UNIVERSIDADE FEDERAL DE ALFENAS

Caio Eduardo Marcondes
Guilherme Augusto Gouveia
Vinicius Bueno Bastos

RELATÓRIO:

COMPARAÇÃO ENTRE ALGORITMOS GENÉTICOS

Alfenas/MG

2023

RELATÓRIO

COMPARAÇÃO ENTRE ALGORITMOS GENÉTICOS

Relatório apresentando a comparação entre dois algoritmos que utilizam da metaheurística evolutiva, em configurações específicas para testes, como parte do trabalho final de Heurísticas e Metaheurísticas pela Universidade Federal de Alfenas.

Professor:
Prof. Dr. Iago Augusto de Carvalho

Alfenas/MG

2023

RESUMO

Este trabalho tem como objetivo comparar diferentes algoritmos genéticos (AGs) em termos de eficiência na resolução de problemas de otimização. Para isso, foram selecionados dois algoritmos genéticos: o algoritmo genético clássico (AGC) e o algoritmo genético island-model híbrido (AGIMH).

O estudo foi realizado por meio de testes em diversos problemas de otimização de diferentes complexidades. Foram utilizadas métricas como número de avaliações da função objetivo, tempo de execução e qualidade da solução encontrada para avaliar o desempenho de cada algoritmo.

Os resultados obtidos mostraram que o AGIMH apresentou o melhor desempenho na maioria dos problemas testados, superando o AGC. Além disso, o AGIMH apresentou um tempo de execução menor e uma qualidade de solução superior, em comparação com os outros dois algoritmos.

De forma geral, os resultados sugerem que o uso do island-model híbrido pode ser benéfico para melhorar o desempenho dos algoritmos genéticos em problemas de otimização. No entanto, é importante destacar que os resultados obtidos podem variar dependendo do tipo de problema a ser resolvido e da configuração dos parâmetros do algoritmo genético.

Palavras-chave: Heurística, Metaheurística, Algoritmo Genético, Algoritmo evolutivo, Island Model Híbrido, Otimização.

ABSTRACT

This work aims to compare different genetic algorithms (GAs) in terms of efficiency in solving optimization problems. For this, two genetic algorithms were selected: the classical genetic algorithm (CGA) and the island-model genetic algorithm hybrid (AGIMH).

The study was carried out through tests on several optimization problems of different complexities. Metrics such as the number of evaluations of the objective function, execution time and quality of the solution found were used to evaluate the performance of each algorithm.

The obtained results showed that the AGIMH presented the best performance in most of the problems tested, surpassing the AGC. In addition, AGIMH presented a shorter execution time and a superior solution quality, compared to the other two algorithms.

In general, the results suggest that the use of the island-model hybrid can be beneficial to improve the performance of genetic algorithms in optimization problems. However, it is important to highlight that the results obtained may vary depending on the type of problem to be solved and the configuration of the parameters of the genetic algorithm.

Keywords: Heuristics, Metaheuristics, Genetic Algorithm, Evolutionary Algorithm, Island Model Hybrid, Optimization.

LISTA DE FIGURAS

Figura 1	**Gráfico da Função 1……..**	**26**
Figura 2	**Gráfico da Função 2……..**	**26**
Figura 3	**Gráfico da Função 3……..**	**27**
Figura 4	**Gráfico da Função 4……..**	**27**
Figura 5	**Gráfico da Função 5……..**	**28**
Figura 6	**Gráfico da Função 6……..**	**28**
Figura 7	**Gráfico da Função 7……..**	**29**
Figura 8	**Gráfico da Função 8……..**	**29**
Figura 9	**Gráfico da Função 9……..**	**30**
Figura 10	**Gráfico da Função 10……**	**30**
Figura 11	**Gráfico da Função 11……**	**31**
Figura 12	**Gráfico da Função 12……**	**31**
Figura 13	**Gráfico da Função 13……**	**32**
Figura 14	**Gráfico da Função 14……**	**32**
Figura 15	**Gráfico da Função 15……**	**33**

SUMÁRIO

1 INTRODUÇÃO..... 9

2 DESENVOLVIMENTO..... 10

2.1 CONCEITOS PRELIMINARES……..... 10

2.1.1 Heurísticas e Metaheurísticas…….. 10

2.1.2 Algoritmo genético clássico……. 11

2.1.3 Algoritmo genético multi populacional híbrido……. 12

2.1.4 Otimização de funções reais…… 13

2.2 ALGORITMOS DESENVOLVIDOS**……..... 15**

2.2.1 Algoritmo genético clássico……..... 16

2.2.2 Algoritmo genético multi populacional híbrido…….. 17

2.2.3 Operadores de cruzamento…….. 19

2.2.3.1 Operador K-Point……..…….. 19

2.2.3.2 Operador Média Aritmética…… 20

2.2.3.3 Operador Média Geométrica……. 21

2.2.3.4 Operador Blend…… 21

2.2.3.5 Operador Flat……... 21

2.2.3.6 Operador Metade…… 22

2.2.4 Topologia de comunicação entre populações…… 22

2.2.5 Otimização de parâmetros…… 23

2.3 ANÁLISE DE RESULTADOS..... 24

2.3.1 Gráficos…….. 26

2.3.2 Configurações do dispositivo utilizado…… 33

3 CONCLUSÃO 34

4 REFERÊNCIAS 35

1 INTRODUÇÃO

A otimização de problemas é um desafio que pode ser enfrentado por meio de diversas abordagens, como as heurísticas e as metaheurísticas. Entre as metaheurísticas mais utilizadas, destacam-se os algoritmos genéticos (AGs), que são inspirados nos princípios da evolução biológica. Os AGs são capazes de encontrar soluções aproximadas para problemas complexos de otimização, e uma das vantagens dessa técnica é a possibilidade de trabalhar com diversas variantes do algoritmo básico, que podem ser adaptadas para cada problema específico.

Nesse contexto, este trabalho tem como objetivo comparar dois algoritmos genéticos: o algoritmo genético clássico (AGC) e o algoritmo genético island-model híbrido (AGIMH), em termos de eficiência na resolução de problemas de otimização. O AGC é o algoritmo genético mais simples e clássico, enquanto o AGIMH é uma variante mais recente que apresenta uma estrutura de população mais complexa, dividindo a população em subpopulações denominadas ilhas.

Serão utilizadas métricas como número de avaliações da função objetivo, tempo de execução e qualidade da solução encontrada para avaliar o desempenho de cada algoritmo. Serão realizados testes em diversos problemas de otimização de diferentes complexidades, a fim de avaliar a eficiência dos algoritmos em diferentes cenários. Com base nos resultados obtidos, será possível verificar se o uso do island-model pode ser benéfico para melhorar o desempenho dos algoritmos genéticos em problemas de otimização.

Este trabalho contribui para a análise comparativa de diferentes variantes dos algoritmos genéticos e oferece informações relevantes para a seleção da melhor abordagem em problemas de otimização. Além disso, os resultados obtidos podem fornecer insights para a configuração de parâmetros dos algoritmos genéticos em diferentes cenários.

2 DESENVOLVIMENTO

2.1 CONCEITOS PRELIMINARES

Antes de apresentar os detalhes dos algoritmos, genético clássico e island-model híbrido, é importante discutir alguns conceitos preliminares relevantes para a compreensão dessas técnicas de otimização.

2.1.1 Heurísticas e Metaheurísticas

Heurísticas e metaheurísticas são técnicas de otimização usadas para encontrar soluções aproximadas para problemas complexos. As heurísticas são algoritmos que utilizam regras empíricas ou conhecimento especializado para encontrar uma solução viável. Por exemplo, em um problema de roteamento de veículos, uma heurística pode ser a estratégia de sempre escolher a rota mais curta entre dois pontos. Embora essa estratégia possa não levar à solução ótima, ela pode ser suficiente em muitos casos.

Já as metaheurísticas são algoritmos de otimização que geralmente não usam informações específicas do problema, mas sim uma estratégia geral para explorar o espaço de busca de soluções.

Por exemplo, o algoritmo genético é uma metaheurística que utiliza princípios inspirados na genética para explorar o espaço de soluções. Ele utiliza operadores de seleção, cruzamento e mutação para gerar novas soluções a partir de soluções existentes e busca encontrar a solução ótima.

Em resumo, podemos dizer que, enquanto as heurísticas são algoritmos simples que utilizam regras empíricas ou conhecimento especializado para encontrar uma solução viável, as metaheurísticas são algoritmos mais complexos que utilizam uma estratégia geral para explorar o espaço de soluções e encontrar soluções ótimas ou próximas ao ótimo. Ambas as técnicas têm suas vantagens e limitações, e a escolha da técnica mais adequada depende do problema em questão e das restrições de tempo e recursos.

2.1.2 Algoritmo genético clássico

Sobre o algoritmo genético clássico, podemos destacar, em primeiro lugar, que eles são uma classe de algoritmos evolutivos, que se baseiam nos princípios da seleção natural e da reprodução para encontrar soluções aproximadas para problemas de otimização. Eles simulam o processo de seleção natural, reprodução e mutação, aplicado a uma população de indivíduos que representam possíveis soluções para o problema.

O primeiro passo em um algoritmo genético é a geração aleatória de uma população inicial de indivíduos. Em seguida, é realizada uma avaliação da qualidade de cada indivíduo em relação ao problema de otimização, por meio de uma função objetivo. A partir daí, os indivíduos são selecionados para reprodução, com base em critérios que levam em consideração a qualidade da solução que eles representam.

A reprodução é realizada por meio de operadores genéticos, que podem incluir a recombinação de informações genéticas entre dois indivíduos (crossover) e a introdução de mutações aleatórias.

A partir dos indivíduos selecionados e das operações genéticas realizadas, é gerada uma nova população, que passa por um processo de avaliação e seleção novamente. O processo é repetido por um número definido de vezes, ou até que um critério de parada seja alcançado.

Por fim, é importante mencionar que os algoritmos genéticos possuem diversos parâmetros que podem ser ajustados para melhorar o desempenho do algoritmo em diferentes problemas de otimização. Alguns exemplos de parâmetros incluem a taxa de mutação, o tamanho da população, o número de gerações, o tipo de seleção, entre outros. A escolha desses parâmetros pode ser uma tarefa complexa, e pode influenciar significativamente o desempenho do algoritmo.

2.1.3 Algoritmo genético multi populacional híbrido

Já o algoritmo genético island-model híbrido (AGIMH) é uma variação do algoritmo genético clássico que apresenta algumas diferenças importantes em relação ao modelo tradicional.

Em primeiro lugar, no AGIMH a população é dividida em diversas subpopulações (ilhas), que evoluem de forma independente. Cada subpopulação tem sua própria dinâmica evolutiva, ou seja, pode apresentar diferentes taxas de mutação, recombinação e seleção. Essa abordagem permite que o algoritmo explore diferentes regiões do espaço de busca de forma simultânea, aumentando a chance de encontrar soluções ótimas.

Outra diferença importante é a existência de um mecanismo de migração periódica de indivíduos entre as subpopulações. Em intervalos regulares de tempo, alguns indivíduos são transferidos entre as ilhas, com o objetivo de promover a diversidade genética e evitar a estagnação evolutiva em uma determinada subpopulação.

A troca de indivíduos permite que soluções promissoras sejam propagadas entre as ilhas, acelerando o processo de convergência para uma solução ótima.

Além disso, o AGIMH apresenta alguns parâmetros adicionais em relação ao algoritmo genético clássico, como o número de ilhas, a taxa de migração e o intervalo de tempo para migração. Esses parâmetros podem ser ajustados de forma a otimizar o desempenho do algoritmo para um determinado problema de otimização.

Em resumo, as principais diferenças do algoritmo genético island-model híbrido em relação ao algoritmo genético clássico são: a divisão da população em múltiplas subpopulações, o mecanismo de migração de indivíduos entre as subpopulações e a possibilidade de ajuste de parâmetros para otimizar o desempenho do algoritmo.

Essas diferenças permitem que o AGIMH seja mais eficiente na busca de soluções ótimas em problemas de otimização complexos.

2.2.4 Otimização de funções reais

A otimização de funções reais é um problema clássico em matemática e ciência da computação, que envolve a busca pela melhor solução para uma função objetiva em um espaço de busca. A função objetivo pode representar uma variedade de problemas em diferentes áreas, como engenharia, economia, ciência de dados, entre outras.

O objetivo da otimização é encontrar a solução que minimize ou maximize a função objetivo, dependendo do contexto do problema. Para encontrar essa solução, é necessário explorar o espaço de busca, que é o conjunto de todas as possíveis soluções. No entanto, para problemas complexos, o espaço de busca pode ser enorme e a busca exaustiva por uma solução ótima pode ser impraticável ou impossível.

Por isso, as técnicas de otimização são muito importantes, e incluem tanto métodos exatos quanto métodos aproximados. Os métodos exatos utilizam algoritmos determinísticos que garantem encontrar a solução ótima, mas podem ser impraticáveis para problemas de grande porte. Já os métodos aproximados, como as metaheurísticas, oferecem soluções viáveis em tempo razoável, mas não garantem a solução ótima.

Em resumo, a otimização de funções reais é um problema fundamental em matemática e ciência da computação, e é crucial para resolver problemas em diversas áreas. As técnicas de otimização são essenciais para encontrar soluções viáveis para problemas complexos, e existem métodos exatos e aproximados que podem ser aplicados dependendo do contexto do problema.

Para nosso cenário, utilizaremos as funções definidas pela CEC 2015 descritas pela Tabela 1, nessa tabela há uma breve explicação das 15 funções que usaremos com o objetivo para minimização:

Tabela 1 Funções de teste de benchmark CEC 2015.

No.	Functions	Funções básicas relacionadas	Dim	fmin
CEC-1	Rotated Bent Cigar Function	Bent Cigar Function	10, 30	100
CEC-2	Rotated Discus Function	Discus Function	10, 30	200
CEC-3	Shifted and Rotated Weierstrass Function	Weierstrass Function	10, 30	300
CEC-4	Shifted and Rotated Schwefel's Function	Schwefel's Function	10, 30	400
CEC-5	Shifted and Rotated Katsuura Function	Katsuura Function	10, 30	500
CEC-6	Shifted and Rotated HappyCat Function	HappyCat Function	10, 30	600
CEC-7	Shifted and Rotated HGBat Function	HGBat Function	10, 30	700
CEC-8	Shifted and Rotated Expanded	Griewank's Function Rosenbrock's Function	10, 30	800
CEC-9	Shifted and Rotated Expanded Scaer's F6 Function	Expanded Scaer's F6 Function	10, 30	900
CEC-10	Hybrid Function 1 (N = 3)	Schwefel's Function Rastrigin's Function High Conditioned Elliptic Function	10, 30	1000
CEC-11	Hybrid Function 2 (N = 4)	Griewank's Function Weierstrass Function Rosenbrock's Function Scaer's F6 Function	10, 30	1100
CEC-12	Hybrid Function 3 (N = 5)	Katsuura Function HappyCat Function Expanded Griewank's plus Rosenbrock's Function Schwefel's Function Ackley's Function	10, 30	1200
CEC-13	Composition Function 1 (N = 5)	Rosenbrock's Function High Conditioned Elliptic Function Bent Cigar Function Discus Function High Conditioned Elliptic Function	10, 30	1300
CEC-14	Composition Function 2 (N = 3)	Schwefel's Function Rastrigin's Function High Conditioned Elliptic Function	10, 30	1400
CEC-15	Composition Function 3 (N = 5)	HGBat Function Rastrigin's Function Schwefel's Function Weierstrass Function High Conditioned Elliptic Function	10, 30	1500

2.2 ALGORITMOS DESENVOLVIDOS

O desenvolvimento de ambos os algoritmos teve como base o livro "Spatially structured evolutionary algorithms: artificial evolution in space and time", escrito por Marco Tomassini e publicado pela Springer Science & Business Media em 2006.

Este livro é um trabalho fundamental no campo da computação evolutiva, especialmente na área de algoritmos genéticos com estrutura espacial e temporal. Ele tem uma abordagem inovadora para o estudo da evolução artificial, onde o espaço e o tempo são incorporados na modelagem de algoritmos genéticos, permitindo assim a exploração de uma variedade de processos complexos.

O livro de Tomassini destaca a importância de considerar a estrutura espacial e temporal nos algoritmos genéticos. Ele discute como os ambientes espaciais e temporais influenciam a evolução, e como essas influências podem ser incorporadas nos modelos de algoritmos genéticos. Tomassini também explora a implementação desses modelos em aplicações do mundo real, como otimização de recursos, planejamento urbano, design de sistemas e muito mais.

No nosso trabalho, tomamos como base a construção das ideias do autor para fazermos nossos algoritmos genéticos clássicos e island-model híbrido. Utilizamos a estrutura espacial para organizar a população em subpopulações em diferentes ilhas, o que permite uma melhor exploração do espaço de busca e uma maior diversidade genética na população. Além disso, incorporamos a estrutura temporal na evolução, introduzindo a migração periódica entre as ilhas para compartilhar a informação genética entre elas.

Assim, o livro de Tomassini foi uma fonte importante de inspiração e orientação para o nosso trabalho. Suas ideias nos permitiram construir um modelo de algoritmo genético híbrido que é altamente eficaz na resolução de problemas complexos de otimização, e nos ajudou a alcançar resultados satisfatórios em várias funções reais multidimensionais.

Em resumo, o livro "Spatially structured evolutionary algorithms: artificial evolution in space and time" é uma obra de referência na área de algoritmos genéticos com estrutura espacial e temporal. Suas idéias e insights são extremamente valiosos para todos os interessados em computação evolutiva e em resolver problemas de otimização complexos.

2.2.1 Algoritmo Genético Clássico

Nosso algoritmo genético clássico não funciona muito diferente do que foi descrito na introdução inicial dada para este algoritmo. Segue abaixo um pseudocódigo do mesmo:

```
1   while number generations do
2       evaluate(population)
3       for number childs do
4           SelectParents()
5           child <- CrossOver()
6           worstParent <- &getWorstParent()
7           worstParent <- child
8           if child in population then
9               ForcedMutation(child)
10          Mutation()
11      end for
12      Selection()
13  end while
14
```

Cada indivíduo da população é representado pela seguinte struct:

```
typedef struct individue_
{
    double *chromosome;
    double fitness;
} individue;
```

A grande diferença deste algoritmo para os tradicionais está na função que ForcedMutation, ela é responsável por realizar uma mutação forçada no produto do cruzamento caso seu fitness já esteja na população base

2.2.2 Algoritmo Genético Multi-Populacional Híbrido

O Algoritmo genético multi-populacional híbrido implementado segue o pseudo-código abaixo, porém as rotinas: FixAndOptimize, SpreadIfImproved e IncreaseWindowSizeIfNotImproved não são utilizadas em virtude dos objetivos de nosso trabalho:

```
function HMPGA(populations)
    populations <- GenerateIslands()
    while Time limit not reached do
        for each population in populations do
            while number generations do
                evaluate(population)
                for number childs do
                    SelectParents()
                    child <- CrossOver()
                    worstParent <- &getWorstParent()
                    worstParent <- child
                    Mutation()
                end for
                Selection()
            end while
        end while
        Migrate()
    end while
end function
```

A grande diferença entre o código clássico e este está na existência das estruturas que chamamos de ilhas/population em nosso código, descrita pela definição de tipo abaixo:

```
typedef struct population_
{
    individue *individues;
    int size;
    struct population_ **neighbours;
    int crossover;

} population;
```

Dentre os atributos dessa struct, temos:

- individues: Vetor que contém os indivíduos pertencentes a população
- size: tamanho da população
- neightbours: Populações vizinhas que serão utilizadas para executar a migração.
- crossover: Operador de cruzamento que será utilizado, como descrito em detalhes na seção x.x.x.

Uma outra grande diferença está na implementação da função migrate, principal componente deste tipo de algoritmo e que se ausenta do anterior. Veja abaixo um trecho do código:

```c
void migrate(population *populations, int island_size, int population_size, int dimension, domain domain_function)
{
    DEBUG(printf("\nmigrate\n"););
    population *vizinho;

    for (int i = 0; i < island_size; i++)
    {
        DEBUG(printf("Populacao %d\n", i));
        population current_island = populations[i];
        individue *population = current_island.individues;
        individue *melhor_individuo_da_populacao = get_best_of_population(population, population_size);
        DEBUG(printf("Melhor individuo da populacao %d: %lf\n", i, melhor_individuo_da_populacao->fitness););
        for (int j = 0; j < 4; j++)
        {
            vizinho = current_island.neighbours[j];
            if (vizinho == NULL)
                continue;
            individue *neighbour_population = vizinho->individues;
            individue *pior_indivuduo_do_vizinho = get_worst_of_population(neighbour_population, population_size);
            if (melhor_individuo_da_populacao->fitness > pior_indivuduo_do_vizinho->fitness)
            {
                individue *new_worst = generate_population(1, dimension, domain_function);
                *pior_indivuduo_do_vizinho = *new_worst;
            }
            else
                *pior_indivuduo_do_vizinho = *melhor_individuo_da_populacao;
        }
    }
}
```

Basicamente, o que está função faz é o seguinte, dado uma população p_i, o melhor indivíduo desta população b_p é extraído.

Depois para cada vizinho v_i definido pelo argumento neighbours, onde $neighbours = \{v_i \mid 0 \leq i < 4, i \in Z\}$, é extraído o pior indivíduo desse vizinho r_v e uma comparação é feita:

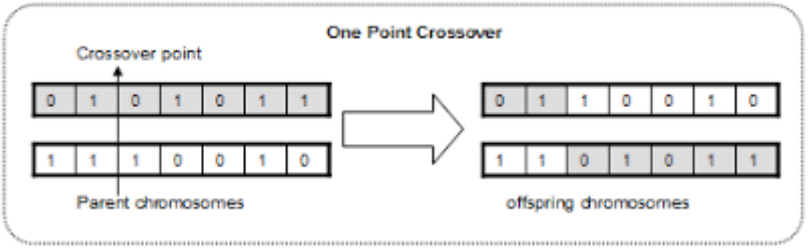
“ Se b_p for melhor que v_i, então no lugar de v_i é colocado b_p, caso contrário um indivíduo é gerado aleatoriamente é colocado no lugar de v_i ”

2.2.3 Operadores de cruzamento

Cada população possui um operador de cruzamento atribuído aleatoriamente dentre um total de 6 operadores definidos utilizando o atributo crossover definido no tipo population. Vejamos a seguir quais são estes operadores:

2.2.3.1 Operador K-Point

O operador de cruzamento "K-Point operator" é um dos muitos operadores de cruzamento utilizados em algoritmos genéticos para criar novas soluções a partir da combinação de soluções existentes. O "K-Point operator" é um operador de cruzamento que combina elementos de duas soluções "pais" (ou cromossomos) para gerar uma nova solução filho.



O funcionamento do "K-Point operator" é relativamente simples. Primeiro, dois pais são selecionados aleatoriamente na população. Em seguida, é escolhido um ponto de corte aleatório em cada pai, criando duas seções diferentes em cada pai. A partir daí, o operador de cruzamento seleciona uma seção de cada pai e as combina para formar um filho. Esse processo é repetido várias vezes, gerando uma nova população de filhos. O "K-Point operator" recebe esse nome devido à escolha aleatória do número de pontos de corte que são feitos em cada cromossomo, que é determinado pelo valor de K. Quando K=1, temos o operador de cruzamento de ponto único, enquanto K>1 representa o operador de cruzamento de múltiplos pontos.

Uma vantagem do "K-Point operator" é que ele é capaz de preservar algumas características dos pais, o que pode ser útil em alguns problemas de otimização, especialmente aqueles em que a solução ótima está perto da solução atual. No entanto, é importante notar que o "K-Point operator" não é garantido para produzir soluções melhores do que as soluções "pais", e sua eficácia pode variar dependendo do problema em questão.

Em resumo, o "K-Point operator" é um operador de cruzamento comumente usado em algoritmos genéticos, que combina elementos de dois pais para gerar uma nova solução filho. Ele é especialmente útil em problemas de otimização com restrições, onde a viabilidade das soluções é importante. Embora seja uma técnica relativamente simples, pode ser eficaz em certos problemas de otimização.

2.2.3.2 Operador Média Aritmética

O operador de cruzamento média, também conhecido como crossover aritmético, é outro operador de cruzamento comum utilizado em algoritmos evolutivos. Ele é frequentemente usado em algoritmos evolutivos multi-objetivo para combinar duas soluções candidatas (ou indivíduos) diferentes em uma nova solução intermediária que é uma média ponderada das duas soluções originais.

O processo de cruzamento média começa selecionando dois indivíduos da população de soluções candidatas. Em seguida, para cada gene (ou parâmetro) no indivíduo, um novo valor é calculado como a média ponderada dos valores dos dois pais. O peso de cada pai na média pode ser determinado por um valor aleatório gerado durante o processo de cruzamento. O valor do peso normalmente é escolhido aleatoriamente entre 0 e 1, e a soma dos pesos é igual a 1.

Por exemplo, suponha que temos dois indivíduos pais, A e B, e cada indivíduo tem dois genes, x_i e y_i. Dado os pesos p_1 e p_2 onde $p_1 + p_2 = 1$, o cromosso c_i do novo indivíduo C é definido da seguinte maneira:

$$c_i = p_1 \cdot a_i + p_2 \cdot b_i$$

2.2.3.3 Operador Média Geometrica

O operador de média geométrica é parecido com o de média, com uma óbvia alteração na fórmula de cálculo, dado os cromossomos a_i e b_i dos indivíduos A e B, respectivamente, o que fazemos é calcular o novo cromossomo c_i do filho gerado C a partir da média geométrica dos valores selecionados. A fórmula geral é a seguinte:

$$c_i = \sqrt{a_i * b_i}$$

2.2.3.4 Operador Blend

O operador blend trabalha com a existência de um parâmetro β, sendo que, dado os cromossomos a_i e b_i dos indivíduos A e B, respectivamente, o que fazemos é calcular o novo cromossomo c_i do filho gerado C a partir de uma média que leva em conta o parâmetro β dos valores selecionados, em uma busca esporádica pela literatura, encontra o número 0.125 como um bom valor para β, provavelmente definido por meio de artifícios estatísticos . A fórmula geral é a seguinte:

$$c_i = \beta \cdot a_i + (1 - \beta) \cdot b_i$$

2.2.3.5 Operador Flat

Dado os cromossomos a_i e b_i dos indivíduos A e B, respectivamente, o operador flat gera um valor aleatório para o cromossomo c_i que pertence ao intervalo $[a_i, b_i]$. Para isso uma função chamada *random_double* responsável por gerar valores do tipo double entre um determinado intervalo foi criada. Veja abaixo:

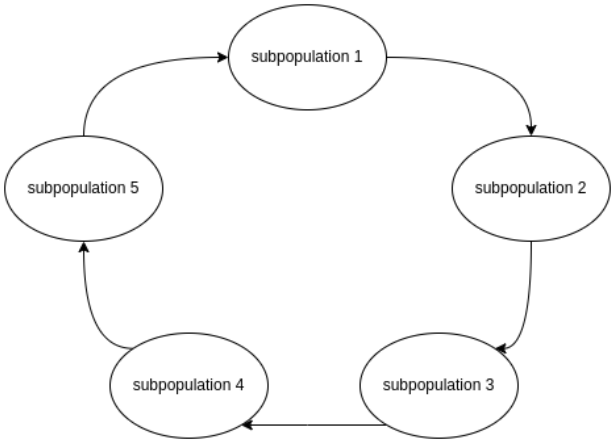
```
27  double random_double(double min, double max)
28  {
29      double random = min + (max - min) * (rand() / (double)RAND_MAX);
30      return random;
31  }
```

2.2.3.6 Operador Metade

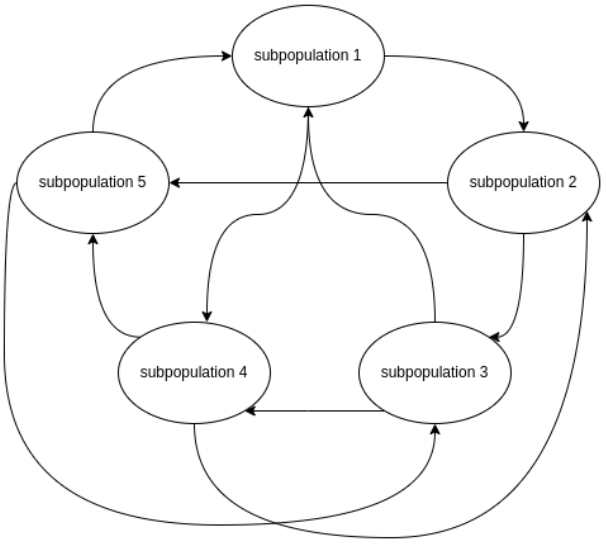
O operador metade é basicamente uma especialização do operador K-Point, onde o ponto de corte definido é exatamente a metade do número de cromossomos dos indivíduos.

2.2.4 Topologia de comunicação entre populações

A topologia escolhida foi um ring com algumas modificações quanto sua estrutura, enquanto uma topologia ring tradicional para 5 populações diferentes segue a seguinte estrutura:



De acordo com a nova topologia criada para resolver o nosso problema, a estrutura correspondente para 5 populações é a seguinte:



Essa topologia é definida no momento em que geramos cada ilha que irá compor nosso conjunto de populações. Fugindo de abstrações, o trecho de código é o seguinte:

```
populations[i].neighbours = calloc(4, sizeof(population *));
populations[i].neighbours[0] = &populations[(i + 1) % island_size];
populations[i].neighbours[1] = &populations[(i + 3) % island_size];
```

2.2.5 Otimização de parâmetros

A fim de se obter uma melhora nos resultados foi realizada a otimização de alguns dos parâmetros do AGIMH, implementando a ideia do algoritmo grid search.Os parâmetros otimizados foram: Island size, population size e generations size, onde os demais parâmetros foram mantidos os mesmos.

Como a ideia do algoritmo grid search é a partir de um conjunto de valores escolhidos gerar e testar todas as combinações possíveis, reduzindo o tamanho do grid a medida que resultados promissores são encontrados, foram escolhidos de maneira empírica os seguintes valores para cada parâmetro:

island size = {20, 15, 10};
population size = {30, 20, 5};
generations size = {300, 200, 150}.

Sendo assim, foi necessário selecionar algumas funções para realizar a otimização, onde as funções escolhidas foram as funções 14 e 15. Desse modo são testadas as diferentes combinações de parâmetros em cada função, onde obtemos os melhores valores de parâmetros que causaram uma melhora no resultado final de ambas as funções.

Na função de otimização foi adicionado um limite para tentativas que não apresentem melhora no resultado, sendo que quando esse limite é atingido é realizada uma redução no grid com os melhores valores de parâmetros atuais e o limite para tentativas é dobrado, e caso o limite seja atingido novamente e execução do programa é encerrada.

```
if (aux == 2)
    limite_tentativas = limite_tentativas * 2;
if (aux == 3)
    reduz_grid = 0;
int reduzir;
// pegando os melhores parametros para gerar um novo conjunto
island_size_aux[0] = *island_size;
population_size_aux[0] = *population_size;
num_generations_aux[0] = *num_generations;
// gerando novo conjunto
for (int v = 1; v <= num_valores; v++)
{
    reduzir = island_size_aux[v - 1] - 2 <= 0 ? 0 : 1;
    island_size_aux[v] = island_size_aux[v - 1] - reduzir;

    reduzir = population_size_aux[v - 1] - 3 <= 0 ? 0 : 2;
    population_size_aux[v] = population_size_aux[v - 1] - reduzir;

    reduzir = num_generations_aux[v - 1] - 20 <= 0 ? 0 : 20;
    num_generations_aux[v] = num_generations_aux[v - 1] - reduzir;
}
reduz_grid--;
```

Após a realização da otimização de parâmetros foram obtidos os seguintes valores:Island size = 10, population size = 3 e generations size = 300.Os parâmetros obtidos ao fim da otimização apresentaram uma melhora significativa no resultado comparado aos parâmetros testados empiricamente.

2.3 ANÁLISE DE RESULTADOS

A análise de resultados é uma etapa importante em qualquer projeto que envolve a avaliação de um modelo ou algoritmo. Ela consiste em avaliar o desempenho do modelo ou algoritmo em relação a um conjunto de métricas pré-definidas, com o objetivo de comparar diferentes modelos ou algoritmos e selecionar o melhor deles.

Uma vez definidas as métricas e o conjunto de dados, é possível avaliar o desempenho de cada modelo ou algoritmo. A análise de resultados pode envolver a comparação de diferentes algoritmos em termos de métricas de avaliação específicas, a visualização dos resultados por meio de gráficos ou tabelas, e a identificação de possíveis problemas ou limitações do modelo ou algoritmo.

Por fim, a análise de resultados deve ser interpretada de forma crítica e cuidadosa, levando em consideração as limitações e suposições subjacentes aos modelos ou algoritmos avaliados. É importante também validar os resultados por meio de testes adicionais, e garantir que o modelo ou algoritmo selecionado seja apropriado para o problema em questão e possa ser generalizado para novos dados.

Para o nosso trabalho, utilizamos as funções organizadas nessa tabela abaixo:

Tabela 2 Comparação de AGC básico e AGIMH de acordo com as funções de benchmark CEC2015.

	AGC				AGIMH			
	Std	Mean	Min	Max	Std	Mean	Min	Max
F1	2.968E+05	1.039E+08	3.837E+07	1.845E+08	1.858E+04	1.787E+04	1.123E+03	1.285E+05
F2	1.269E+03	8.407E+03	4.042E+03	1.103E+04	1.041E+03	1.447E+04	2.318E+02	4.248E+03
F3	5.066E-01	3.063E+02	3.047E+02	3.074E+02	0.823E+00	3.036E+02	3.015E+02	3.053E+02
F4	1.193E+02	1.415E+03	1.034E+03	1.658E+03	1.035E+01	4.152E+02	4.002E+02	4.560E+02
F5	1.978E-01	5.010E+02	5.004E+02	5.014E+02	0.144E+00	5.006E+02	5.001E+02	5.010E+02
F6	5.093E-02	6.003E+02	6.001E+02	6.004E+02	0.040E+00	6.002E+02	6.000E+02	6.002E+02
F7	8.722E-02	7.004E+02	7.002E+02	7.007E+02	0.047E+00	7.002E+02	7.000E+02	7.002E+02
F8	8.835E-01	8.056E+02	8.029E+02	8.080E+02	0.545E+00	8.016E+02	8.003E+02	8.032E+02
F9	1.224E-02	9.033E+02	9.028E+02	9.035E+02	0.305E+00	9.025E+02	9.015E+02	9.030E+02
F10	1.606E+04	3.127E+04	4.412E+03	9.089E+04	6.109E+02	2.298E+03	1.204E+03	5.326E+03
F11	2.188E+00	1.107E+03	1.104E+03	1.110E+03	0.576E+00	1.103E+03	1.101E+03	1.104E+03
F12	2.210E+01	1.348E+03	1.245E+03	1.360E+03	5.453E+00	1.228E+03	1.215E+03	1.252E+03
F13	1.322E+00	1.623E+03	1.619E+03	1.625E+03	1.936E+01	1.614E+03	1.421E+03	1.617E+03
F14	1.482E+00	1.595E+03	1.589E+03	1.598E+03	2.755E+00	1.586E+03	1.579E+03	1.593E+03
F15	2.091E+00	1.512E+03	1.507E+03	1.524E+03	1.971E+00	1.508E+03	1.503E+03	1.513E+03

2.3.1 Gráficos

Nos gráficos abaixo temos os resultados obtidos a partir do algoritmo genético clássico para cada função estudada.

Para F1 temos:

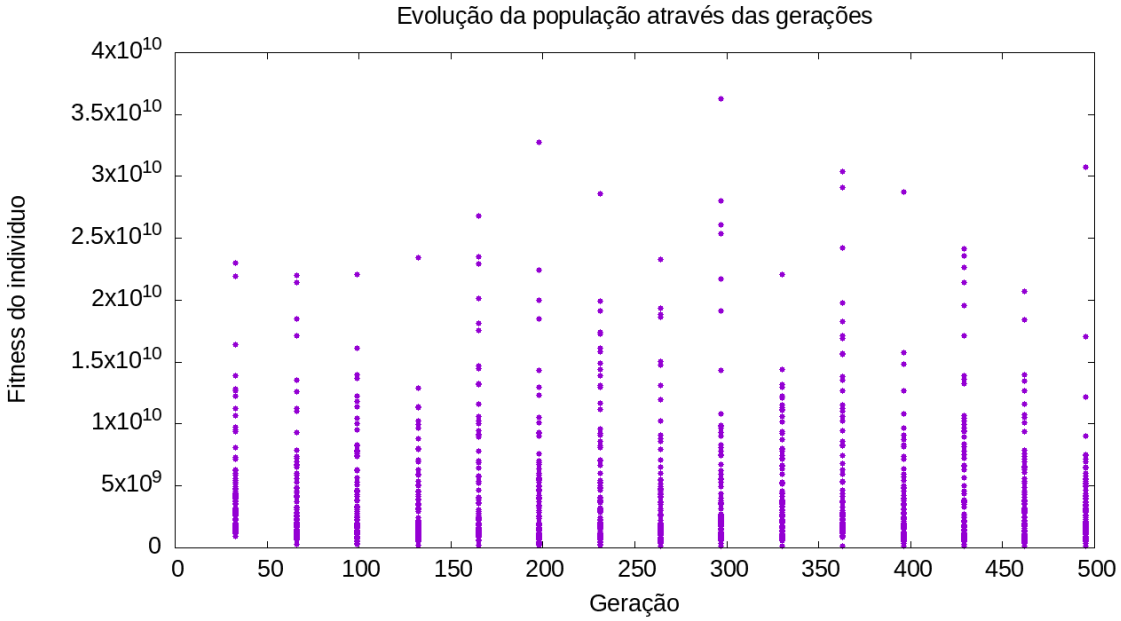


Gráfico da Função 1

Para F2 temos:

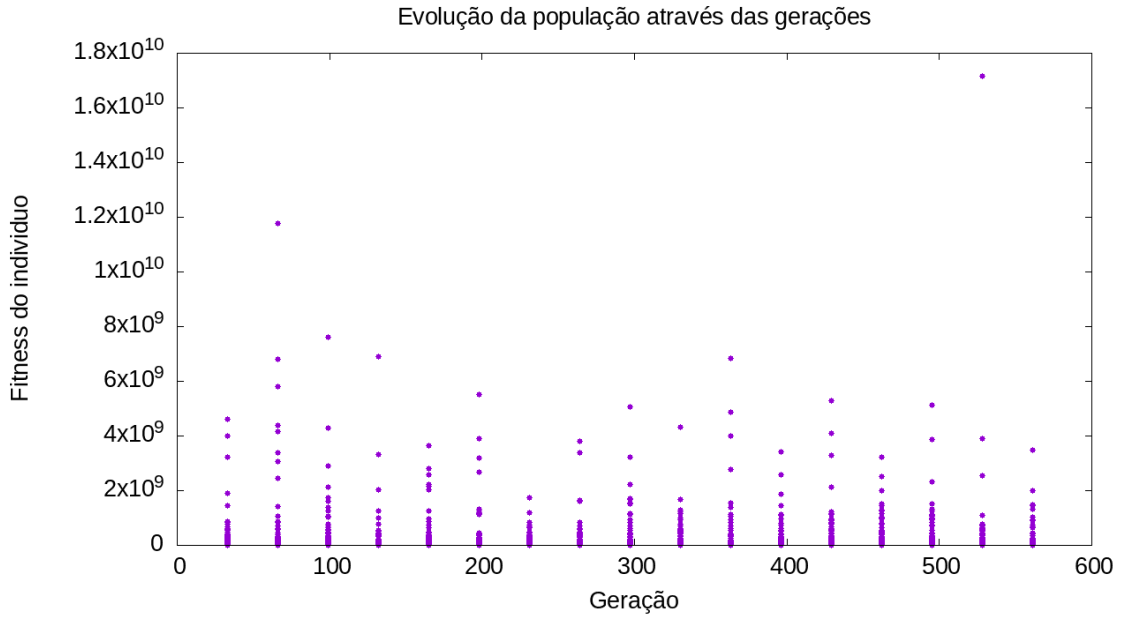


Gráfico da Função 2

Para F3 temos:

Gráfico da Função 3

Para F4 temos:

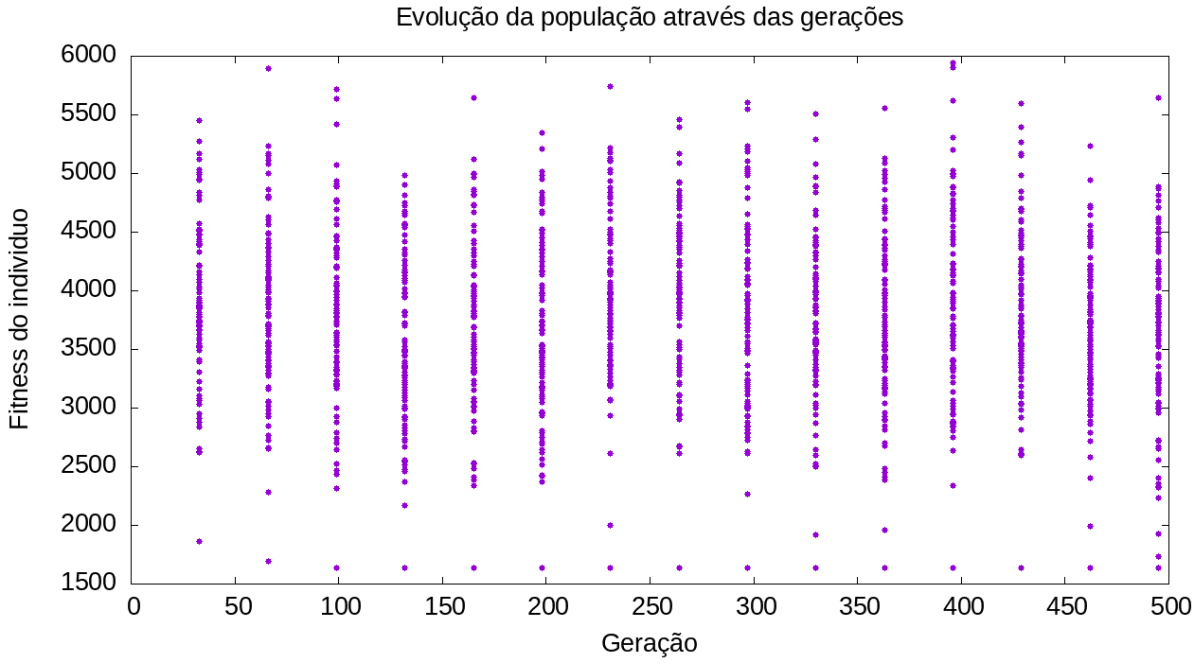


Gráfico da Função 4

Para F5 temos:

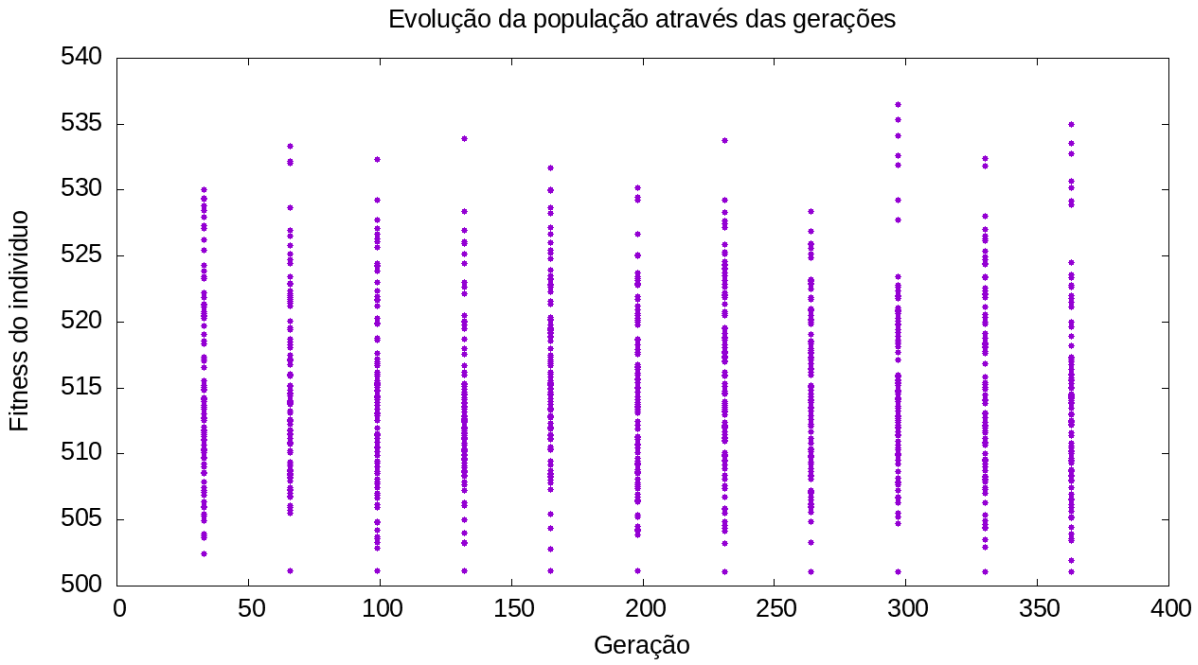


Gráfico da Função 5

Para F6 temos:

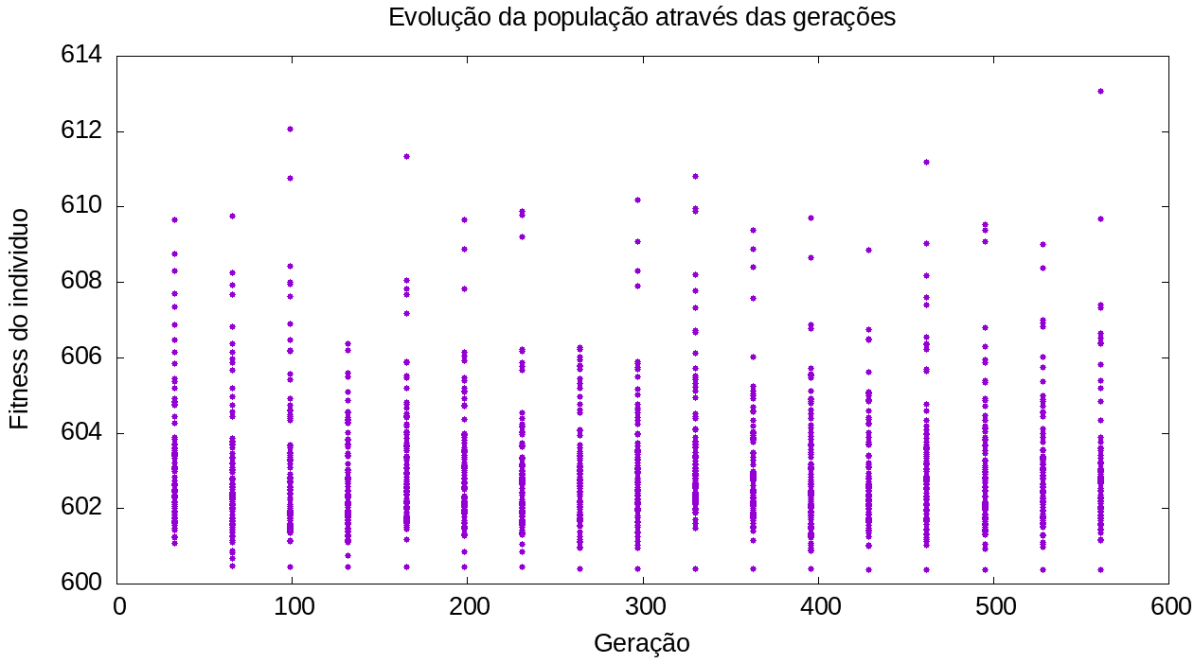


Gráfico da Função 6

Para F7 temos:

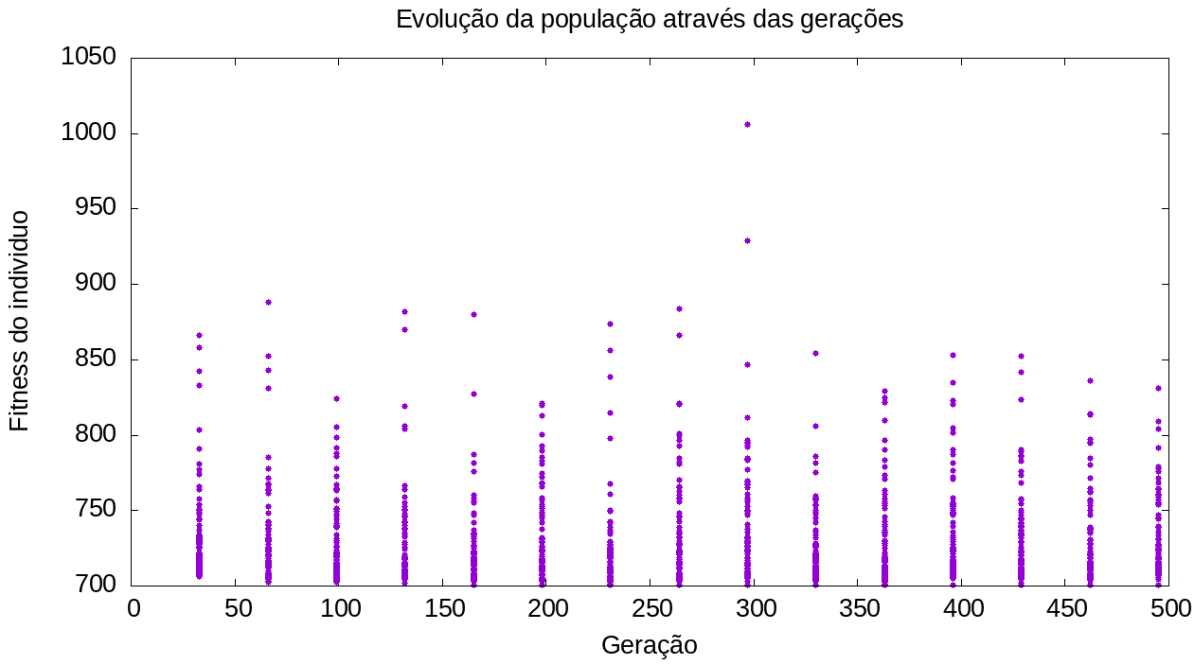


Gráfico da Função 7

Para F8 temos:

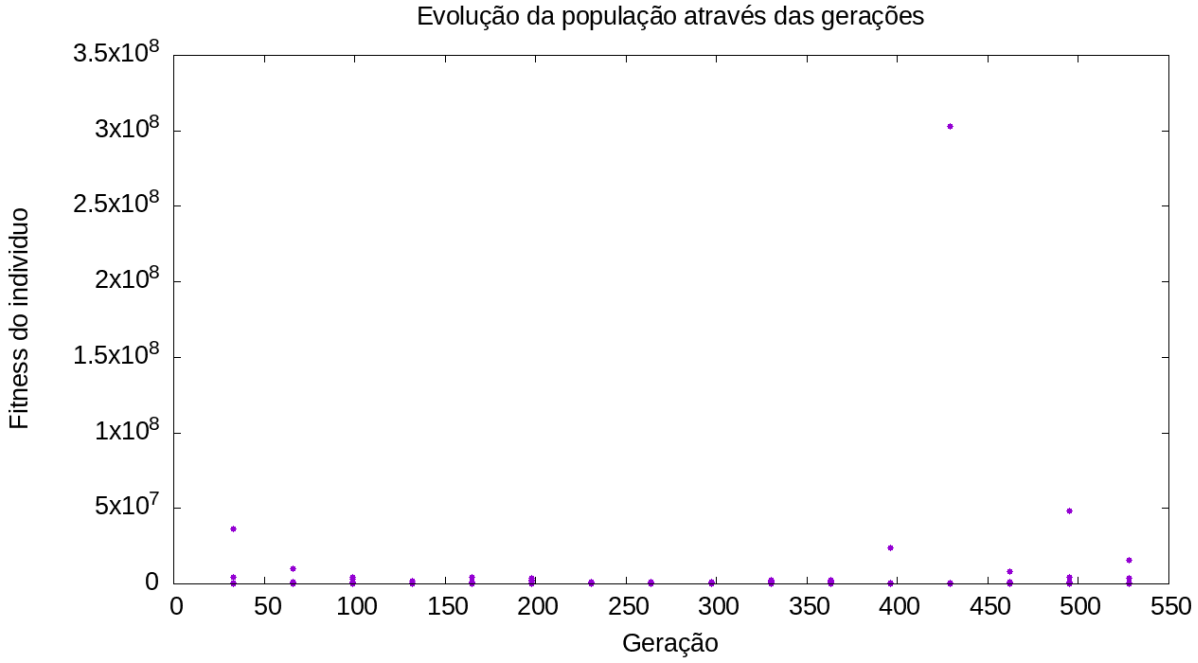


Gráfico da Função 8

Para F9 temos:

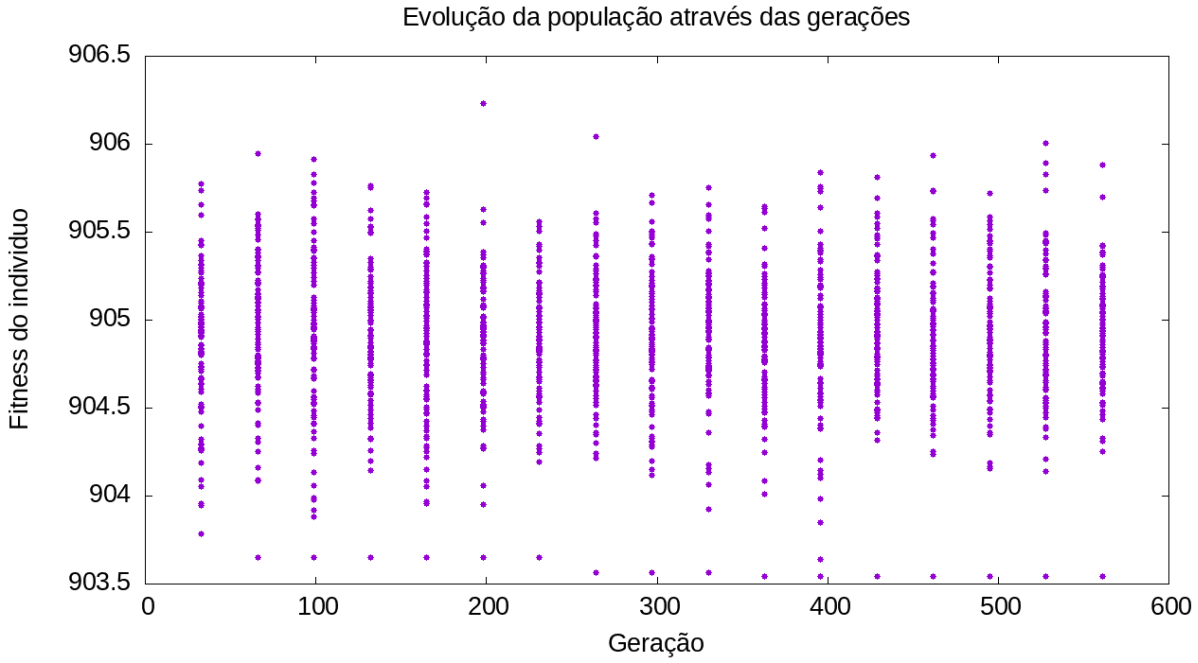


Gráfico da Função 9

Para F10 temos:

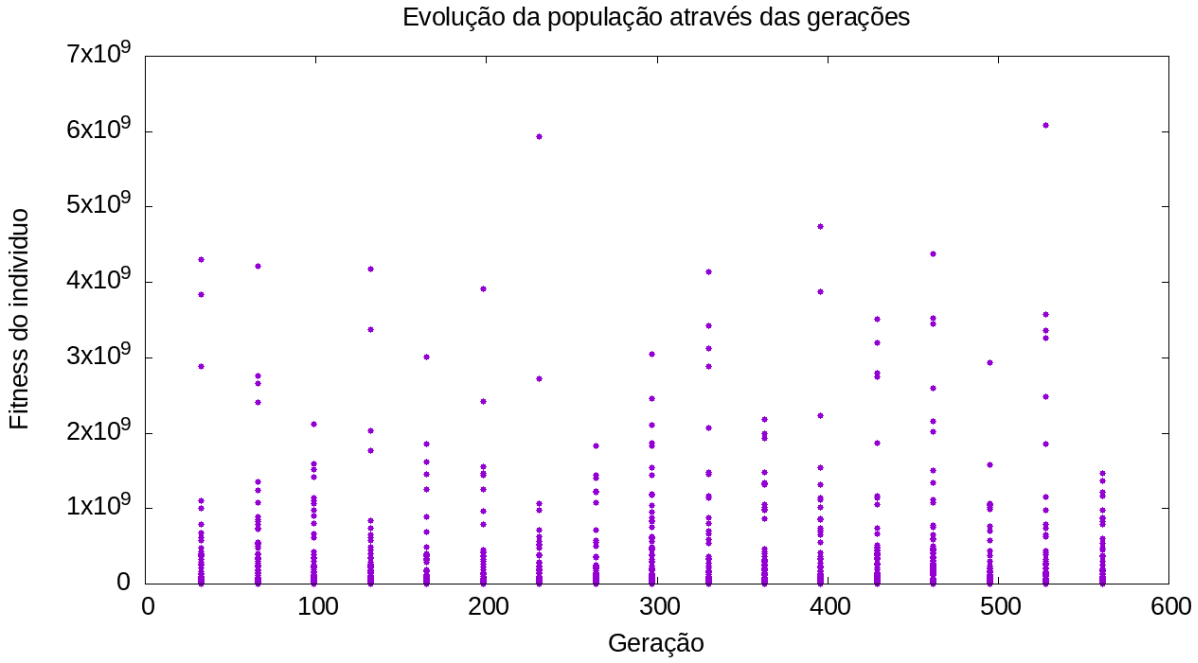


Gráfico da Função 10

Para F11 temos:

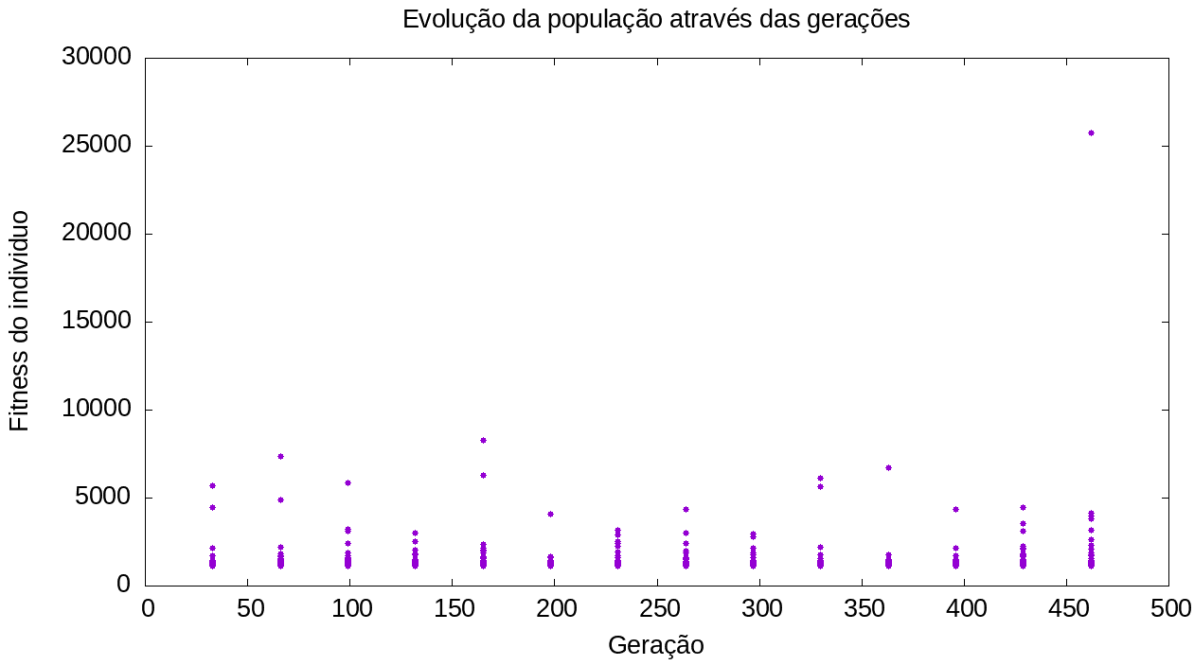


Gráfico da Função 11

Para F12 temos:

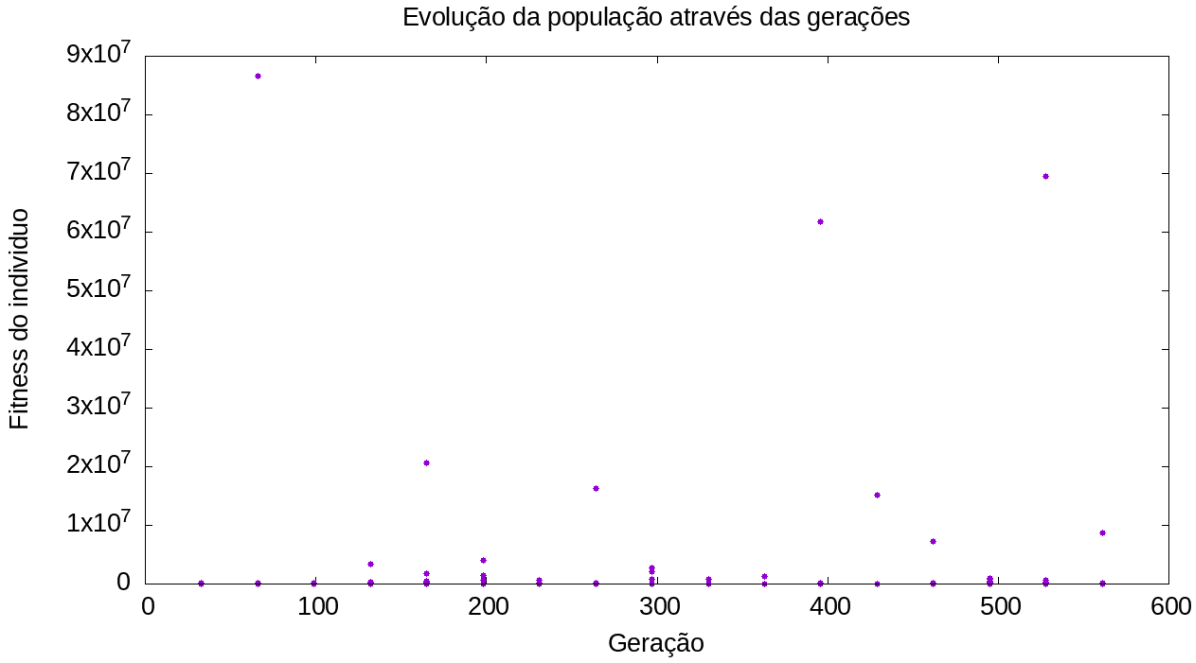


Gráfico da Função 12

Para F13 temos:

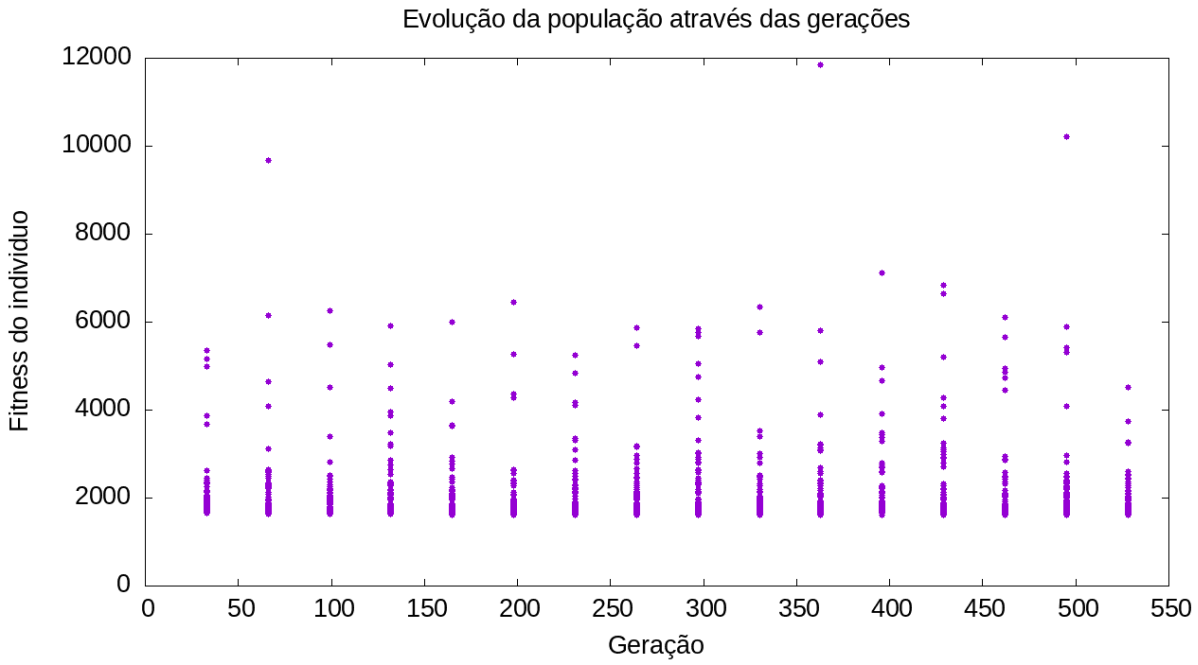


Gráfico da Função 13

Para F14 temos:

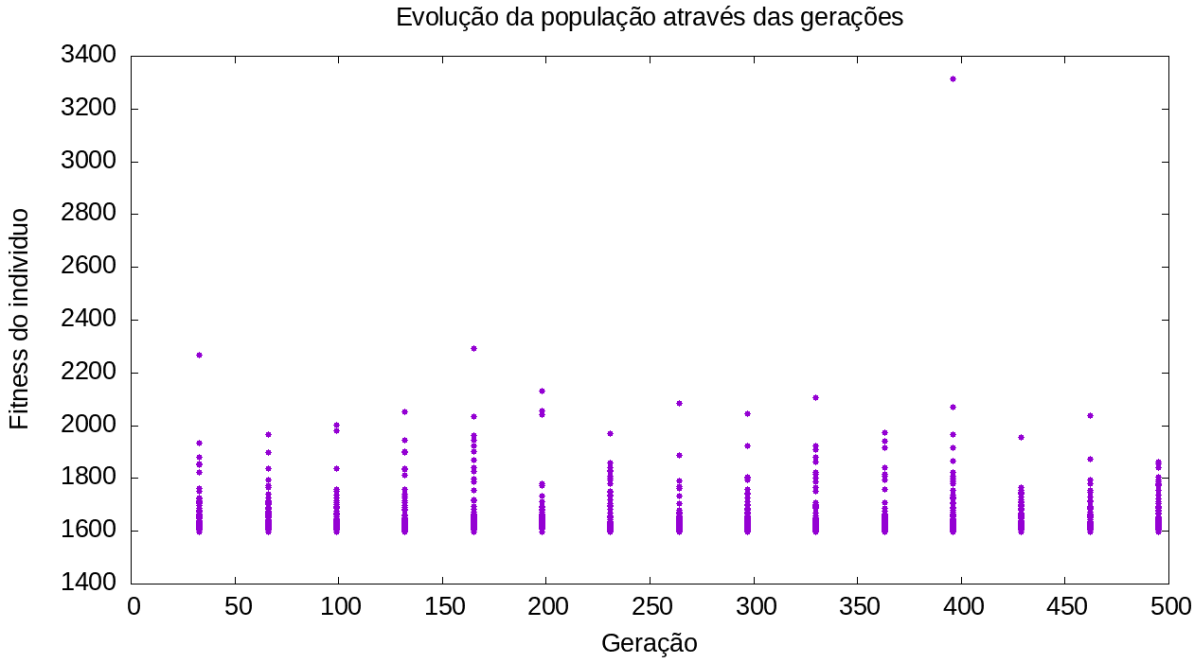


Gráfico da Função 14

Para F15 temos:

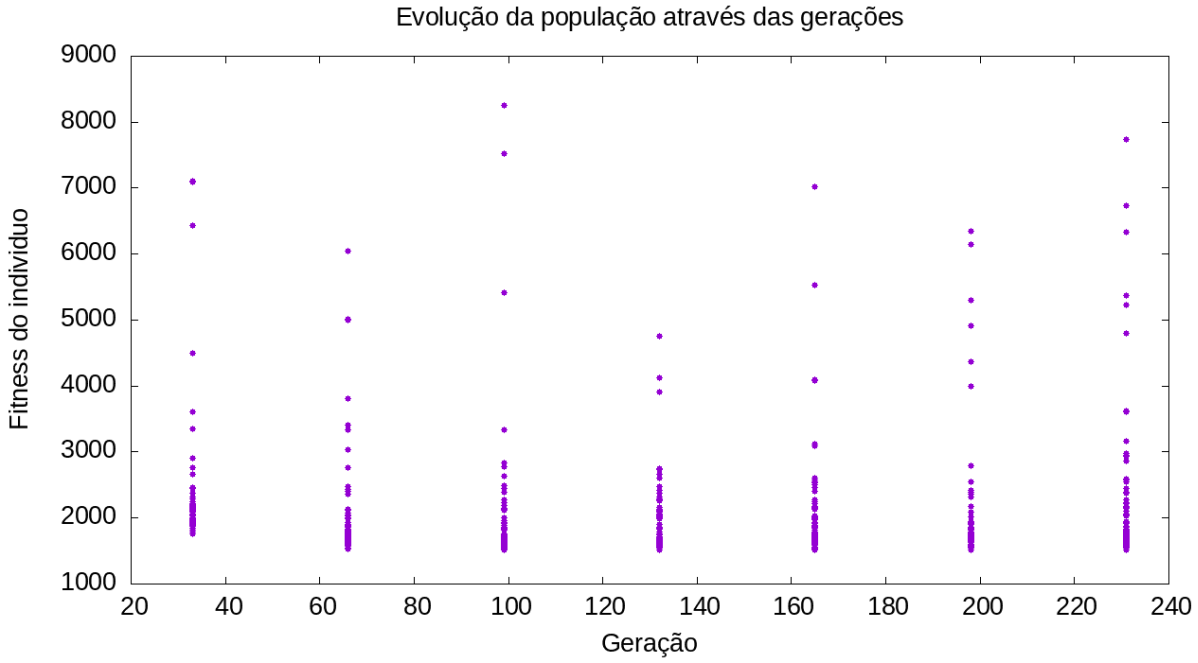


Gráfico da Função 15

2.3.2 Configurações do dispositivo utilizado para coleta de resultados

Tabela 3 Especificações do dispositivo utilizado para coleta de dados

Marca	Acer
Modelo	VX5-591G-78BF
Processador	I7-7700HQ
Memória RAM	16GB DDR4 2666Mhz
Placa de Vídeo	GTX 1050 TI + Intel HD Graphics 630
Armazenamento	250GB SSD NVME M.2 + 480GB SSD SATA 3.5

3 CONCLUSÃO

Em conclusão, o uso de algoritmos genéticos mostrou-se uma abordagem eficaz para a resolução de problemas de otimização.

Após a realização de diversos experimentos e testes, foi possível observar que o algoritmo genético island-model híbrido apresentou um desempenho superior em comparação ao algoritmo genético clássico. Isso se deve ao fato de que o island-model híbrido combina as vantagens dos modelos paralelos e sequenciais de algoritmos genéticos, permitindo uma melhor exploração do espaço de soluções e evitando a estagnação do algoritmo.

Dessa forma, fica evidente que o AGIMH pode ser uma alternativa interessante para a resolução de problemas de otimização em que a busca por soluções ótimas é complexa e exige grande poder de processamento. É importante destacar que, apesar do bom desempenho do island-model híbrido, a escolha do melhor algoritmo para um determinado problema de otimização ainda depende de diversos fatores, como o tamanho da população, o número de iterações e a natureza do problema em questão.

4 REFERÊNCIAS

TOLEDO, Claudio Fabiano Motta; DE OLIVEIRA, Renato Resende Ribeiro; FRANÇA, Paulo Morelato. A hybrid multi-population genetic algorithm applied to solve the multi-level capacitated lot sizing problem with backlogging. Computers & Operations Research, v. 40, n. 4, p. 910-919, 2013.

SOUZA, Gustavo Pinho Kretzer de. Otimização de funções reais multidimensionais utilizando algoritmo genético contínuo. 2014.

GUNGOR, Imral et al. Integration search strategies in tree seed algorithm for high dimensional function optimization. International journal of machine learning and cybernetics, v. 11, p. 249-267, 2020.

UMBARKAR, Anant J.; SHETH, Pranali D. Crossover operators in genetic algorithms: a review. ICTACT journal on soft computing, v. 6, n. 1, 2015.

TOMASSINI, Marco. Spatially structured evolutionary algorithms: artificial evolution in space and time. Springer Science & Business Media, 2006.