Evolução Diferencial e o Algoritmo de Enxame de Partículas. Tal algoritmo co-evolucionário utilizará um esquema de ilhas (ou subpopulações), de maneira semelhante ao recente artigo de **?**. 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

Alba, E. (2005). *Parallel metaheuristics: a new class of algorithms*, volume 47. John Wiley & Sons.

Das, A. K. e Pratihar, D. K. (2021). Solving engineering optimization problems using an improved real-coded genetic algorithm (irga) with directional mutation and crossover. *Soft Computing*, 25: 5455–5481.

Holland, J. H. (1975). Adaptation in natural and artificial systems. *University of Michigan Press*.

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

Katoch, S., Chauhan, S. S., e Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80:8091–8126.

Lässig, J. e Sudholt, D. (2013). Design and analysis of migration in parallel evolutionary algorithms. *Soft Computing*, 17(7):1121–1144.

Li, C., Yang, S., et al. (2008). An island based hybrid evolutionary algorithm for optimization. In *Proceedings of the International Conference on Simulated Evolution and Learning*, p. 180–189. Springer.

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.

Ruciński, M., Izzo, D., e Biscani, F. (2010). On the impact of the migration topology on the island model. *Parallel Computing*, 36(10):555–571.

Sekaj, I. (2004). Robust parallel genetic algorithms with re-initialisation. In *Parallel Problem Solving from Nature-PPSN VIII*, p. 411–419. Springer.

Skolicki, Z. e De Jong, K. (2005). The influence of migration sizes and intervals on island models. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, p. 1295–1302. ACM.

Tomassini, M. (2005). *Spatially structured evolutionary algorithms: artificial evolution in space and time*. Springer.

Whitley, D., Rana, S., e Heckendorn, R. B. (1998). The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7: 33–48.

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.