UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
FACULDADE DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

**Gabriel Nunes Crispino**

# Paralelização de algoritmo genético com operador não convencional

Belém – Pará

2018

Gabriel Nunes Crispino

# Paralelização de algoritmo genético com operador não convencional

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. Instituto de Ciências Exatas e Naturais. Faculdade de Computação. Universidade Federal do Pará.

---

**Prof. Dr. Claudomiro de Souza de Sales Junior**
Orientador

---

**Me. Reginaldo Cordeiro dos Santos Filho**
Coorientador

---

**Prof. Dr. Josivaldo de Souza Araújo**
Avaliador

---

**Me. Cassio Trindade Batista**
Avaliador

Belém – Pará
2018

*Dedico este trabalho aos meus pais, Nicolau e Gláucia,*
*e à minha irmã, Rafaela*

# Agradecimentos

Agradeço primeiramente aos meus amados pais e à minha querida irmã, por estarem sempre presentes em minha vida e por todo apoio prestado durante o período do curso.

Agradeço à minha namorada Lilya, por todo o amor e carinho que me deu sempre que necessário e por ter me ajudado no decorrer deste trabalho.

Ao meu orientador, prof. Claudomiro, por tudo que me ensinou durante a graduação e por toda a assistência dada durante a orientação do presente trabalho. Agradeço também ao meu coorientador Reginaldo pela ajuda prestada no desenvolvimento do trabalho.

A todos os outros professores com os quais eu tive a oportunidade de aprender algo durante a minha vida, ajudando direta ou indiretamente na minha vida acadêmica.

Aos amigos que fiz na universidade, por terem me acompanhado tanto nos momentos de dedicação quanto de lazer. Agradeço também aos amigos que a minha cidade natal Macapá me trouxe. Além disso, agradeço aos amigos que o intercâmbio me deu, por terem sido uma grande família pra mim e compartilhado essa experiência inesquecível comigo.

A toda a minha família, por estar sempre presente em todos os momentos de conforto e de dificuldades que a vida nos proporciona.

Enfim, gostaria de agradecer a todas as pessoas que de certa forma me ajudaram durante todos os anos de graduação.

# Resumo

Algoritmos genéticos paralelos se aproveitam de execução concorrente para obter melhores resultados e um melhor aproveitamento do *hardware* da máquina. Geralmente são utilizadas diversas subpopulações que evoluem concorrentemente e que se comunicam através de uma política de migração definida, a fim de alcançar uma melhor exploração do espaço de busca. Existem também os operadores genéticos não convencionais, que se inspiram no funcionamento de alguns organismos, como vírus e bactérias, para alterar a arquitetura do algoritmo genético. É comum que esses operadores utilizem populações auxiliares contendo indivíduos especiais para obter maior variabilidade genética. Este trabalho propõe uma implementação de um algoritmo genético paralelo que se utiliza do operador genético não convencional de recombinação por transformação bacteriana, com o objetivo de comparar o seu desempenho tanto com algoritmos genéticos sequenciais que utilizam esse mesmo operador quanto com versões paralelas que utilizam operadores convencionais. Os resultados mostraram que a implementação apresentada em geral trouxe uma maior velocidade de convergência, maior robustez e precisão, se comparada a outras implementações utilizadas.

**Palavras-chave**: algoritmos genéticos, otimização, algoritmos genéticos paralelos, computação paralela, operadores genéticos não convencionais.

# Abstract

Parallel genetic algorithms take advantage of concurrent execution to obtain better results and better use of the machine's hardware. Usually there are multiple subpopulations that evolve concurrently and communicate through a defined migration policy, to achieve better exploration of the search space, for example. Non conventional genetic operators are the ones inspired by some natural organisms, such as viruses and bacteria, to modify the genetic algorithm architechture. It's common that these operators use auxiliar populations containing special individuals to obtain better genetic variability. This work proposes an implementation of a parallel genetic algorithm that makes use of the recombination by bacterial transformation genetic operator, and then compares its performance with both sequential genetic algorithms that make use of this same operator and parallel versions that use conventional genetic operators. The results show that the presented implementation in general brought a higher speed of convergence, higher robustness, and precision, if compared to the other implementations that are used.

**Keywords**: genetic algorithms, optimization, parallel genetic algorithms, parallel computing, non conventional genetic operators.

# Lista de ilustrações

# Lista de tabelas

# Lista de abreviaturas e siglas

| | |
|---|---|
| AG | Algoritmo Genético |
| AGSC | Algoritmo Genético Sequencial Convencional |
| AGSN | Algoritmo Genético Sequencial Não Convencional |
| AGPC | Algoritmo Genético Paralelo Convencional |
| AGPN | Algoritmo Genético Paralelo Convencional |
| RIGA | *Retroviral Iterative Genetic Algorithm* |
| PMDSAM | Política de Migração Dupla de Substituição Aleatória dos Melhores |
| OpenMP | *Open Multi-Processing* |

# Sumário

# 1 Introdução

Algoritmos genéticos (AGs) constituem um ramo dentro da área de computação evolucionária focado em solucionar problemas de difícil resolução. Com essa finalidade, utilizam um processo de recombinação geneticamente inspirado como um meio para gerar novos candidatos a solução dentro de uma população de indivíduos (EIBEN; SMITH, 2015). Por já constituirem uma boa alternativa para esse tipo de problema, o seu uso é comum para com os mais diversos objetivos. Alguns exemplos de aplicações são, dentre outros, coloração de grafos (CHEN et al., 2015), uso em motores de busca *online* (KUMAR; KUMAR; DIXIT, 2015), reconhecimento de padrões para diagnóstico de doenças (TARIQUE; ZAMEE; KHAN, 2014) e *esteganálise* (UFRJ, 2008; FUQIANG; MINQING; JIA, 2014) .

Propostas de novas estruturas e arquiteturas para AGs são temas constantes de trabalhos na área. Diferentes ideias de operadores genéticos (como de seleção, cruzamento e mutação) são concebidas com frequência, com o objetivo de tornar o seu processo mais efetivo e robusto (HERDA, 2016; COHOON et al., 1987).

Uma das arquiteturas propostas exploradas com frequência é a dos AGs paralelos. Existem diversas maneiras de paralelizar um AG. Uma das primeiras propostas de implementação de um AG com diversas populações executando em paralelo e compartilhando indivíduos foi apresentada em Cohoon et al. (1987). Nesse trabalho, é demonstrado que a evolução de diferentes subpopulações distribuídas tem como consequência melhores resultados. Várias dessas arquiteturas são reunidas e demonstradas em Cantú-paz (1999) e Alba e Troya (1999). Por exemplo, pode-se paralelizar partes do seu funcionamento (como o cálculo do *fitness*) ou levar isso a um passo além e modificar a sua arquitetura como um todo para que funcione de modo mais distribuído (UPADHYAYULA; KOBTI, 2015; MORADY; DAL, 2016; COHOON et al., 1987). O objetivo deste trabalho se alinha mais com o segundo caso.

Outra alternativa que é proposta no contexto da área de algoritmos genéticos é a do uso de operadores não convencionais (PEREIRA, 2017). Esses operadores buscam incorporar elementos inspirados em outros organismos da natureza (por exemplo, bactérias e vírus) à arquitetura do AG, com o objetivo de encontrar algum tipo de melhora nos resultados do algoritmo. Kubota, Fukuda e Shimojima (1996) propõem uma das primeiras implementações de AG inspiradas na teoria de evolução dos vírus, por exemplo. Nesse caso, uma população auxiliar de indivíduos chamados de vírus coexiste com a população principal de candidatos, e é definido um operador de infecção para dar origem a novas possíveis soluções a partir dos indivíduos e dos vírus. A existência desta população auxiliar e do operador de infecção aumenta o potencial para uma maior variabilidade genética nos

indivíduos da população principal.

Este trabalho propõe uma implementação conjunta utilizando um operador não convencional de recombinação por transformação bacteriana (PEREIRA, 2017) e um AG com arquitetura paralela assíncrona (FERREIRA, 2017), com o objetivo de melhorar a eficiência de busca do algoritmo e, consequentemente, o encontro de melhores resultados.

## 1.1 Revisão da literatura

Por prometerem diferentes modos de melhorar o funcionamento de um AG, as suas alternativas paralelas já foram e continuam sendo o foco da discussão de vários trabalhos, como em Herda (2016) , Chen et al. (2015) e Cohoon et al. (1987). Em Morady e Dal (2016) e Upadhyayula e Kobti (2015), são utilizados AGs paralelos com multipopulações funcionando concorrentemente, trazendo um diferente plano de exploração dentro do espaço de busca do domínio em questão. A alternativa apresentada nesse trabalho consiste no uso de um AG paralelo assíncrono com várias subpopulações que trocam indivíduos por meio de uma política de migração.

Ao se paralelizar um AG, o uso de uma arquitetura multipopulacional não é estritamente necessário. Herda (2016) propõe uma implementação com apenas uma população, mas que utiliza programação paralela para operadores do AG, principalmente no uso da *hipermutação*, que nesse trabalho é definida como um processo em que "cada gene na solução é trocado por todos os outros genes admissíveis ainda não incluídos na solução e a troca que causar maior melhora na função objetivo é executada". Em Chen et al. (2015) é utilizada a tecnologia CUDA (NVIDIA, 2018) para se aproveitar do seu alto número de núcleos para paralelizar operadores do AG com o uso de um número de *threads* consideravelmente maior que em uma CPU, tendo um maior foco na melhora de desempenho do algoritmo. Ao paralelizar um AG dessa maneira, o objetivo principal é em obter um melhor desempenho e um melhor aproveitamento do poder computacional oferecido pela máquina.

Em Ferreira (2017), são avaliadas diferentes políticas de migração, que estabelecem o modo com que as subpopulações trocam indivíduos durante o funcionamento do AG paralelo. Nesse trabalho é exposta a eficiência das políticas de migração dupla, quando duas populações trocam indivíduos dentro do mesmo processo de migração, formando uma comunicação bilateral. Dependendo de como essa comunicação é feita, isso pode trazer uma maior capacidade de exploraçao às populações do AG, melhorando seus resultados.

Pereira (2017) traz uma comparação de alguns operadores não convencionais, expondo suas características, juntamente com sua implementação e resultados. São estudados os operadores microbiano (TARIQUE; ZAMEE; KHAN, 2014), de seleção negativa (LI; GU; LIU, 2006), meiose (WIRIYASERMKUL; BOOBJING; CHANVARASUTH, 2010),

imunidade (LI; GU; LIU, 2006), retrovírus (MOREIRA; TEIXEIRA; OLIVEIRA, 2010) e é introduzido o operador de recombinação por transformação bacteriana. Este novo operador apresenta melhores resultados quanto à convergência e ao encontro dos valores ótimos das funções de teste, em comparação com as implementações dos outros operadores não convencionais e convencionais.

Em Fuqiang, Minqing e Jia (2014) pode-se ver um trabalho que utiliza um AG com evolução de vírus, aplicado como classificador de conjuntos em um problema de esteganálise. Nesse trabalho é demonstrado que este classificador gerado com o auxílio da população adicional de vírus tem melhor desempenho com relação a outros classificadores existentes.

## 1.2 Motivação

Pela revisão de trabalhos correlatos observa-se que existe o constante estudo de arquiteturas alternativas para adaptar os AGs visando melhores resultados ou melhor eficiência, como é o caso do uso de paralelismo nesse contexto (HERDA, 2016; CHEN et al., 2015; COHOON et al., 1987).

Utilizar uma população auxiliar e introduzir um novo operador também é objeto de estudo de alguns trabalhos recentes, e bem comum no caso dos AGs com vírus, como pode ser visto em Fuqiang, Minqing e Jia (2014) Moreira, Teixeira e Oliveira (2010). Como é demonstrado em Pereira (2017) a aplicação de operadores não convencionais traz vantagens, dependendo de qual técnica é utilizada.

A partir disso, constata-se a motivação de se buscar uma alternativa de mesclar arquiteturas de AGs paralelos com o uso de algum operador não convencional. Ademais, verificar a possibilidade de aproveitar-se das duas técnicas, com o fim de obter melhores resultados e dispor de mais uma alternativa para a solução de problemas computacionalmente não triviais. Tendo isso em vista, o presente trabalho apresenta uma alternativa que une essas duas abordagens, utilizando uma arquitetura paralela de múltiplas subpopulações em conjunto com um operador genético não convencional.

## 1.3 Justificativa

Verifica-se que a proposta de novas arquiteturas, assim como novos operadores dentro do desenvolvimento de algoritmos genéticos pode trazer bons resultados. Em Ferreira (2017) são mostrados bons resultados utilizando-se de uma proposta de AG paralelo multipopulacional assíncrono com diferentes políticas de migração, além de uma comparação entre elas.

Do mesmo modo, Pereira (2017) conclui que alguns operadores não convencionais, apresentam qualidade considerável, tanto na robustez quanto na velocidade de convergência da população em específico. Além disso, nesse trabalho destaca-se o novo operador de recombinação por transformação bacteriana, que consegue realizar uma estratégia de busca aprimorada, apresentando bons resultados se comparado aos outros operadores convencionais e não convencionais.

Pela revisão dos trabalhos correlatos atesta-se a existência de estratégias de uso de operadores não convencionais. Esse número, no entanto, não é extenso. O mesmo pode ser dito com relação ao uso desses operadores em AGs que utilizam qualquer tipo de arquitetura paralela. Isso justifica avaliar o desempenho de um AG com arquitetura paralela de múltiplas ilhas e um operador não convencional, e verificar se há melhora nos resultados.

## 1.4 Objetivos

### 1.4.1 Objetivos gerais

O objetivo geral deste trabalho é realizar a implementação de um algoritmo genético paralelo assíncrono multipopulacional, que utiliza o operador genético não convencional de recombinação por transformação bacteriana. Além disso, na implementação é utilizada a política de migração dupla de substituição aleatória dos melhores (PMDSAM) (FERREIRA, 2017), que indica como as subpopulações trocam indivíduos entre si.

### 1.4.2 Objetivos específicos

- Fazer a implementação de um AG paralelo, assíncrono, que utiliza a PMDSAM e o operador genético de recombinação por transformação bacteriana;
- Para efeito de comparação, realizar a implementação prática de um AG sequencial convencional, de um AG sequencial com operador não convencional de recombinação por transformação, e de um AG paralelo assíncrono que utiliza operadores convencionais;
- Coletar os resultados de todas as implementações realizadas, e compará-los entre si;
- Verificar se a estrutura presente na implementação proposta oferece alguma vantagem com relação às outras arquiteturas de AGs definidas nas outras implementações feitas.

## 1.5 Metodologia

De início foi realizado um estudo sobre algoritmos genéticos, suas arquiteturas e operadores mais comuns. Depois, um estudo mais direcionado foi feito, principalmente sobre AGs paralelos, políticas de migração e operadores não convencionais. Além disso, foi feito um estudo da biblioteca de computação paralela `OpenMP` (OpenMP Architecture Review Board, 2013), para implementar o paralelismo dentro do trabalho.

A partir do estudo feito e da literatura foi adotada uma estrutura para o AG paralelo, no qual foi incluso o uso do operador não convencional já citado. Para a implementação foi utilizada a linguagem C++, em conjunto com a biblioteca de programação paralela `OpenMP` 4.0 para o uso de paralelismo com *threads* no programa. Essa escolha foi feita por constituir uma combinação muito usada nos mais diversos trabalhos acadêmicos e também pelo fato de `OpenMP` ser uma biblioteca fácil de assimilar e de incorporar a um projeto.

O AG implementado utiliza os mesmos parâmetros de Pereira (2017), para efeito de comparação. Além disso, é aplicado na otimização em 2 dimensões de 8 funções: *Schaffer2*, *Booth*, *Ackley*, *Griewank*, *Eggholder*, *DropWave*, *Rosenbrock* e *Rastrigin*. Em todas é utilizada a representação de números de ponto-flutuante para candidatos a soluções. A implementação foi testada em um computador com um único processador Intel® Core i7-6700HQ de 4 núcleos, e 16 GB de memória RAM. Foram realizadas 30 execuções para cada AG diferente implementado e para cada uma das funções de teste, e depois coletados os resultados referentes a essas execuções.

# 2 Referencial teórico

Como já citado no capítulo anterior, esse trabalho se utiliza de dois recursos que diferenciam a sua implementação de uma abordagem convencional de um AG sequencial: o uso de uma arquitetura paralela e de um operador genético não convencional. Nas duas próximas seções deste capítulo, esses temas serão discutidos com mais detalhes. Na última seção do capítulo, serão demonstradas as 8 funções objetivo que foram utilizadas para teste neste trabalho.

## 2.1 Algoritmos genéticos paralelos

Nesta seção do capítulo serão tratados temas relacionados à área de Algoritmos genéticos paralelos. Principalmente, nas 3 subseções seguintes, uma discussão sobre **classificação**, **topologia**, e **políticas de migração** de um AG paralelo, relacionadas ao presente trabalho.

### 2.1.1 Classificação

Tal qual dito no capítulo anterior, Cohoon et al. (1987), Alba e Troya (1999) e Cantú-paz (1999) apresentam abordagens de uso de paralelismo no contexto de um AG. Em seguida nesta seção serão tratados 3 tipos de classificações de um AG paralelo: uso de apenas uma população (também conhecido como paralelização global, ou *global parallelization*), AG paralelo com várias subpopulações com granularidade grossa (*coarse-grain*) e AG paralelo com granularidade fina (*fine-grain*).

AGs com **paralelização global** ou AG mestre-escravo (*master-slave*), de acordo com Alba e Troya (1999), consistem em "avaliar, ou talvez realizar cruzamento e mutação em paralelo, enquanto a seleção utiliza toda a população". Ou seja, consiste em uma categoria de AGs que ainda utiliza uma única população global, mas possui algumas partes do algoritmo que são paralelizadas. Mais especificamente, nos operadores de seleção, mutação e principalmente no cálculo do *fitness*. Dependendo do domínio do problema, a última pode ser uma tarefa computacionalmente custosa, porém facilmente paralelizável, o que faz com que essa alternativa seja de implementação mais óbvia e com ganhos no desempenho da aplicação como um todo.

AGs **paralelos de granularidade grossa** (*coarse-grain*) são AGs que possuem várias subpopulações fracamente acopladas que evoluem separadamente (CANTÚ-PAZ, 1999). Também conhecidos como AGs com múltiplas **ilhas** ou **demes**, essa arquitetura distribuída evolui cada uma das subpopulações como AGs sequenciais isolados, os quais se

comunicam trocando indivíduos. Como e quando essa comunicação ocorre é definida pelo que chamamos de política de migração (FERREIRA, 2017). Na Figura 1, pode-se ver uma descrição visual de como as populações se comportam dentro de um AG que segue esse modelo. Essa classificação de AG paralelo é a que será mais explorada durante o curso deste trabalho.

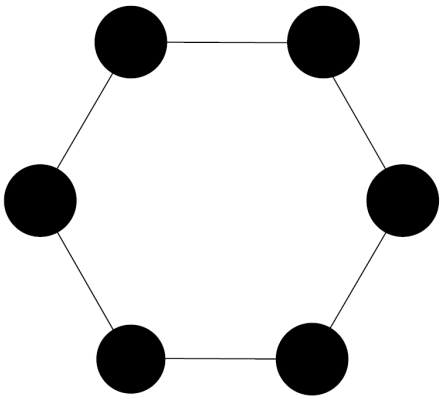


Figura 1 – Representação visual de uma topologia de um AG paralelo de granularidade grossa.

Fonte: (CANTÚ-PAZ, 1999)

Em (COHOON et al., 1987), é feita uma explicação do porquê de AGs paralelos com múltiplas ilhas funcionarem. Essa explicação é baseada na noção de equilíbrio pontuado (*punctuated equilibria*), que é um modelo evolutivo baseado em dois princípios: **estase** e **especiação alopátrica**. Estase implica que após o equilíbrio ter sido atingido no ambiente, haverá pouca mudança na composição genética dos indivíduos. Especiação alopátrica consiste na evolução rápida de espécies após terem sido geográficamente separadas. Na Figura 2, está disponível um pseudocódigo de um AG paralelo síncrono com múltiplas subpopulações. Nesse algoritmo, as múltiplas populações, cada uma em um processo diferente, evoluem isolada e paralelamente por E épocas, que são as iterações principais do algoritmo e são divididas em G gerações. Após o término do processamento das gerações, cada subpopulação realiza migração ao trocar um conjunto de soluções com as populações vizinhas. Por fim, cada população seleciona os indivíduos que vão sobreviver para a próxima época, e assim o algoritmo continua até o fim das E épocas.

Em (CANTÚ-PAZ, 1999) são listadas 2 razões intuitivas de porque AGs paralelos com granularidade grossa constituem uma boa extensão ao formato tradicional de um AG sequencial:

- AGs com múltiplas ilhas parecem ser uma extensão simples do AG sequencial convencional: o mesmo algoritmo já implementado é utilizado nas subpopulações, restando adicionar a lógica do paralelismo e a troca de indivíduos entre essas

```
para cada iteração E
    para cada processador I
        executar AG por G gerações
    fimpara

    para cada processador i
        para cada vizinho j de i
            enviar um conjunto de soluções,
            S_ij, de i para j
        fimpara
    fimpara

    para cada processador i
        selecionar uma subpopulação de n elementos
    fimpara

fimpara
```

Figura 2 – Pseudocódigo de um AG paralelo síncrono com múltiplas subpopulações.

Fonte: (COHOON et al., 1987)

populações no momento da migração. Isso facilita, também, a conversão de um AG sequencial para um AG paralelo com múltiplas ilhas;

- Computadores paralelos de granularidade grossa são frequentemente disponíveis, e quando não são, podem ser substituídos por vários computadores conectados em uma rede de processamento, ou até mesmo em um único processador usando softwares adequados.

Alba e Troya (1999) fornecem uma explicação mais teórica e detalhada do porquê desse tipo de AG funcionar bem. Como já foi dito antes, um dos objetivos no uso de várias subpopulações é o aumento da capacidade de exploração dentro do espaço de busca do problema em questão. Tendo isso como ponto de partida, esse mesmo trabalho detalha como o teorema do esquema (*schema theorem*) (EIBEN; SMITH, 2015) pode ser estendido para AGs paralelos com várias subpopulações e, dessa maneira, expor como é possível atingir um maior potencial na exploração do espaço de busca em questão.

AGs **paralelos de fina granularidade**, ao contrário dos de grossa granularidade, possuem apenas uma população. No entanto, nela os indivíduos são dispostos em uma grade (usualmente bidimensional), diferentemente de um AG sequencial. Nessa disposição, esses indivíduos ficam restritos a interagir apenas com os seus vizinhos. Além disso, eles são fortemente conexos e estão disponíveis em um número consideravelmente maior do que ilhas em um AG paralelo de granularidade grossa. Sua disposição permite com que essa abordagem seja bem adequada para computadores massivamente paralelos, onde cada indivíduo (ou pequeno grupo de indivíduos) possa ser alocado para cada unidade de processamento desse computador (ALBA; TROYA, 1999; CANTÚ-PAZ, 1999). Na

Figura 3, pode ser visto um exemplo de disposição de elementos em um AG paralelo de granularidade fina.

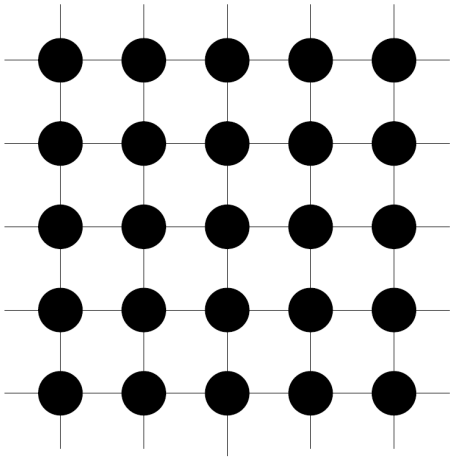


Figura 3 – Representação visual de uma topologia de um AG paralelo de granularidade fina.

Fonte: Cantú-paz (1999)

Outro tipo de classificação de AG paralelo é a **híbrida**, que consiste em combinar uma abordagem de paralelização de um AG com outra, mesmo que ambas sejam do mesmo tipo. Por exemplo, criar um AG com múltiplas ilhas, onde cada ilha executa um AG com múltiplas ilhas. Na Figura 4, podemos visualizar um AG paralelo híbrido. Nesse caso, há uma combinação de AGs paralelos com granularidade fina e grossa, onde o algoritmo é de granularidade grossa, mas cada uma de suas ilhas executa um AG com granularidade fina (ALBA; TROYA, 1999).

### 2.1.2 Topologia de um AG paralelo com múltiplas ilhas

Outro aspecto importante de um AG paralelo é a sua topologia. Cantú-paz (1999) afirma que um AG paralelo é mais complexo que a sua versão sequencial e além disso, que a sua topologia é um dos parâmetros que vão indicar o funcionamento da conexão entre as suas subpopulações. Em outras palavras, ela define como as populações de um AG paralelo se dispõem espacialmente.

Existem vários tipos de topologia comumente utilizados em AGs paralelos. Nos trabalhos citados em Alba e Troya (1999) e Cantú-paz (1999), topologias que são mencionadas com frequência são as de hipercubo, anel e de grafo (completo ou não).

A topologia de **hipercubo** costuma ser utilizada porque ela pode ser facilmente mapeada a um computador paralelo que segue essa mesma organização. Entende-se que tal escolha seja natural dependendo do equipamento disponível para o pesquisador.

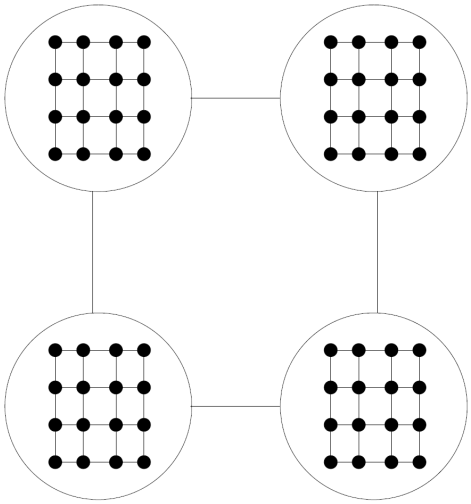


Figura 4 – Representação visual de uma topologia de um AG paralelo híbrido.

Fonte: Cantú-paz (1999)

A estrutura de **grafo completo** é também comumente utilizada. Por não criar muitas assunções sobre a topologia em si, já que nela qualquer população pode se comunicar com qualquer outra, pode ser uma boa alternativa para o desenvolvimento rápido de um AG paralelo. Em contrapartida, a escolha dessa topologia pode causar problemas quanto à escalabilidade e paralelização do algoritmo, em função do seu forte acoplamento (ALBA; TROYA, 1999). É a topologia utilizada em Ferreira (2017), e também foi a escolhida para se utilizar no presente trabalho.

Outra topologia de fácil implementação é a estrutura de **anel**. Essa consiste em uma estrutura de um grafo circular, onde cada nó (subpopulação) se conecta apenas com os nós anterior e próximo. Apesar de poder ser bidirecional, é comum que topologias de um AG paralelo e, consequentemente, a topologia de anel, sejam unidirecionais (ALBA; TROYA, 1999).

Nas Figuras 5 e 1, podem ser vistos exemplos das topologias de grafo completo e anel, respectivamente.

### 2.1.3 Políticas de migração

Eiben e Smith (2015) definem migração como o processo de seleção e troca de um número de indivíduos de cada subpopulação que serão trocados com subpopulações vizinhas. Ou seja, define um dos parâmetros que indicam como a variabilidade genética atingida pela evolução paralela entre subpopulações funciona.

Além disso, Eiben e Smith (2015) também expõem duas perguntas relacionadas ao funcionamento e desempenho de um AG paralelo, que também estão relacionadas ao

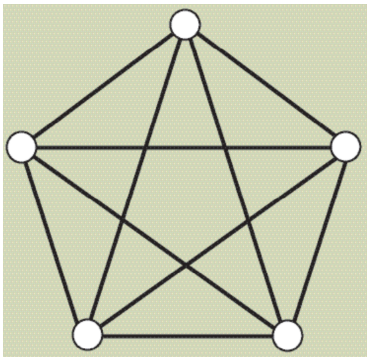


Figura 5 – Exemplo de grafo completo.

Fonte: GRAFOS... (2012)

processo de migração:

- **Com que frequência deve-se trocar indivíduos?** É importante para uma política de migração que se defina em que momento um ou mais indivíduos serão trocados entre subpopulações. Por exemplo, é comum que em AGs paralelos com migração **síncrona** (o conceito de sincronia na migração será tratado com mais detalhes mais a frente), a migração ocorra ao final de um certo número de gerações (também chamado de **época**) ocorridas no conjunto das subpopulações do AG (COHOON et al., 1987). Pode-se usar, por exemplo, um parâmetro de probabilidade de migração, para limitar a troca de indivíduos e assim evitar convergência prematura. Cantú-paz (1999) indica que é preciso avaliar o momento certo para executar o processo de migração dentro do algoritmo. Se for executado muito cedo, por exemplo, pode provocar um desperdício de recursos, pelo fato dos indivíduos trocados não impactarem de maneira considerável o processo de busca na direção certa. Por outro lado, se for executado muito tarde, pode deixar de acompanhar certos passos da evolução de uma ilha e consequentemente, de trocar essa informação com outras ilhas, podendo comprometer o processo evolutivo.

- **Quantos e quais indivíduos devem ser trocados?** O critério de escolha de quais indivíduos de uma subpopulação serão trocados com outra também é um parâmetro importante dentro do comportamento de uma política de migração. Intuitivamente, o número de indivíduos a serem trocados com outra população pode afetar no nível de isolamento entre essas populações. Ademais, a escolha de quais indivíduos serão trocados também impacta o desempenho do algoritmo. No AG paralelo proposto em Upadhyayula e Kobti (2015), indivíduos fracos de uma população que migram para outra população podem obter um papel diferente, e isso pode acarretar em um maior valor de *fitness* na subpopulação de destino.

Em Ferreira (2017) são comparadas políticas consolidadas na literatura, com o foco de verificar a viabilidade de políticas de migração dupla, que são políticas em que duas populações trocam indivíduos dentro de um único processo de migração. É constatado que essas políticas se comportam bem em relação a políticas simples. Uma delas, a PMDSAM, se comporta bem nas avaliações dentro das funções de testes apresentadas no trabalho e, por isso, é a política que será tratada pelo presente trabalho. Essa política, como sugere o nome, escolhe duas populações p1 e p2, aleatoriamente, dentro do conjunto de todas as subpopulações. Depois, envia o melhor de p1 para p2, e vice-versa. Por fim, em cada subpopulação, substitui o indivíduo recebido por um indivíduo aleatório dentro da população em questão. Na Figura 6, pode-se ver um diagrama de atividades que descreve o funcionamento dessa política de migração.

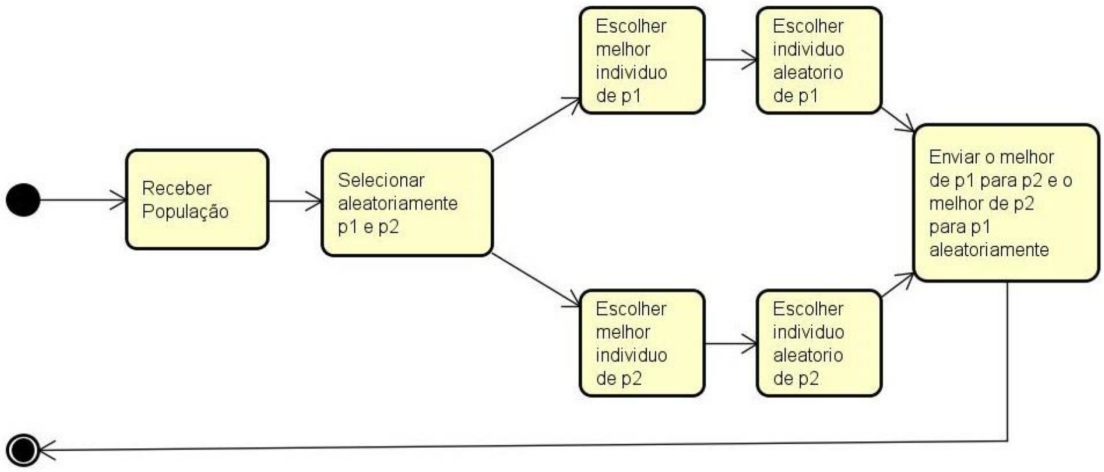



Figura 6 – Diagrama de atividades que descreve a política de migração PMDSAM.

Fonte: Ferreira (2017)

Outro conceito importante é o de sincronia dentro de um AG paralelo com múltiplas ilhas. Alba e Troya (1999) definem que um AG paralelo **síncrono** é aquele em que a migração ocorre sempre em uma mesma parte do algoritmo. Isso significa que a migração dentro de um algoritmo dessa classificação ocorre em intervalos constantes. Por exemplo, a migração em um AG paralelo síncrono pode ocorrer ao fim da n-ésima geração de cada população. Do mesmo modo, sua alternativa **assíncrona** é aquela que potencialmente ocorre em qualquer momento da execução de cada subpopulação. Como já foi dito antes nesta mesma seção, classicamente AGs paralelos costumam utilizar uma política de migração síncrona, que vai ser executada a cada época do algoritmo. No presente trabalho, assim como em Ferreira (2017), é utilizada uma política de migração assíncrona.

## 2.2 Operadores genéticos não convencionais

A definição de operadores genéticos não convencionais pode ser resumida a operadores que diferem do funcionamento dos operadores convencionais (por exemplo, operadores de mutação e cruzamento). Como já foi mencionado previamente neste trabalho, o funcionamento desses operadores normalmente é inspirado em organismos da natureza, como bactérias e vírus. Nos próximos parágrafos será brevemente explicado o operador de retrovírus, e na subseção seguinte será explicado com mais detalhes o operador não convencional utilizado nesse trabalho, e que possui uma estrutura semelhante com o de retrovírus: o operador genético de recombinação por transformação bacteriana.

O operador não convencional de retrovírus é utilizado no RIGA, algoritmo proposto em Moreira, Teixeira e Oliveira (2010), e que quer dizer Algoritmo Genético Iterativo Retroviral (*Retroviral Iterative Genetic Algorithm*, em inglês). Esse operador se baseia nos retrovírus, com a justificativa de que esses vírus não possuem mecanismos de correção para desfazer possíveis mutações genéticas que ocorrem no processo de multiplicação viral, o que é um fator importante dentro do funcionamento do RIGA.

O seu funcionamento basicamente se resume a realizar um processo de infecção da população de indivíduos ao fim do *loop* principal do AG. Essa população é formada por cromossomos especiais, com o mesmo tamanho dos cromossomos convencionais, mas que possuem lacunas, cuja quantidade e localização dentro do indivíduo são determinadas aleatoriamente. De acordo com Moreira, Teixeira e Oliveira (2010), a escolha de representar os vírus como indivíduos com espaços vazios se deu em função do objetivo de compartilhar material genético sem a necessidade de manter outra população de cromossomos em paralelo. Na Figura 7, podem ser vistos três exemplos de vírus em representação binária de 8 *bits*, que seguem o formato do RIGA, com lacunas dispostas aleatoriamente.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| (A) | | 1 | | 0 | | 1 | 0 | |
| (B) | | | 1 | | 1 | | | 1 |
| (C) | 0 | | | 1 | | 1 | | |

Figura 7 – Representação de vírus no RIGA.

Fonte: Moreira, Teixeira e Oliveira (2010)

O processo de criação de novos vírus é dado a partir do aproveitamento de material genético de cromossomos da população principal de indivíduos, de maneira análoga ao funcionamento natural de retrovírus. Esse processo pode ser visto na Figura 8: primeiro, um indivíduo vírus é gerado aleatoriamente (A); é obtido um cromossomo da população principal (B); depois, o material genético desse cromossomo preenche as lacunas do vírus

gerado no primeiro passo (C); Por fim, é gerado um novo vírus baseado no material genético do cromossomo gerado no passo anterior, contendo informação tanto do vírus gerado aleatoriamente de início quanto do cromossomo obtido da população principal (D). As lacunas desse novo vírus são inseridas em posições aleatórias. Nas outras posições onde elas não existem, são copiados os valores do cromossomo obtido no passo anterior, para as mesmas posições.

Ademais, o processo de infecção se dá por intermédio do uso de um indivíduo da população de vírus e de um indivíduo da população principal, gerando um novo indivíduo. O vírus sobrescreve o seu material genético nas mesmas posições no cromossomo infectado, formando assim o novo cromossomo. Se esse novo indivíduo tiver melhor aptidão que o indivíduo infectado, o seu `fitvirus`, valor que indica o quão bem o vírus em questão agiu, é incrementado. Caso contrário, é decrementado. A Figura 9 mostra o processo de infecção de um vírus (A) infectando um cromossomo (B), processo que gera um novo cromossomo (C).

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| (A) | | 1 | | 1 | | 1 | 0 | |
| (B) | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| (C) | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| (D) | 0 | 1 | | 1 | | | | 1 |

Figura 8 – Processo de criação de um vírus no RIGA.

Fonte: Moreira, Teixeira e Oliveira (2010)

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **(A)** | | 1 | | 1 | | 1 | 0 | |
| **(B)** | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| **(C)** | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

Figura 9 – Processo de infecção de um indivíduo por um vírus no RIGA.

Fonte: Moreira, Teixeira e Oliveira (2010)

Alguns parâmetros importantes do RIGA são, dentre outros, a taxa de infecção, que indica o número de indivíduos da população que serão infectados; a taxa de elitismo, que define quantos vírus serão mantidos para a próxima geração da população de vírus, e o número de indivíduos da população de vírus em si.

### 2.2.1 Operador genético de recombinação por transformação bacteriana

O operador genético de recombinação por transformação bacteriana é um operador genético não convencional baseado no processo de recombinação que acontece em bactérias. É definido em Pereira (2017) e, nesse trabalho, é avaliado como uma boa alternativa tanto aos AGs convencionais quanto a outros operadores não convencionais avaliados (incluindo o RIGA). Assim como o já citado operador de retrovírus, possui uma população auxiliar, mas dessa vez chamada de população de **mortos**. Esses indivíduos infectam os elementos da população principal após a seleção de sobreviventes para acrescentar mais um fator de variabilidade ao algoritmo.

Durante a seleção dos sobreviventes para a próxima geração do AG, o pior elemento da população principal no momento da escolha de cada sobrevivente é adicionado à população de mortos. Essa população auxiliar é, então, composta apenas por indivíduos que foram em algum momento rejeitados da população principal por meio desse processo. Além disso, a população de mortos possui o seu tamanho definido como um dos parâmetros do operador não convencional.

O processo de infecção é simples: para cada infecção, é escolhido um cromossomo aleatório dentro da população principal, e um cromossomo aleatório dentro da população de mortos. Um indivíduo novo é gerado por intermédio do cruzamento aritmético simples entre esses dois indivíduos selecionados aleatoriamente. Caso esse indivíduo tenha melhor aptidão que o indivíduo selecionado para infecção, o cromossomo gerado substitui o previamente selecionado.

Nesse operador existe o valor do `fitmorto` (análogo ao `fitvirus` do operador de retrovírus), que, de maneira semelhante, indica o quanto o indivíduo morto realizou infecções de sucesso e o quanto deve ser preservado nas próximas gerações. Esse valor é incrementado se a infecção realizada for um sucesso, ou seja, se gerar um indivíduo com melhor aptidão do que o selecionado para ser infectado. Se a infecção realizada for falha, esse valor é decrementado. Quando o número de indivíduos na população de mortos superar o valor do parâmetro de tamanho dessa população, são mantidos os indivíduos com maior valor do `fitmorto`. Os outros indivíduos são removidos, até que a população de mortos alcance o seu tamanho máximo novamente.

Na Figura 10, pode ser visto um diagrama de atividades que descreve como um AG que implementa o operador de recombinação por transformação funciona.

## 2.3 Funções objetivo para otimização

Como já foi mencionado na introdução do presente trabalho, para realizar os testes da implementação foram utilizadas 8 funções objetivo para teste. O uso dessas funções

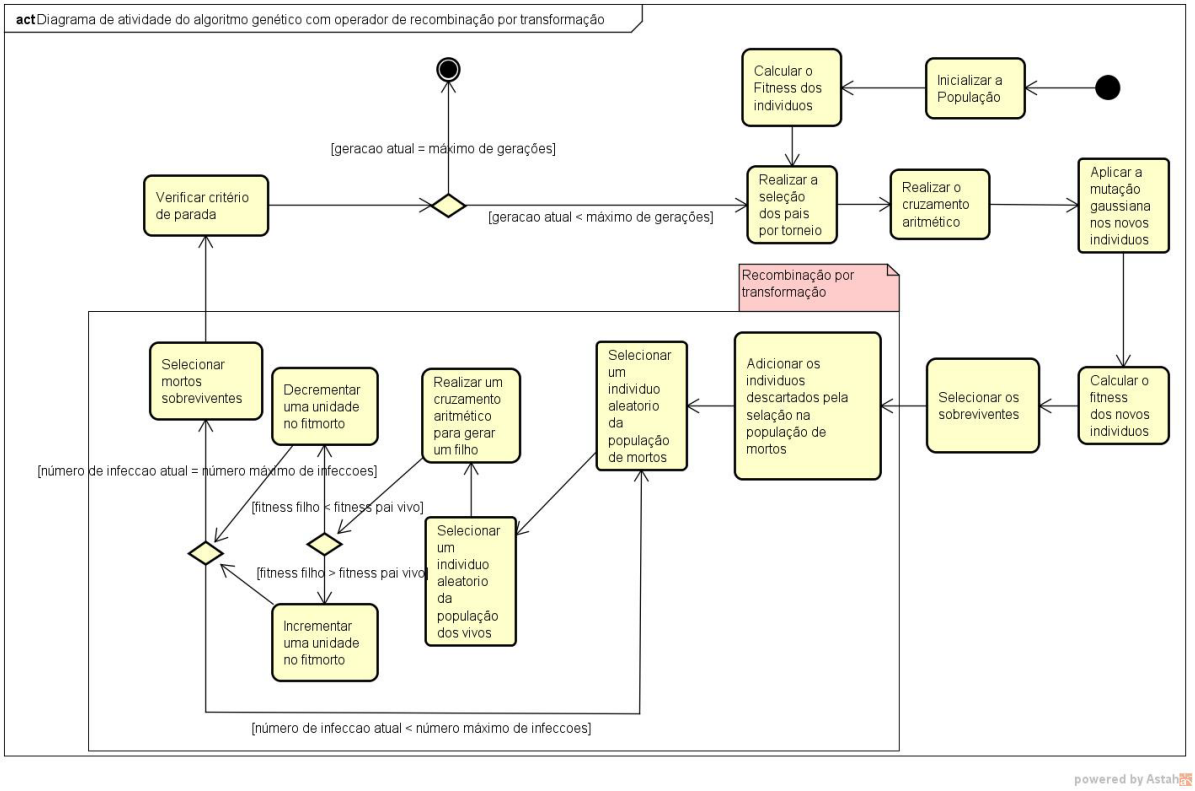



Figura 10 – Diagrama de atividades que descreve o funcionamento do operador não convencional de recombinação por transformação.

Fonte: Pereira (2017)

nos permite avaliar se o algoritmo testado converge da maneira esperada e nos permite quantificar o seu desempenho, para também compará-lo com outras alternativas. Nas subseções a seguir, serão listadas e brevemente discutidas. A definição de todas essas funções foi retirada de Surjanovic e Bingham (2013). No presente trabalho, todas essas funções foram otimizadas levando-se em conta 2 dimensões.

### 2.3.1 *Booth*

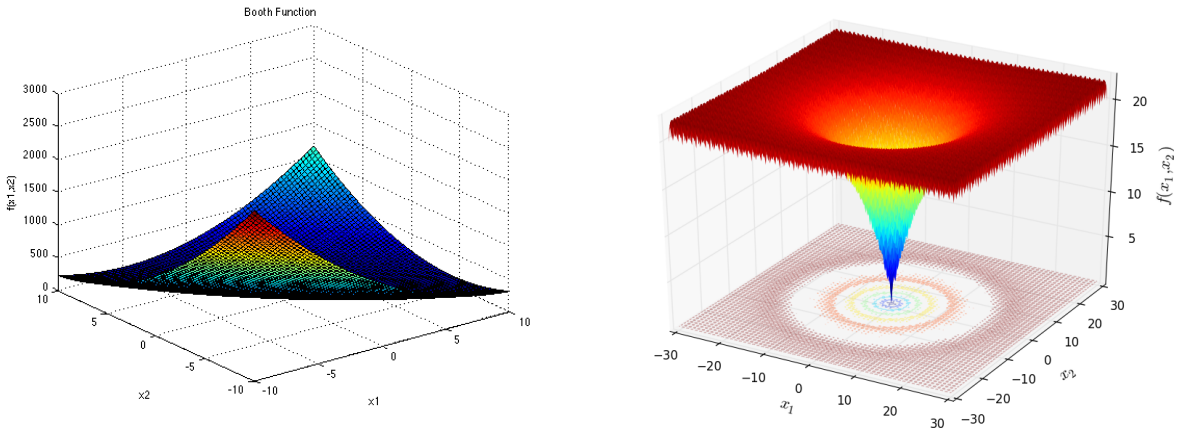
Essa função, dentre todas as apresentadas e testadas nesse trabalho, é a mais fácil de ser minimizada. Isso pode ser visto na sua superfície na Figura 11a: podemos definí-la como uma grande região de um único vale, tornando fácil para o algoritmo para minimizá-la.

Equação:

$$f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 \tag{2.1}$$

Dimensões: 2

Mínimo global: $f(0, 0) = 0$




(a) *Booth* (b) *Ackley*

Figura 11 – Gráficos das funções de teste *Booth* e *Ackley*.

Fonte: Surjanovic e Bingham (2013) e Gavana (2013)

Domínio: $x_i \in [-10, 10]$, para $i = 1, 2$

### 2.3.2 *Ackley*

A função *Ackley* já representa diversos mínimos locais na sua superfície mais externa. Isso apresenta um dos primeiros desafios para um algoritmo de otimização, que é não deixar com que essas regiões comprometam a busca e o encontro do mínimo global.

Equação:

$$f(x) = -20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)} + 20 + e \tag{2.2}$$

Dimensões: $n$

Mínimo global: $f(0, ..., 0) = 0$

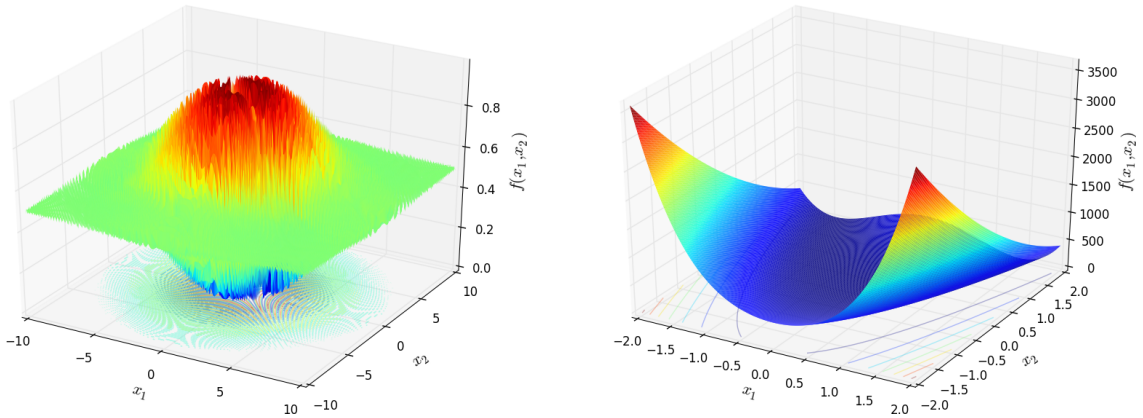
Domínio: $x_i \in [-32.768, 32.768]$, para $i = 1, ..., n$

### 2.3.3 *Schaffer02*

Essa função também é um exemplo de função com vários mínimos locais, mas com uma superfície predominantemente plana exceto pela região mais próxima do seu centro, onde existe um pico nos valores, o que pode criar uma empecilho para o algoritmo a ser testado, e assim este acabar não encontrando o mínimo global localizado mais ao seu centro.

Equação:

$$f(x) = 0.5 + \frac{\sin^2(x_1^2 - x_2^2) - 0.5}{[1 + 0.001(x_1^2 + x_2^2)]^2} \tag{2.3}$$




(a) Schaffer2 (b) Rosenbrock

Figura 12 – Gráficos das funções de teste *Schaffer2* e *Rosenbrock*.

Fonte: Gavana (2013)

Dimensões: 2

Mínimo gloal: $f(0, 0) = 0$

Domínio: $x_i \in [-100, 100]$ para $i = 1, 2$

### 2.3.4 *Rosenbrock*

A superfície formada por essa função, assim como a função *Booth* (Figura 11a), forma um grande vale. Apesar disso, consiste numa função consideravelmente mais difícil de se otimizar. O fato de ser uma função não convexa e possuir baixa inclinação são alguns dos fatores que dificultam a sua minimização

Equação:

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \tag{2.4}$$

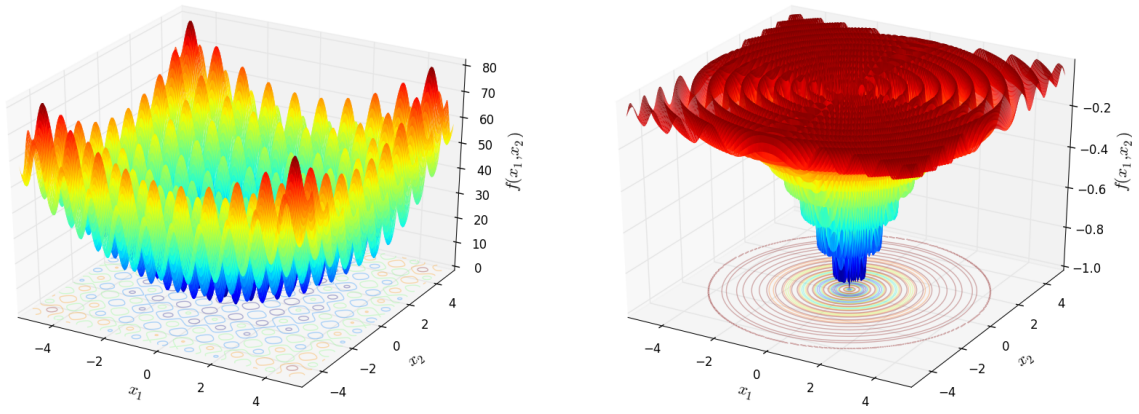
Dimensões: $n$

Mínimo global: $f(1, ..., 1) = 0$

Domínio: $x_i \in [-5, 10]$, para $i = 1, ..., n$

### 2.3.5 *Rastrigin*

Essa função é o primeiro exemplo que podemos ver de uma superfície que possui diversos mínimos globais em basicamente toda a sua extensão, o que ainda traz a dificuldade ao algoritmo de não se poder prender a nenhum desses mínimos, mas desta vez para toda a extensão da função.



(a) Rastrigin (b) DropWave

Figura 13 – Gráficos das funções de teste *Rastrigin* e *DropWave*.

Fonte: Gavana (2013)

Equação:

$$f(x) = 10n + \sum_{i=1}^{n}[x_i^2 - 10\cos(2\pi x_i)] \tag{2.5}$$

Dimensões: $n$

Mínimo global: $f(0, ..., 0) = 0$

Domínio: $x_i \in [-5.12, 5.12]$, para $i = 1, ..., n$

### 2.3.6 *DropWave*

Essa função se assemelha com a que foi exposta na subseção anterior, pela quantidade de mínimos locais que possui. Além disso, sua superfície ainda apresenta uma grande abertura próxima ao centro, que leva, ainda que não diretamente, ao seu mínimo global.

Equação:

$$f(x) = -\frac{1 + \cos\left(12\sqrt{x_1^2 + x_2^2}\right)}{0.5(x_1^2 + x_2^2) + 2} \tag{2.6}$$

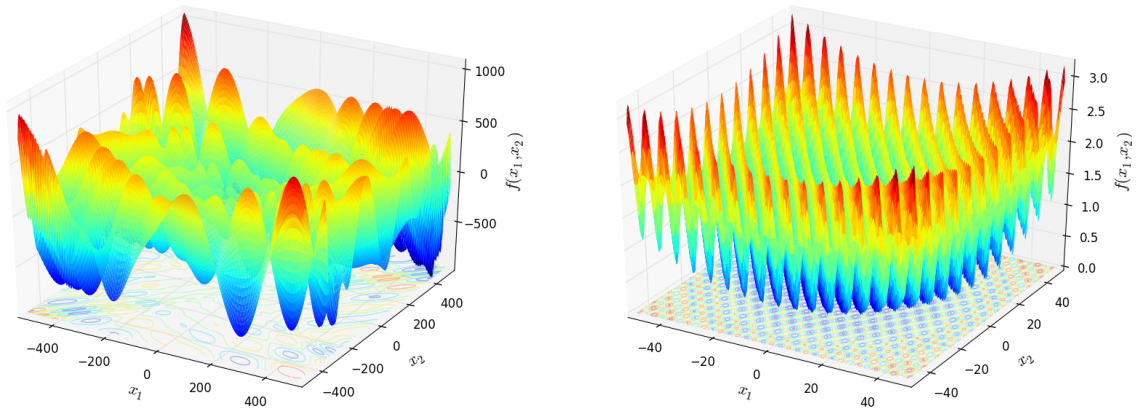
Dimensões: 2

Mínimo global: $f(0, 0) = -1$

Domínio: $x_i \in [-5.12, 5.12]$, para $i = 1, 2$

### 2.3.7 *Eggholder*

Essa é uma função consideravelmente difícil de se otimizar em função da sua superfície cheia de curvas acentuadas tendendo a mínimos locais. Além disso, seu domínio é consideravelmente maior do que o domínio das outras funções apresentadas.




(a) Eggholder (b) Griewank

Figura 14 – Gráficos das funções de teste *Eggholder* e *Griewank*.

Fonte: Gavana (2013)

Equação:

$$f(x) = -(x_2 + 47)\sin\left(\sqrt{\left|x_2 + \frac{x_1}{2} + 47\right|}\right) - x_1\sin\left(\sqrt{|x_1 - (x_2 + 47)|}\right) \tag{2.7}$$

Dimensões: 2

Mínimo global: $f(512, 404.2319) = -959.6407$

Domínio: $x_i \in [-512, 512]$, para $i = 1, 2$

### 2.3.8 *Griewank*

Essa função se assemelha à função *Eggholder* porque também possui diversos vales separados por espaços estreitos e possui o mesmo intervalo de domínio dela. Também constitui uma função de difícil otimização.

Equação:

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{2.8}$$

Dimensões: $n$

Mínimo global: $f(0, ..., 0) = 0$

Domínio: $x_i \in [-600, 600]$, para $i = 1, ..., n$

# 3 Paralelização de algoritmo genético com operador não convencional

A ideia do presente trabalho se originou do estudo de dois outros trabalhos: Ferreira (2017) e Pereira (2017). O primeiro tratou de analisar o impacto de diferentes políticas de migração simples e duplas em uma arquitetura de AG paralelo com múltiplas ilhas e, o segundo, da implementação e avaliação de diversos operadores genéticos não convencionais. Como as avaliações de ambos os trabalhos mostraram-se positivas, decidiu-se realizar uma implementação conjunta dessas duas arquiteturas: de um AG paralelo com um operador genético não convencional.

Tendo isso em vista, neste trabalho foi implementado um AG paralelo de granularidade grossa que utiliza um operador genético não convencional. Essa implementação daqui em diante será referida como algoritmo genético paralelo não convencional (**AGPN**). Como já foi dito antes, o único operador não convencional escolhido para compor esta implementação foi o de recombinação por transformação bacteriana, e a política de migração, PMDSAM. A linguagem de programação utilizada para desenvolver o AGPN foi C++.

Para efeito de comparação, foram também implementados e testados um AG sequencial convencional (**AGSC**), um AG sequencial com o operador não convencional (**AGSN**) e um AG paralelo com as mesmas características do já mencionado, com exceção do uso do operador não convencional (**AGPC**). Exceto as variações já mencionadas de cada implementação, os operadores e parâmetros do AG funcionam do mesmo jeito para cada uma delas: é utilizada representação de ponto-flutuante; seleção dos pais por torneio; a seleção dos sobreviventes se dá pela substituição dos piores elementos da população pelos filhos gerados, no momento da substituição; operador de cruzamento aritmético simples e duplo; mutação Gaussiana; condição de parada se dá quando o AG já finalizou a execução de todas as suas gerações.

O presente capítulo contém 3 seções: **Visão geral**, **Paralelização do AGPN** e **Operador genético não convencional no AGPN**. Essas seções expõem, respectivamente, uma descrição de aspectos gerais do projeto, além de uma explicação mais específica de como a arquitetura do projeto é construída tanto do ponto de vista da paralelização do AG quanto do uso do operador não convencional.

## 3.1 Visão geral

Como mencionado logo antes, a implementação contida neste trabalho é baseada em conceitos e implementações de dois outros trabalhos. Mais que isso, aqui são aproveitadas noções de duas subáreas distintas dentro da computação evolucionária: o uso de arquiteturas paralelas e de operadores genéticos não convencionais em algoritmos genéticos. Na Figura 15, pode ser visto um diagrama que resume onde este trabalho se encontra em relação às subáreas mencionadas.

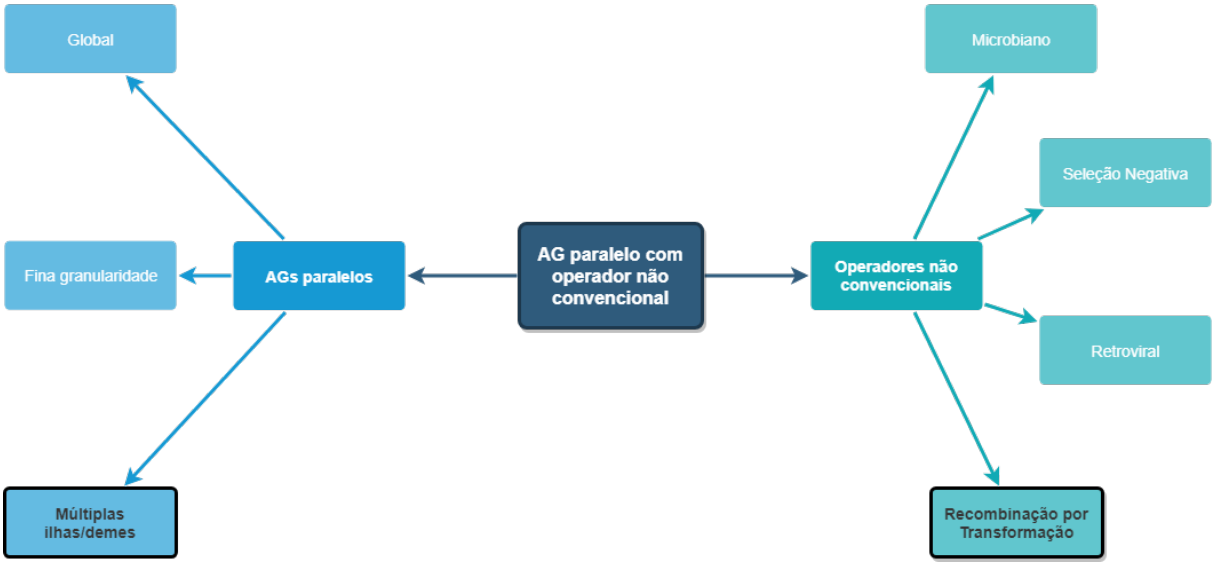



Figura 15 – Diagrama que ilustra as subáreas da computação evolucionária exploradas neste trabalho.

Fonte: elaborado pelo autor

Quanto à implementação, a linguagem C++ foi escolhida por ser eficiente, flexível e muito usada para os mais diversos tipos de projetos. Além disso, a biblioteca de computação paralela `OpenMP` foi escolhida por ser fácil de se integrar a um projeto e fornecer abstrações poderosas para a criação e uso de várias *threads* em um programa (no caso, para a paralelização do AG).

Na Figura 16, pode ser visto o diagrama de classes do projeto desenvolvido.

As classes principais são `Populacao` e `Cromossomo`. Para implementar o operador de recombinação por transformação bacteriana, essas classes foram estendidas, respectivamente, nas subclasses `PopulacaoTransformacao` e `CromossomoMorto`. A funcionalidade de migração é implementada na classe `Migracao`. As funcoes de teste são representadas por objetos da classe `Funcao`, e definidas no módulo `Funcoes`, para que no programa principal todas as funções definidas nele sejam automaticamente importadas e utilizadas nas

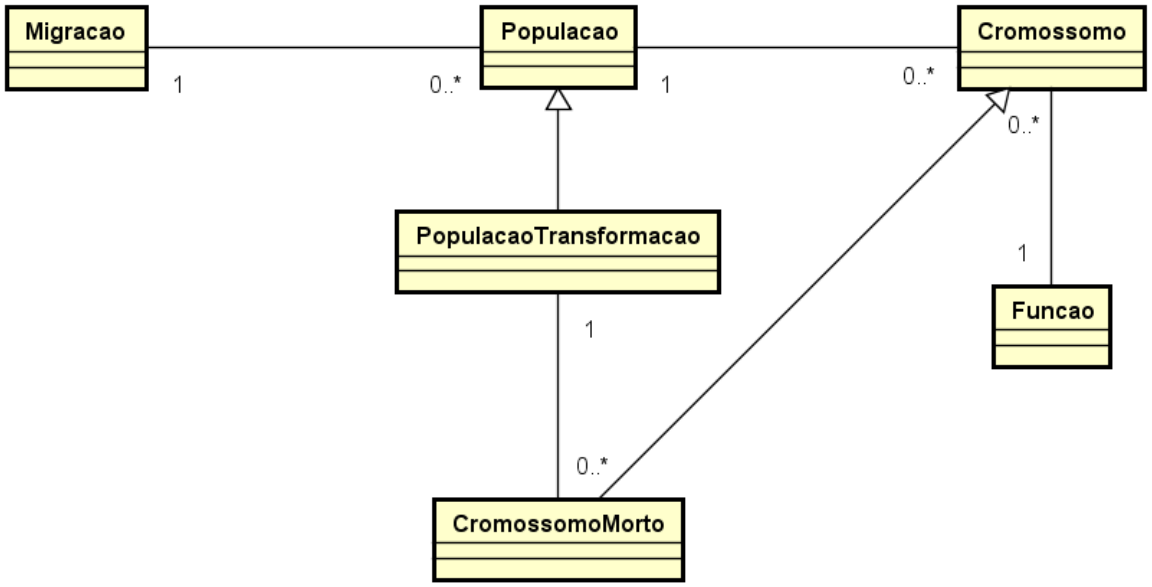



Figura 16 – Diagrama de classes do AGPN.

Fonte: elaborado pelo autor

execuções dos algoritmos. Por fim, o módulo `Algoritmos` é o que possui as implementações completas dos algoritmos. Nas duas próximas seções, o funcionamento do algoritmo será explicado com mais detalhes.

## 3.2 Paralelização do AGPN

Para a paralelização utilizada dentro da implementação do AGPN, procurou-se reaproveitar o máximo possível de Ferreira (2017): como já foi mencionado antes neste trabalho, um dos objetivos é avaliar o comportamento de um operador não convencional aplicado a uma arquitetura paralela em um AG. Nesse contexto, a arquitetura paralela em questão é em boa parte inspirada na do trabalho mencionado acima. Lá, foi utilizado um AG paralelo com múltiplas subpopulações que se comunicam através de uma política de migração. Além disso, foram analisadas diversos tipos de tais políticas, das quais a PMDSAM foi escolhida como uma das melhores alternativas, tanto dentro das políticas de migração simples quanto duplas. Por essa razão, como também já fora antes mencionado, será a única política escolhida para ser utilizada no presente trabalho. Algumas mudanças, no entanto, foram realizadas quanto à arquitetura paralela do presente trabalho com a apresentada em Ferreira (2017). Essas mudanças serão exploradas com mais detalhes ao longo desta seção.

Ademais, a topologia escolhida foi a de um grafo não direcionado completo, onde

cada população é um nó que pode trocar indivíduos com qualquer outra subpopulação. Como o foco do trabalho está mais alinhado em avaliar uma implementação de um AG paralelo com múltiplas ilhas com o operador de recombinação por transformação bacteriana, e não exatamente avaliar outros aspectos do AG como a disposição das ilhas, essa escolha é razoável. Essa topologia é simples e não implica em grandes exigências para a implementação da política de migração, além de também ter sido utilizada em Ferreira (2017), o que ajuda na comparação entre os trabalhos e os seus resultados. Um problema que pode aparecer de uma topologia densamente conectada como essa é a de grandes custos de comunicação entre as populações, como é mencionado em Cantú-paz (1999). No entanto, pelo projeto ter sido feito pensando na aplicação para uma arquitetura de único processador, ao menos de início, isso não foi um problema. Isso poderia ser preocupante se a aplicação funcionasse para arquiteturas como a de um sistema com múltiplos processadores, ou de uma rede de computação distribuída.

Em poucas palavras, o AGPN executa paralelamente várias populações concorrentemente. Cada uma das subpopulações evolui de maneira isolada, até que o processo de migração ocorre. O operador de migração é executado por uma população arbitrária quando esta já finalizou o *loop*, como modo de haver um mínimo controle na execução desse operador, sem que este deixe de ser assíncrono. No entanto, as outras populações podem estar a qualquer momento do funcionamento do seu próprio processo evolutivo. Por isso, esse operador é considerado assíncrono. Ao contrário do que acontece em Ferreira (2017), o operador de migração não é inicializado antes do AG principal começar a funcionar e, então, funciona em uma única *thread* repetidamente. Essa foi a abordagem inicial, mas depois foi modificada por perceber-se que esse operador ser executado muitas vezes, mesmo com o uso de uma probabilidade baixa de migração. O seu funcionamento básico foi alterado para que uma das populações fosse responsável por executar esse operador ao fim do seu laço principal. Na Figura 17, pode-se ver o pseudocódigo que descreve o funcionamento geral do AG paralelo. Até a linha 10, são feitas as inicializações de cada população e de variáveis de controle. Depois, para cada população, ocorre o processo de evolução do AG, de maneira isolada e paralela. Nessa parte se destacam as chamadas aos métodos do operador não convencional (linha 20) e do operador de migração (linha 26). É importante notar que a migração é apenas chamada por uma população específica (como é comentado no pseudocódigo), mas que isso não implica que esta mesma população participe no processo de migração. É pelo operador de migração que é feita a seleção das populações que irão trocar indivíduos.

O paralelismo implementado é gerenciado por meio da API do `OpenMP`. A sua interface permite a paralelização de trechos de código com o uso de uma série de cláusulas aplicadas por diretivas de pré-processamento, além de um conjunto de rotinas complementares expostas pela biblioteca `omp.h`. Isso facilita a paralelização de algoritmos, pois muitas vezes é necessário apenas aplicar uma certa cláusula no topo de um trecho de

```
para cada subpopulação p
        inicializarPopulacao(p)
        calcularFitness(p)
fimpara

nPopulacoesProcessadas := 0
N_POPULACOES := 10

para cada subpopulação p
        enquanto (condicao de parada):
                selecaoTorneio(p) // selecionar pais por torneio
                filhos := gerarFilhos(p)
                /*
                 * seleciona sobreviventes dentro dos filhos e adiciona pior indivíduo
                 * da subpopulação à população de mortos
                 */
                selecionarSobreviventes(p,filhos) //
                calcularFitness(p)
                recombinacaoTransformacao(p) // realizar recombinação por transformação

                /*
                 * cálculo feito para ter certeza de que uma única população dentro das
                 * subpopulações vai chamar o operador de migração
                 */
                se idPopulacao % N_POPULACOES == 9
                        operadorMigracao.realizarMigracao(nPopulacoesProcessadas)
                fimse
        fimenquanto

        nPopulacoesProcessadas := nPopulacoesProcessadas + 1
fimpara
```

Figura 17 – Pseudocódigo do AG paralelo AGPN.

Fonte: elaborado pelo autor

código para que este seja executado de maneira paralela, sem maiores modificações na sua estrutura. Existe, por exemplo, a cláusula `parallel`, que indica que para o trecho de código indicado será alocado um conjunto de *threads* e então este trecho será executado para cada uma delas. Outra cláusula útil que é utilizada no presente trabalho é a `for`, que paraleliza automaticamente um laço do tipo *for*. Na Figura 18, é demonstrada a estrutura de um trecho de código que utiliza essas duas cláusulas no momento de executar o laço principal de evolução dos AGs para cada população, paralelamente.

Para implementar o processo de migração assíncrono e não deixar isso afetar a evolução de cada subpopulação, alguns modos de controle de estado precisaram ser implementados na lógica das populações e do operador de migração em si. Após o processo de migração ser iniciado por uma população e de as duas populações que participarão desse processo serem escolhidas, o próprio objeto de migração manda uma mensagem para os objetos das populações, para que o valor da variável booleana `migracao` seja modificada para `verdadeiro`. Da mesma maneira, quando o processo de migração finaliza, outra mensagem é enviada para os objetos das populações para reverter essa modificação e alterar o valor da mesma variável de volta para `falso`.

```
#pragma omp parallel
{
    #pragma omp for
    for (unsigned int i = 0;i < N_POPULACOES;++i){
        Populacao &p = *(populacoes[i]);

        int j = 0;
        do {
            //Execução de geração do AG para uma população
            // ...
        } while (++j < nGeracoes);
        /*
         * Restante do código executado quando
         * processamento das gerações finaliza
         */
        // ...
    }
}
```

Figura 18 – Estrutura de trecho de código que executa as populações em paralelo por meio de cláusulas do `OpenMP`.

Fonte: elaborado pelo autor

É importante salientar que, apesar do número de *threads* ser um dos parâmetros definidos no algoritmo, não há um controle no sentido de que um número de *threads* deve ser alocado para cada população, que é como acontece, por exemplo, em Ferreira (2017). Esse gerenciamento é feito automaticamente e de maneira transparente pela biblioteca `OpenMP`, o que facilita a implementação do algoritmo.

Além disso, os métodos dos operadores dentro do AG verificam se a variável `migracao` possui o valor verdadeiro. Se sim, ela fica em um laço até que este valor mude. Isso previne que o valores de cromossomos sejam modificados enquanto o processo de migração está sendo executado, e que da mesma maneira o processo de migração interfira nos operadores genéticos do algoritmo. Essa questão em específico precisa ser tratada com cuidado, pois pode ser a causa de problemas de *deadlock*. Isso não acontece para os parâmetros e configuração atual utilizados nos testes do AGPN, mas para números de *threads* diferentes, por exemplo, esse tipo de problema foi presente.

Tendo isso em vista e além do que já foi explicado, o funcionamento do processo de migração pode ser resumido em:

- O operador de migração é chamado por alguma subpopulação por meio do método *Migracao::realizarMigracao*;
- Se um número aleatório é menor que a probabilidade de migração, ela é executada;

- Populações têm o valor das suas variáveis booleanas `migracao` atribuídos como `verdadeiro`
- O algoritmo entra em um laço até que duas subpopulações que não estejam ocupadas e cujo processamento das suas gerações ainda não tenha sido finalizado tenham sido selecionadas aleatoriamente para participar do processo de migração;
- Os indivíduos que serão migrados entre essas subpopulações também são selecionados aleatoriamente;
- A troca dos indivíduos é realizada;
- Populações têm o valor das suas variáveis booleanas `migracao` atribuídos como `falso`.

## 3.3 Operador genético não convencional no AGPN

Como já mencionado anteriormente, o AGPN faz proveito do operador de recombinação por transformação introduzido e avaliado como um bom operador genético não convencional em Pereira (2017), pois é demonstrado que este apresenta bons resultados quando comparado a outros operadores não convencionais. Aspectos gerais do funcionamento do AG listados no início deste capítulo (como tipo dos operadores de seleção e mutação) foram aproveitados desse trabalho, assim como o funcionamento do operador propriamente dito e os seus parâmetros específicos: tamanho da população de mortos e o número máximo de indivíduos que serão infectados no processo de infecção. O que o presente trabalho propõe é uma implementação deste mesmo operador sendo utilizado dentro da arquitetura de um AG paralela com múltiplas ilhas. Além disso, a própria comparação de implementações de AGs não convencionais com AGs paralelos em geral (sejam eles convencionais ou não), pode ser considerada como uma adição ao trabalho anterior, pois isso trás mais informações de como essas duas classes de AGs se comportam quanto, por exemplo, à convergência dos indivíduos. Esses detalhes serão mais explorados no próximo capítulo.

A integração do operador não convencional na arquitetura paralela deste trabalho não foi complexa: como exposto na Figura 16, ela basicamente se resume ao uso das classes `PopulacaoTransformacao` e `CromossomoMorto`, que são as responsáveis por adicionar a lógica (uso da população de mortos e o processo de infecção em si) do operador não convencional às classes Populacao e Cromossomo. Assim, para que este operador seja utilizado, basta utilizar as duas classes derivadas ao invés das suas classes base. Além disso, é necessário chamar o método *PopulacaoTransformacao::recombinacaoTransformacao*, que é responsável pelo funcionamento do operador não convencional, em si, ao fim do laço principal do AG. Isso pode ser visualizado na Figura 17.

# 4 Resultados

Como foi explorado com maiores detalhes no capítulo anterior, o AGPN utiliza o operador não convencional de recombinação por transformação bacteriana dentro da arquitetura de um AG paralelo. De forma resumida, esse operador é utilizado dentro de cada ilha de uma estrutura de AG paralelo com populações que se comunicam pela ininterruptapolítica de migração assíncrona PMDSAM. Essa arquitetura conjunta trouxe bons resultados se comparada às outras implementações testadas no trabalho. A seguir, nas próximas seções deste capítulo é exposto de maneira detalhada, além dos resultados propriamente ditos, como esses dados foram obtidos e organizados a partir dos testes realizados.

## 4.1 Configuração e parâmetros utilizados

Para a realização dos testes e posterior avaliação do desempenho do AGPN, foi preciso definir detalhes de configuração e os valores dos parâmetros utilizados nas implementações deste trabalho. Ao longo dessa seção, esses detalhes serão expostos com mais detalhes.

Todos os algoritmos implementados foram testados com todas as funções de testes descritas no último capítulo. Parte do comportamento dos operadores utilizados nesses algoritmos pode ser visto na Tabela 1.

Tabela 1 – Descrição dos operadores utilizados no AG

| Característica/Operador | Descrição |
|---|---|
| Seleção | Torneio |
| Número de indivíduos no torneio | 2 |
| Número máximo de pares de pais selecionados pelo torneio | 10 |
| Número de filhos que vão substituir indivíduos na população | 5 |
| Migração (para AGs paralelos) | PMDSAM |

Fonte: elaborado pelo autor

Os parâmetros de probabilidade de mutação, cruzamento e migração podem ser visualizados na Tabela 2. Na Tabela 3, os parâmetros restantes dos AGs implementados são listados, como por exemplo número de indivíduos de uma população e valor da margem de erro utilizado em relação ao valor ótimo de cada função de teste.

Como efeito de compraração com Pereira (2017) e Ferreira (2017), os parâmetros utilizados no presente trabalho foram, quase que em sua maioria, reaproveitados desses trabalhos. Dos valores da Tabela 1, por exemplo, todos exceto a política de migração

Tabela 2 – Valores de probabilidade utilizados para os operadores do AG

| Probabilidade | Valor |
|---|---|
| Probabilidade da mutação | 0.05 |
| Probabilidade de cruzamento | 0.9 |
| Probabilidade de migração | 0.003 |

Fonte: elaborado pelo autor

Tabela 3 – Parâmetros gerais

| Parâmetro | Valor |
|---|---|
| Margem de erro | 0.001 |
| Condição de parada | Finalizar execução de todas as gerações |
| Número de execuções | 30 |
| Número de *threads* | 4 |
| Número de gerações | 1000 |
| Número de indivíduos da população | 100 |
| Número subpopulações (para AGs paralelos) | 10 |
| Média para gerar números gaussianos | 0 |
| Desvio padrão para gerar números gaussianos | 1.55 |
| Número máximo de indivíduos a infectar* | 30 |
| Tamanho da população de mortos* | 30 |

* - valores utilizados no operador não convencional de recombinação por transformação bacteriana

Fonte: elaborado pelo autor

escolhida foram retirados do primeiro. Os valores de probabilidades de mutação e cruzamento (Tabela 2), além dos valores de margem de erro, condição de parada, número de execuções, número de gerações, número de indíviduos da população, taxa de infecção e tamanho da população de mortos (Tabelas 1 e 3), também foram retirados do mesmo trabalho. Isso foi feito para que os resultados das implementações dos AGs desse trabalho que utilizam o operador não convencional não desviassem muito dos resultados do trabalho que o introduz. Os parâmetros de política de migração em si, probabilidade de migração e número de subpopulações foram retirados de Ferreira (2017), como ponto de partida e efeito de comparação.

## 4.2 Formato dos resultados obtidos

Para a coleta dos resultados, cada um desses algoritmos foi executado **30** vezes para cada função de teste. Após essas execuções, os resultados são agrupados e escritos em arquivos de texto simples: um arquivo contendo dados de convergência, mais especificamente, que contém a média e desvio padrão das execuções dos dados de melhor e pior valores de *fitness* para cada geração do algoritmo. Da mesma maneira, são calculadas as médias e desvios padrão dos valores médios de *fitness* em cada geração. Ademais, existe outro

arquivo de saída que contém um resumo das execuções, para cada função de teste, com tabelas escritas em texto simples. Mais a frente nesse capítulo o formato desses resultados serão tratados com mais detalhes.

Para cada rodada de execuções os dados de convergência do algoritmo são obtidos, em conjunto com os resultados resumidos sobre o seu desempenho para uma função de teste específica. Dos dados de convergência são obtidos os gráficos de convergência, que nos fornecem uma melhor visualização do processo de convergência do algoritmo, dado um intervalo de gerações. Mais especificamente, são obtidos esses dados brutos para uma execução do algoritmo, e ao fim de todas as execuções são calculadas as médias e desvios padrão para cada geração. Esses dados são os que são de fato escritos nos arquivos de saída do programa, e que depois são utilizados para gerar os gráficos de convergência.

Os resultados resumidos indicam de forma sucinta como foi o desempenho do algoritmo para uma função de teste. Os critérios de avaliação, dos quais cada um é utilizado em uma única tabela, são: médias dos melhores, médias e piores valores de *fitness* de todas as execuções para a última geração; melhor e pior valores de *fitness* considerando a execução de todo algoritmo, além da média de todos os valores de *fitness* processados pelo algoritmo; Por fim, o número de vezes que o algoritmo convergiu para o mínimo global da função (dentro de uma margem de erro especificada) nas suas execuções, e a média das gerações em que cada uma dessas convergências aconteceram. O algoritmo ou algoritmos que apresentaram melhor desempenho para o critério de avaliação de cada uma das tabelas serão marcados em negrito, por questão de maior facilidade de visualização e interpretação.

Foram escolhidas 4 funções cujos resultados serão avaliados em seguida: *Schaffer02*, *Rosenbrock*, *Rastrigin* e *Griewank*. A escolha dessas funções foi feita em razão de todas elas, com exceção da segunda, possuírem vários mínimos locais, o que põe à prova a capacidade do algoritmo de encontrar o mínimo global apesar disso. A superfície gerada pela função Rosenbrock, por outro lado, possui um formato de vale. No entanto, pelos resultados avaliados em Pereira (2017), mostrou-se ser uma função de difícil otimização. Pelos mesmos resultados, a otimização da função Schaffer02 não se mostrou de grande dificuldade, e por isso foi escolhida, como forma de testar se a convergência dos algoritmos estava ocorrendo como esperado.

## 4.3 Análise de desempenho do AGPN

Na presente seção os resultados obtidos dos testes das implementações feitas serão efetivamente expostos. A partir disso o desempenho do AGPN será avaliado em comparação com os resultados dos outros AGs implementados. Nas próximas duas subseções, serão avaliadas respectivamente as características de convergência, e de robustez e precisão do

algoritmo. A sua robustez indica o quão bom é o desempenho do algoritmo em diferentes cenários ou, no caso do presente trabalho, para diferentes funções de teste. A precisão, por outro lado, permite avaliar a divergência das soluções encontradas pelo algoritmo se comparadas aos ótimos globais dessas funções.

### 4.3.1 Análise de convergência do AGPN

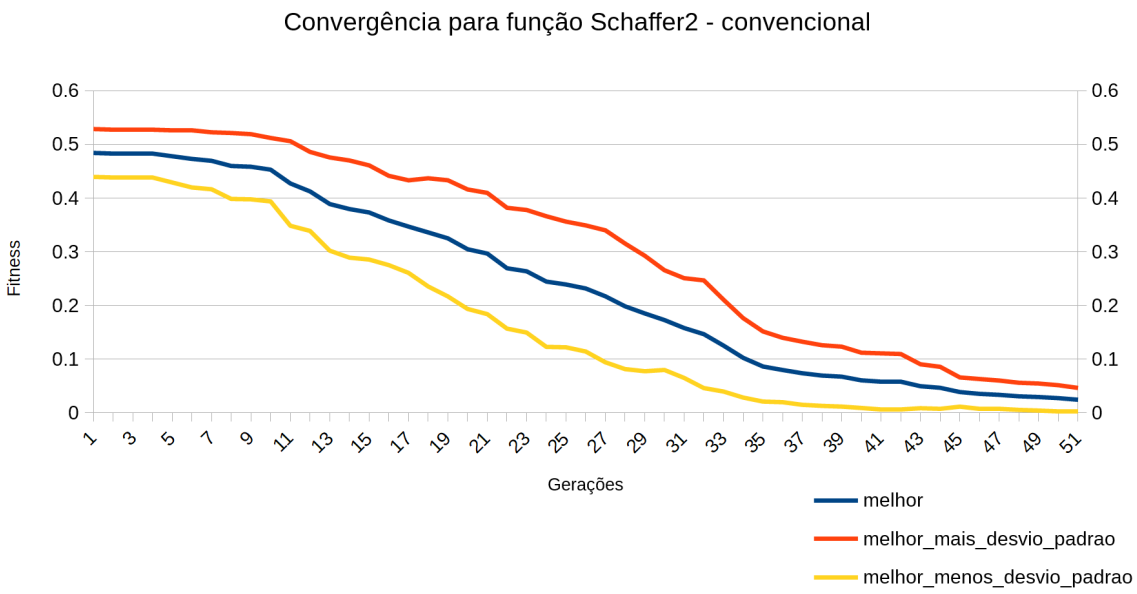
Nas Figuras 20, 22, 24, e 26, podem ser vistos os gráficos de convergência de todas as implementações para cada função de teste. As duas figuras superiores a cada grupo dizem respeito aos AGs sequenciais (AGSC e AGSN), e as inferiores, aos paralelos (AGPC e AGPN). Os gráficos mostram no eixo $x$ as gerações do algoritmo e no eixo $y$ os valores de média do melhor *fitness* em todas as execuções para cada geração, além desse valor somado e subtraído do desvio padrão correspondente calculado. Vale ressaltar que os gráficos não mostram todas as 1000 gerações pelas quais os AGs executaram por uma questão de simplicidade das análises, já que a partir de uma certa geração o AG converge e o valores de *fitness* tendem a se repetir. Nas Figuras 22b e 24b, por exemplo, o tamanho das séries do eixo $x$ são, respectivamente, 20 e 25, porque as execuções relativas a esses gráficos convergeram pouco antes disso.

Por meio dos gráficos apresentados, é possível ver que o AGPN se sai bem quando comparado às outras implementações feitas no trabalho, tanto sequenciais quanto paralelas. Percebe-se também que a curva de convergência é ligeiramente mais íngreme que as dos gráficos correspondentes ao AG paralelo convencional, com exceção dos resultados da função *Rosenbrock*. Isso pode ser visto na Figura 22b, que ilustra o gráfico de convergência dessa função para o AG paralelo não convencional em comparação com o gráfico da Figura 22a, que mostra o mesmo para o AG paralelo convencional. Além disso, é visível a melhora na velocidade de convergência, principalmente se comparado com os AGs sequenciais apresentados, assim como em relação ao AG paralelo convencional, mesmo que nesse caso geralmente a diferença não seja tão acentuada.

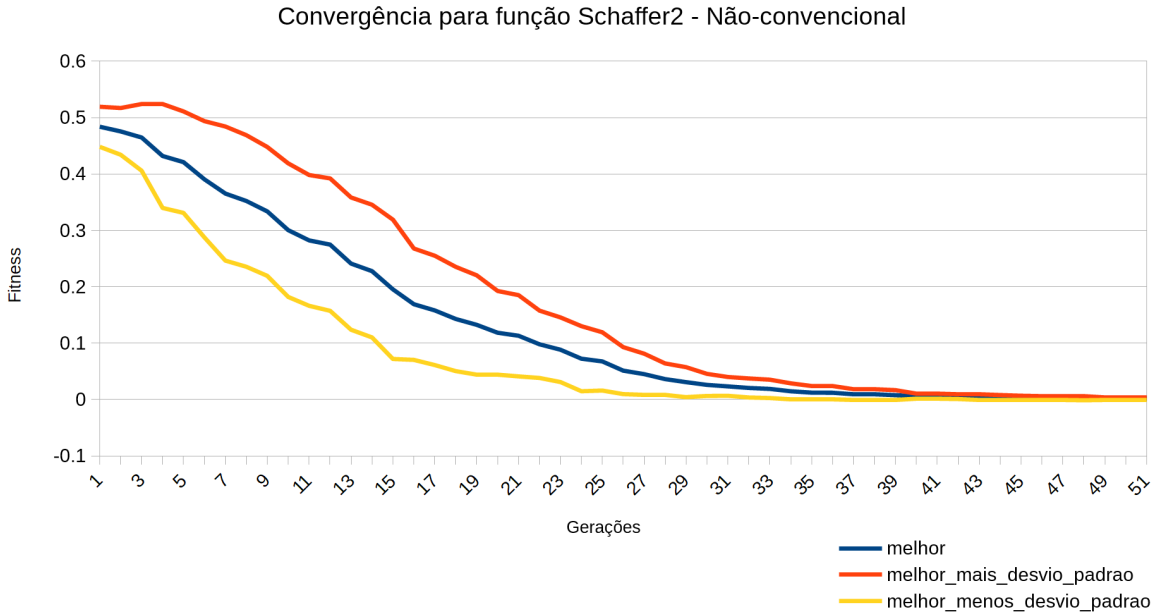
A Tabela 4 mostra as médias de gerações em que cada AG encontrou o mínimo global. Os resultados dessa tabela mostram que o AGPN se saiu melhor do que todas as outras implementações nesse critério de avaliação, para todas as funções de teste em questão. Como já tinha sido indicado pelos gráficos de convergência apresentados na subseção anterior, sua velocidade de convergência nos testes feitos é melhor que as dos outros AGs para as mesmas funções de teste.

### 4.3.2 Análise da robustez e precisão do AGPN

A Tabela 5 mostra os melhores valores de *fitness* encontrados de cada função, considerando todas as 30 execuções feitas. Assim pode-se ver se cada AG encontra o



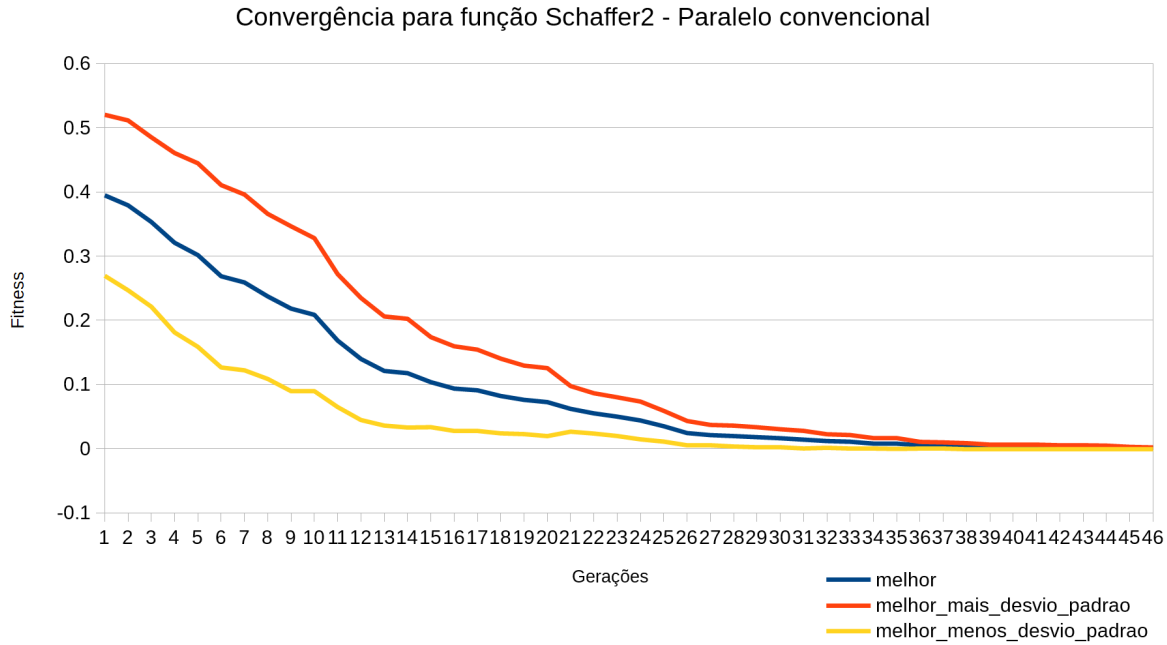

(a) Resultado da função *Schaffer02* para AG convencional sequencial



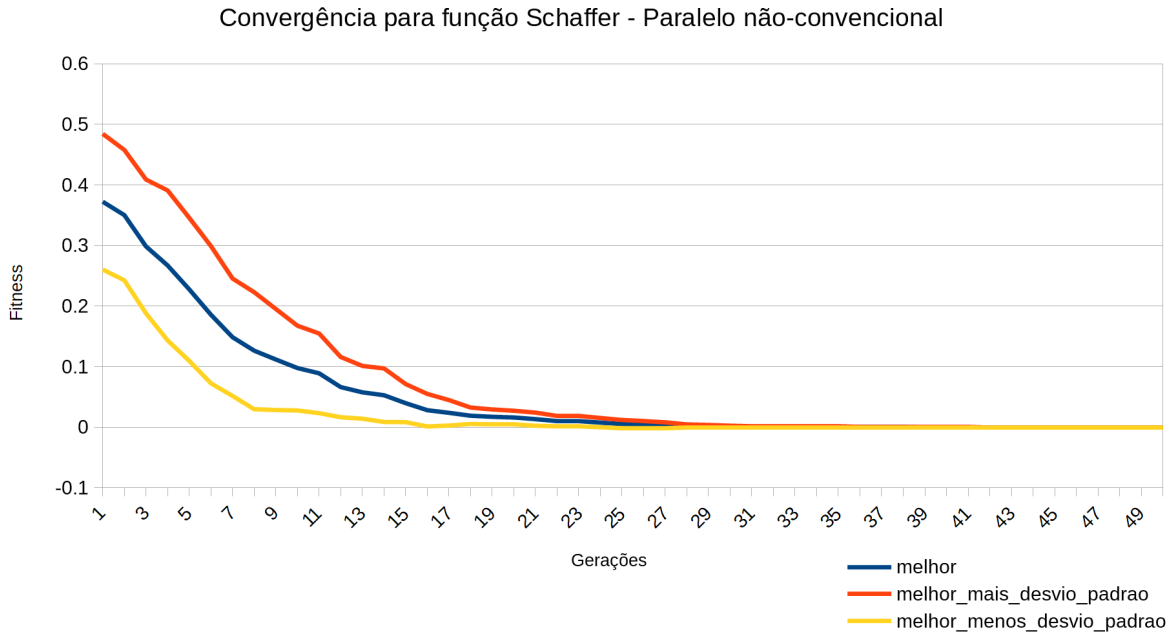

(b) Resultado da função *Schaffer02* para AG não convencional sequencial

Figura 19 – Gráficos de convergência da função de teste *Schaffer02* para AGs sequenciais.

Fonte: elaborado pelo autor



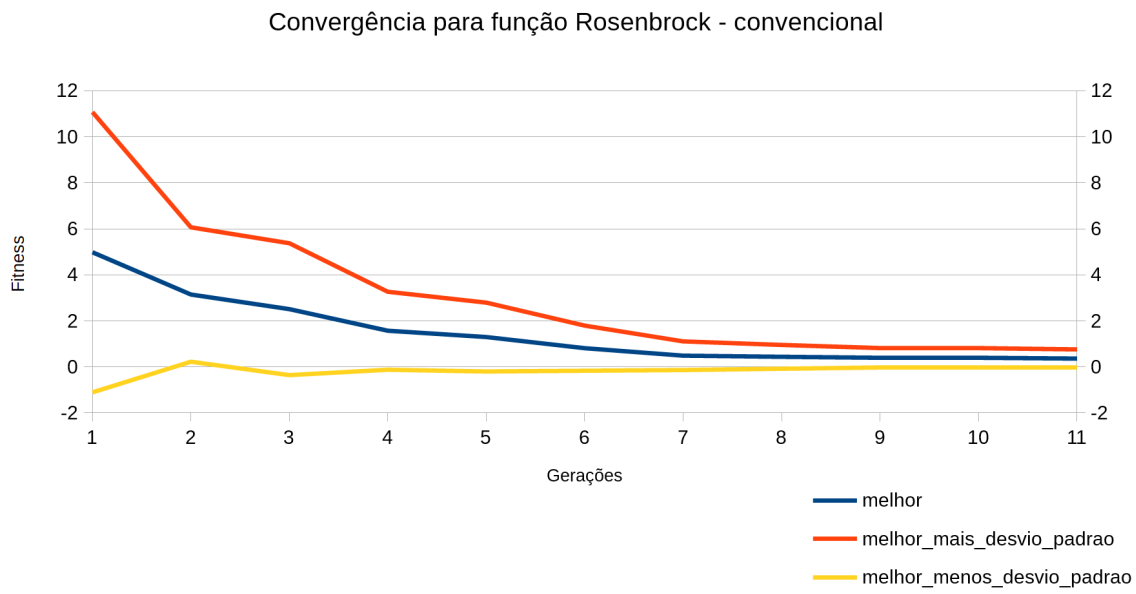

(a) Resultado da função *Schaffer02* para AG paralelo convencional



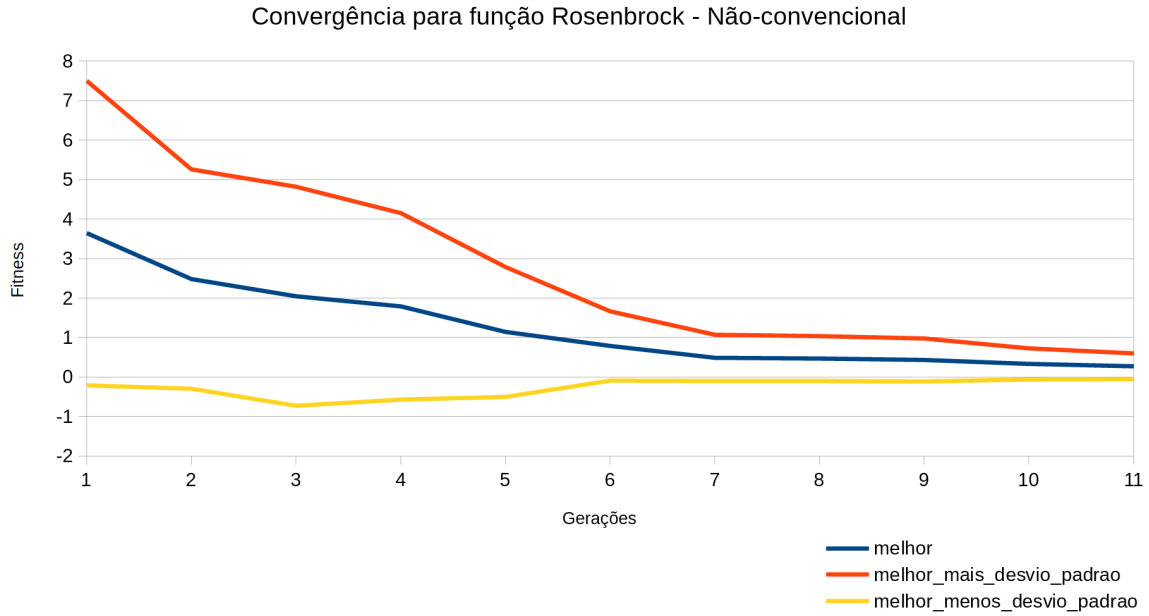

(b) Resultado da função *Schaffer02* para AG paralelo não convencional

Figura 20 – Gráficos de convergência da função de teste *Schaffer02* para AGs paralelos.

Fonte: elaborado pelo autor



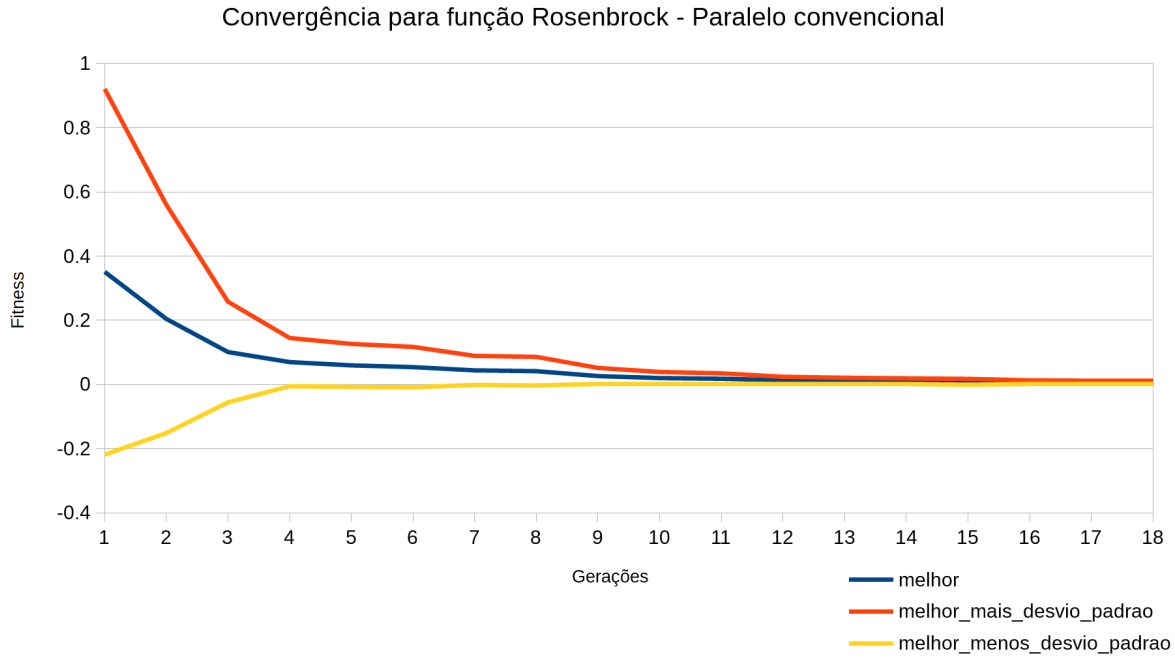

(a) Resultado da função *Rosenbrock* para AG convencional sequencial



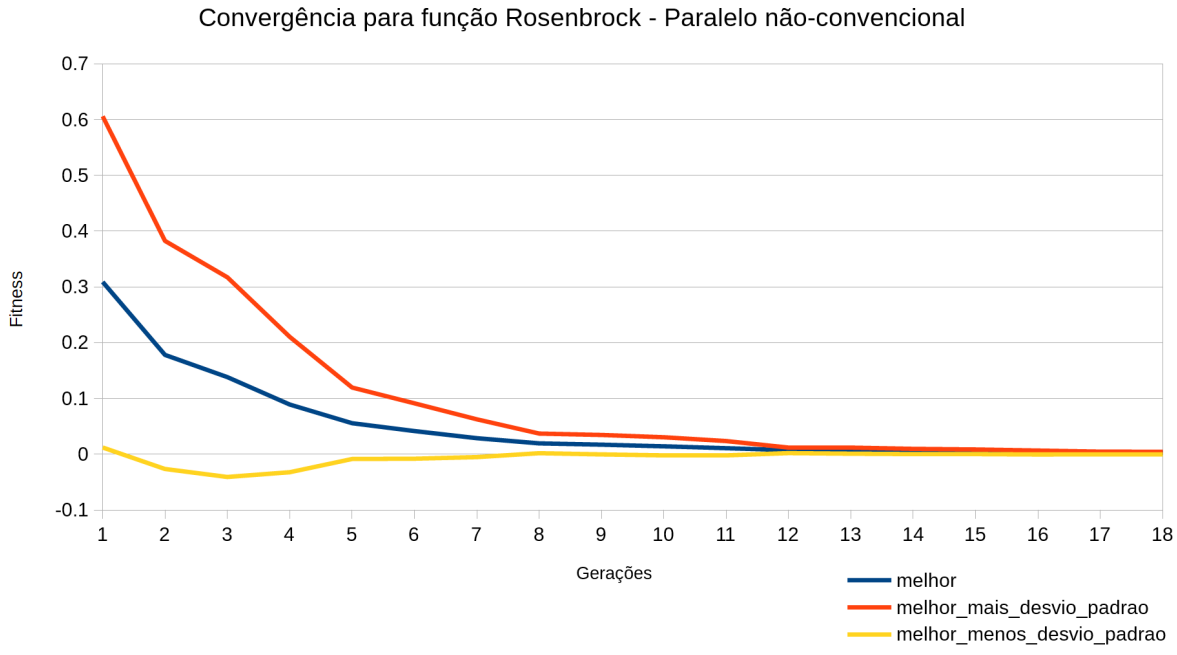

(b) Resultado da função *Rosenbrock* para AG não convencional sequencial

Figura 21 – Gráficos de convergência da função de teste *Rosenbrock* para AGs sequenciais.

Fonte: elaborado pelo autor



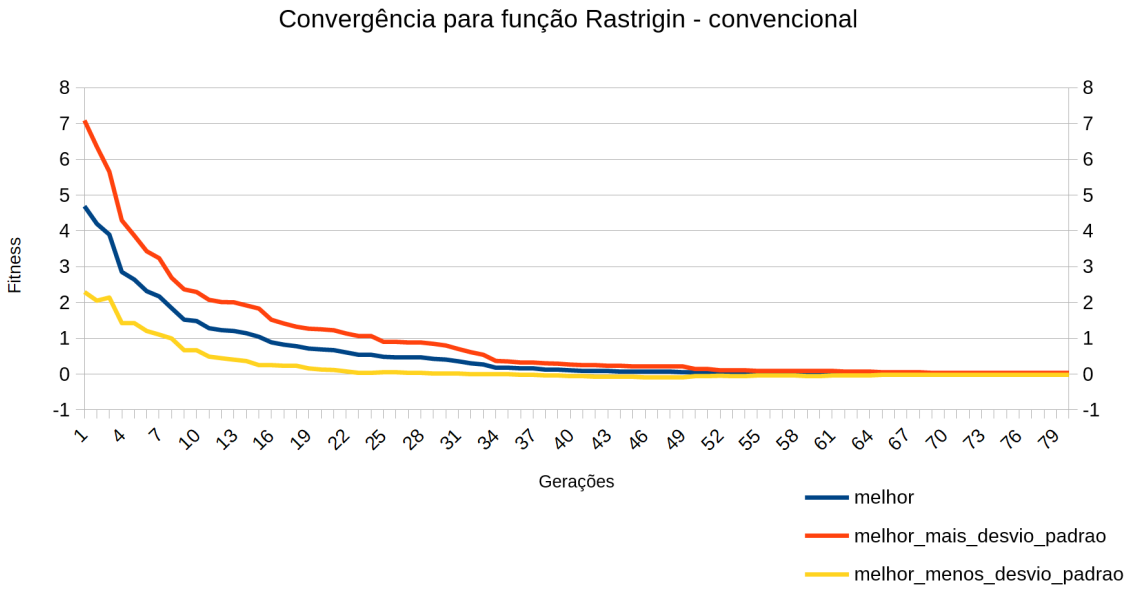

(a) Resultado da função *Rosenbrock* para AG paralelo convencional



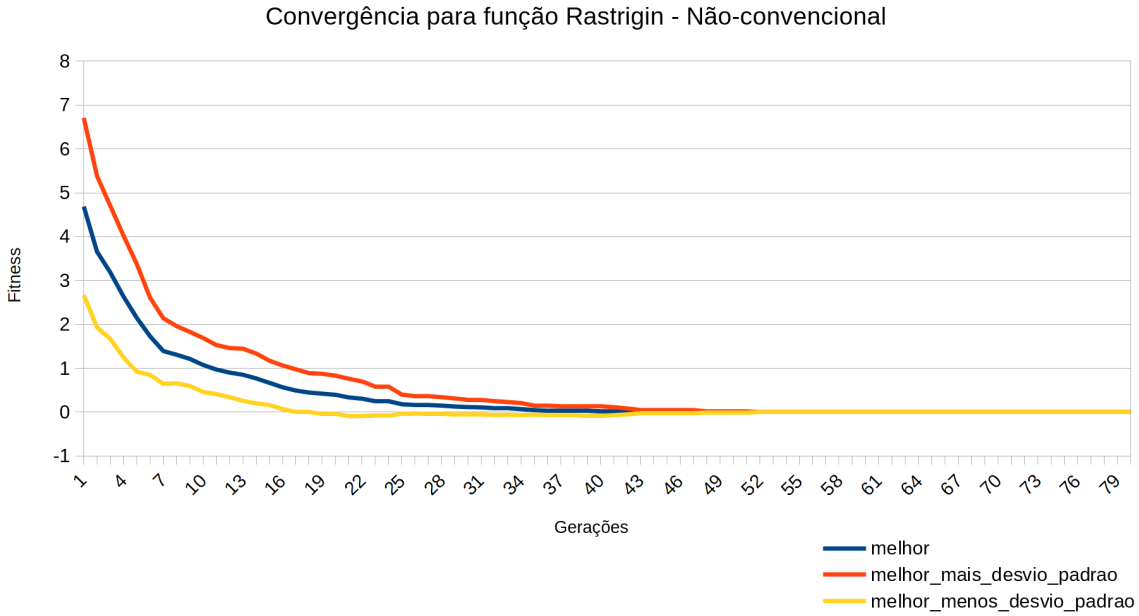

(b) Resultado da função *Rosenbrock* para AG paralelo não convencional

Figura 22 – Gráficos de convergência da função de teste *Rosenbrock* para AGs paralelos.

Fonte: elaborado pelo autor



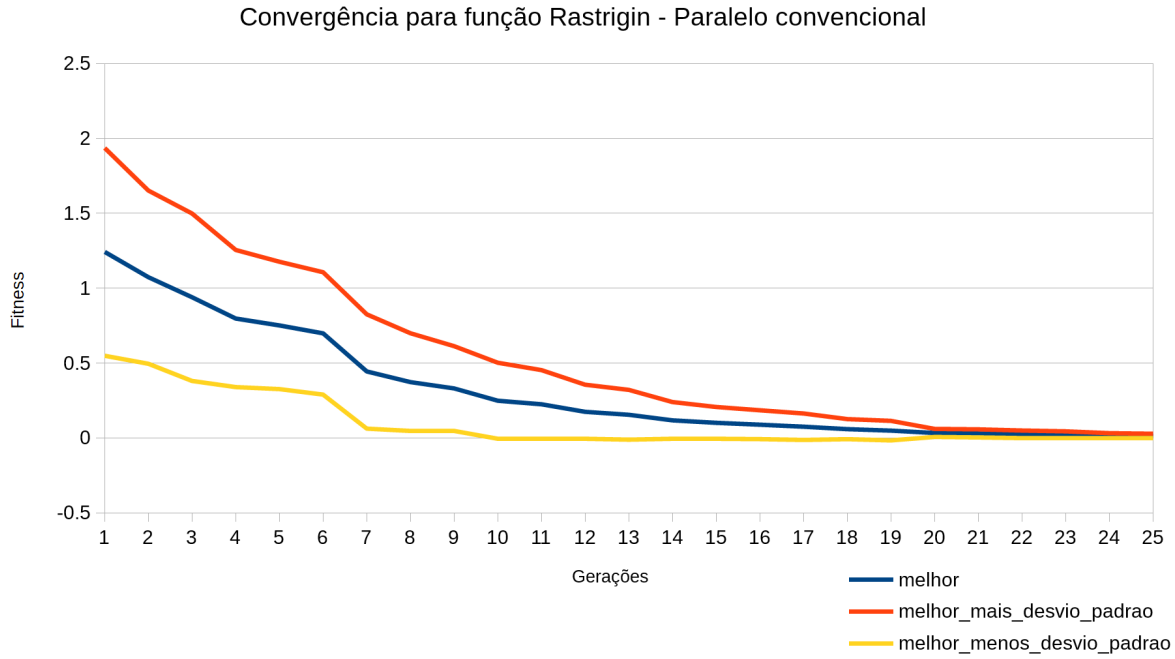

(a) Resultado da função *Rastrigin* para AG convencional sequencial



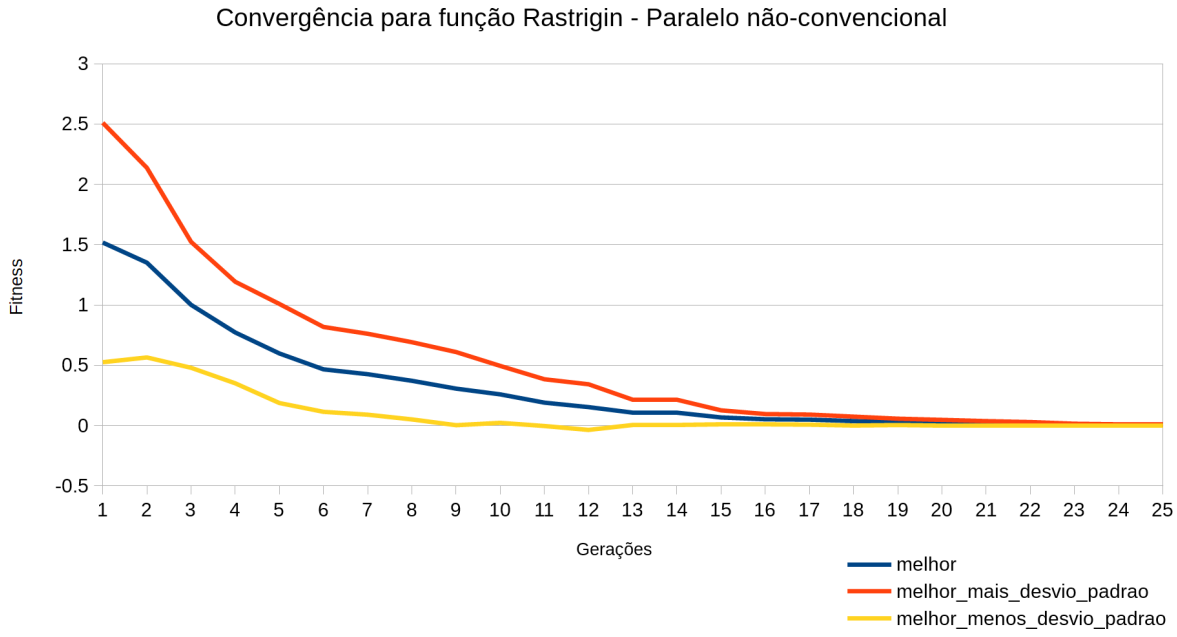

(b) Resultado da função *Rastrigin* para AG não convencional sequencial

Figura 23 – Gráficos de convergência da função de teste *Rastrigin* para AGs sequenciais.

Fonte: elaborado pelo autor



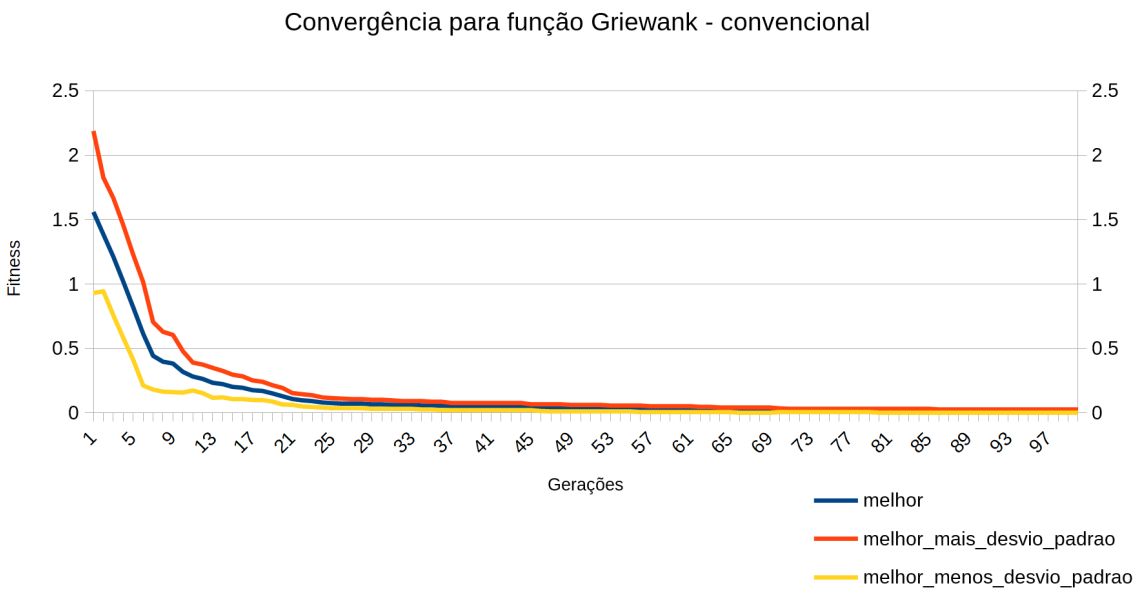

(a) Resultado da função *Rastrigin* para AG paralelo convencional



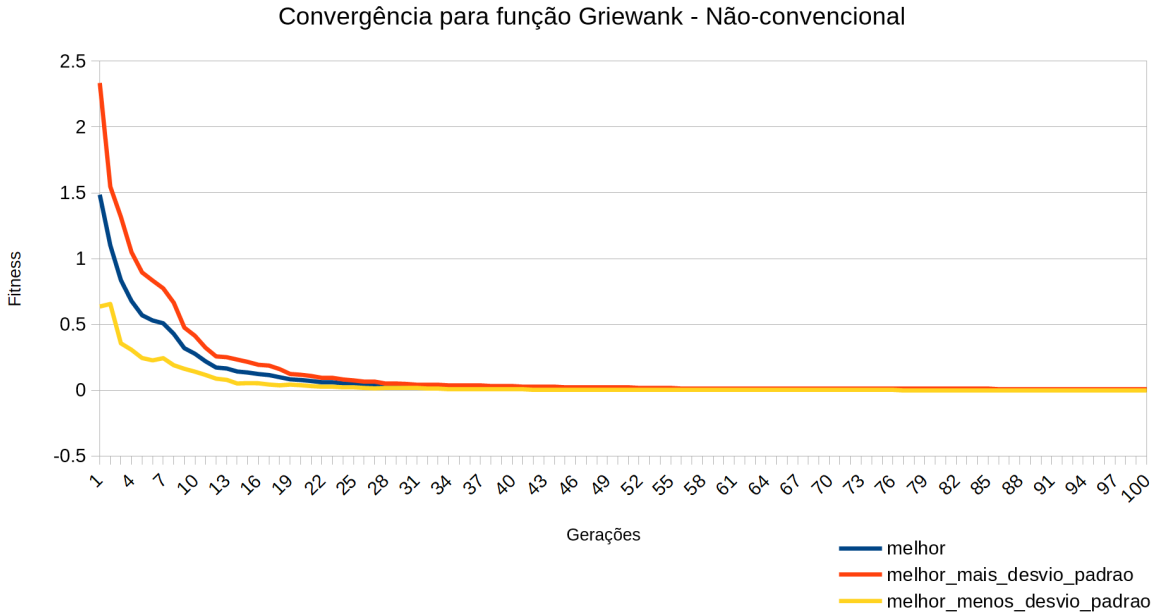

(b) Resultado da função *Rastrigin* para AG paralelo não convencional

Figura 24 – Gráficos de convergência da função de teste *Rastrigin* para AGs paralelos.

Fonte: elaborado pelo autor



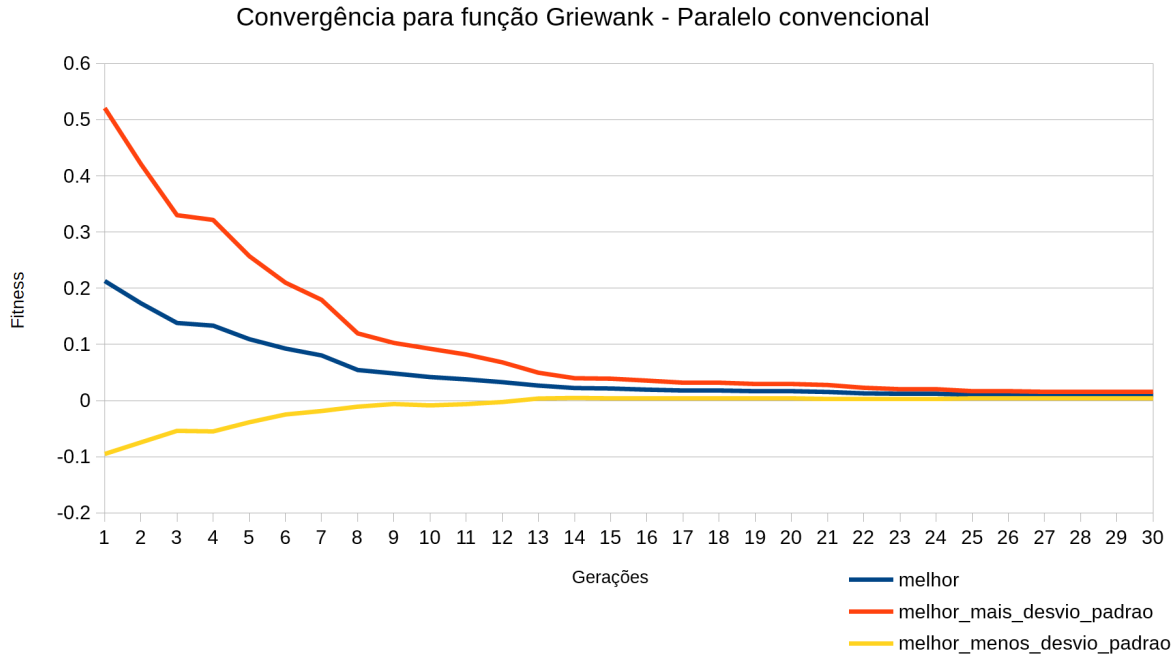

(a) Resultado da função *Griewank* para AG convencional sequencial



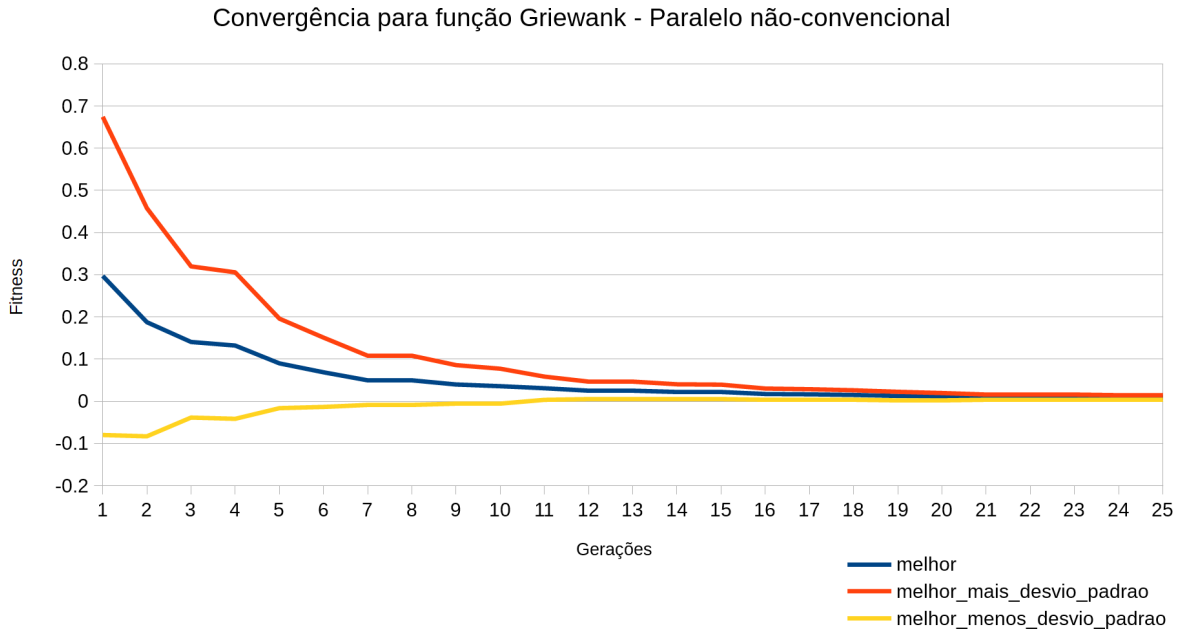

(b) Resultado da função *Griewank* para AG não convencional sequencial

Figura 25 – Gráficos de convergência da função de teste *Griewank* para AGs sequenciais.

Fonte: elaborado pelo autor




(a) Resultado da função *Griewank* para AG paralelo convencional




(b) Resultado da função *Griewank* para AG paralelo não convencional

Figura 26 – Gráficos de convergência da função de teste *Griewank* para AGs paralelos.

Fonte: elaborado pelo autor

Tabela 4 – Média das gerações em que os valores de mínimo global foram encontrados, para as 30 execuções

| Função | Convenc. | Não convenc. | Paralelo convenc. | Paralelo não convenc. |
|---|---|---|---|---|
| Schaffer02 | 79.1905 | 48.7743 | 271.223 | **42.2596** |
| Rosenbrock | 78.9696 | 53.8673 | 81.9382 | **36.2482** |
| Rastrigin | 53.8129 | 53.3201 | 52.2687 | **31.3299** |
| Griewank | 227.98 | 88.3056 | 178.4 | **74.4247** |

Fonte: elaborado pelo autor

mínimo global da função dentro da margem de erro, ou se não, o quão perto consegue chegar.

Tabela 5 – Melhores valores de *fitness* encontrados para todas as execuções e gerações dos AGs

| Função | Convenc. | Não convenc. | Paralelo convenc. | Paralelo não convenc. |
|---|---|---|---|---|
| Schaffer02 | **0** | **0** | **0** | **0** |
| Rosenbrock | 3.91756e-026 | 1.32595e-025 | 3.66821e-019 | **0** |
| Rastrigin | **0** | **0** | **0** | **0** |
| Griewank | **0** | **0** | **0** | **0** |

Fonte: elaborado pelo autor

Com exceção da função *Rosenbrock*, todos os AGs conseguiram encontrar o mínimo global (nesse caso, **0**). Para essa função, apenas o AGPN foi capaz de chegar nesse valor. Isso pode ter acontecido porque a superfície formada por essa função é consideravelmente íngreme, o que pode dificultar o encontro do mínimo global dependendo de que região do espaço de busca o algoritmo inicie a sua exploração e da sua velocidade de convergência. De qualquer modo, valores bem próximos foram encontrados nos outros algoritmos para essa função.

Na Tabela 6 são mostradas as médias dos melhores valores de *fitness* da última geração do AG, também para todas as 30 execuções, e para cada uma das funções de teste.

Tabela 6 – Média dos melhores valores de *fitness* da última geração encontrados para todas as execuções dos AGs

| Função | Convenc. | Não convenc. | Paralelo convenc. | Paralelo não convenc. |
|---|---|---|---|---|
| Schaffer02 | 0.000123891 | **0** | **0** | **0** |
| Rosenbrock | 0.00811501 | 0.000206899 | 1.63082e-006 | **4.87321e-011** |
| Rastrigin | 1.05217e-005 | **0** | **0** | **0** |
| Griewank | 0.00910585 | 0.00184238 | 2.89741e-006 | **0** |

Fonte: elaborado pelo autor

No critério utilizado para esta tabela não se chega tão facilmente ao mínimo global

quanto no da tabela anterior, por se tratar de uma média de apenas as últimas gerações de todas as execuções, ao invés de levar em conta as médias de todas as gerações do algoritmo. Também, o AGPN consegue ter desempenho igual ou superior às outras implementações para as funções apresentadas.

A próxima tabela (Tabela 7) mostra o número de execuções em que o AG, para cada função, encontrou o mínimo global dentro da margem de erro definida. Isso mostra o quão robusto cada algoritmo é.

Tabela 7 – Número de vezes que o mínimo global foi encontrado para as 30 execuções

| Função | Convenc. | Não convenc. | Paralelo convenc. | Paralelo não convenc. |
|---|---|---|---|---|
| Schaffer02 | 29 | **30** | **30** | **30** |
| Rosenbrock | 24 | 28 | **30** | **30** |
| Rastrigin | **30** | **30** | **30** | **30** |
| Griewank | 9 | 22 | **30** | **30** |

Fonte: elaborado pelo autor

Essa avaliação nos mostra que o AGPN é, sim, robusto, pois para as funções aqui expostas, ele consegue encontrar o valor ótimo em todas as 30 execuções. Além disso, isso mostra que o mesmo desempenho é obtido para o AGPC. Isso mostra que, como visto por exemplo em (COHOON et al., 1987), AGs com múltiplas populações realizam uma maior exploração no espaço de busca e conseguem trazer melhores resultados, como a melhora da robustez do algoritmo. Ademais, destacam-se os resultados para a função de teste *Griewank*, que para os resultados do AG convencional sequencial entre as suas 30 execuções só encontrou o mínimo global em 9.

Pelos gráficos de convergência apresentados e avaliados na seção anterior, observa-se que o AGPN é em geral mais preciso que as outras variantes, já que os valores de desvio padrão reunidos pelas suas gerações são na maioria menor que esses mesmos valores apresentados na saída dos outros algoritmos.

A partir dos resultados de velocidade de convergência apresentados na Tabela 4 e dos demais resultados listados nas tabelas 5, 6 e 7, apesar de ser robusto, o AGPC geralmente demora mais que os sequenciais para encontrar o valor ótimo global do *fitness*, dados os parâmetros especificados nas tabelas 1, 2 e 3. Isso ocorre porque no total as alternativas paralelas possuem mais indivíduos, já que possuem múltiplas populações, o que resulta em um maior processamento, no total. No entanto, o AGPN é capaz de inverter esses resultados e o faz o melhor nesse critério de avaliação para todas essas funções.

# 5 Conclusão

Algoritmos genéticos paralelos e operadores genéticos não convencionais são possíveis variações dentro da arquitetura de um AG. O uso dessas alternativas é feito para aumentar a robustez e/ou precisão do algoritmo, dentre outros fatores. A partir do estudo dessas duas subáreas foi decidido para o presente trabalho realizar uma implementação conjunta dessas duas variações. Mais especificamente, um AG paralelo assíncrono com múltiplas ilhas, que também utiliza o operador genético de recombinação por transformação bacteriana, e que utiliza a política de migração dupla de substituição aleatória dos melhores.

As dificuldades encontradas no trabalho em grande parte estavam relacionadas à paralelização das populações do AG. Mesmo com o uso das cláusulas do `OpenMP`, que minimizam as mudanças que devem ser feitas no código, foi necessário um cuidado com alguns trechos específicos. Na sincronia do operador de migração, para controlar casos de *deadlocks* no momento de escolha das populações que vão participar no processo de migração, é necessário o uso de variáveis booleanas de controle. Além disso, a própria execução mais controlada desse operador para que seja iniciado por uma única população precisou ser feita, de maneira que não seja executado de maneira ininterrupta em uma *thread* separada.

Pela observação dos resultados, percebeu-se um bom desempenho dessa implementação. A maior variabilidade genética provinda tanto do operador não convencional utilizado quanto do uso de múltiplas ilhas no AG ajudaram a fugir dos mínimos locais nas funções e a encontrarem ótimos globais com uma alta frequência. Juntamente com isso, a convergência mais rápida dessas alternativas apresentadas por essas duas variações ajudaram o algoritmo a encontrar o mínimo global mais rápido, além de uma maior robustez e precisão. Essas características foram comparadas com outras 3 implementações (AG sequencial sem operador não convencional, AG sequencial com operador não convencional, e AG paralelo sem operador não convencional) e, comparado com elas, o algoritmo apresentado mostrou desempenho tão bom quanto, ou melhor.

## 5.1 Trabalhos futuros

Trabalhos futuros poderiam focar em outras características que não foram o foco ou não foram tratadas com muita atenção no presente trabalho. Novos trabalhos podem ter como objeto de estudo detalhes como alterações na arquitetura principal do algoritmo, e tentar como alternativa um maior grau de paralelização ao seu funcionamento. Um exemplo disso seria utilizar uma topologia híbrida, onde cada ilha seria um AG com

paralelização global. Outra alternativa seria introduzir paralelização dentro do operador não convencional utilizado, ao invés de apenas replicá-lo em um AG paralelo. Para o caso de aplicação em um problema real, uma avaliação mais cautelosa de como os operadores genéticos e parâmetros do algoritmo pudessem ser otimizados para atingir resultados ainda melhores, além da aplicação de funções mais difíceis de se otimizar, também seria de grande utilidade.

Por último, aplicar e, se necessário, adaptar a implementação do presente trabalho para outros ambientes/linguagens poderia também ser útil. Existem linguagens funcionais que são conhecidas por serem muito adequadas para ambientes altamente concorrentes. O uso delas juntamente com uma arquitetura massivamente paralela poderia ser proveitoso. Outro trabalho interessante seria adaptar a implementação feita para o ambiente CUDA. Isso não exigiria grandes recursos, pois a linguagem principal desse ambiente é C, e o presente trabalho foi desenvolvido com C++. Além disso, utilizar essa tecnologia traria um imenso ganho de desempenho para o algoritmo, o que facilitaria o seu uso para problemas que demandam um maior poder computacional.

# Referências

ALBA, E.; TROYA, J. M. A survey of parallel distributed genetic algorithms. *Complex.*, John Wiley & Sons, Inc., New York, NY, USA, v. 4, n. 4, p. 31–52, mar. 1999. ISSN 1076-2787. Disponível em: <http://dx.doi.org/10.1002/(SICI)1099-0526(199903/04)4:4<31::AID-CPLX5>3.3.CO;2-W>. Citado 6 vezes nas páginas 11, 16, 18, 19, 20 e 22.

CANTÚ-PAZ, E. A survey of parallel genetic algorithms. 06 1999. Citado 8 vezes nas páginas 11, 16, 17, 18, 19, 20, 21 e 34.

CHEN, B. et al. A fast parallel genetic algorithm for graph coloring problem based on cuda. In: *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*. [S.l.: s.n.], 2015. p. 145–148. Citado 3 vezes nas páginas 11, 12 e 13.

COHOON, J. P. et al. Punctuated equilibria: a parallel genetic algorithm. In: HILLSDALE, NJ: L. ERLHAUM ASSOCIATES, 1987. *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*. [S.l.], 1987. Citado 8 vezes nas páginas 11, 12, 13, 16, 17, 18, 21 e 51.

EIBEN, A.; SMITH, J. E. *Introduction to Evolutionary Computing*. [S.l.]: Springer-Verlag Berlin Heidelberg, 2015. Citado 3 vezes nas páginas 11, 18 e 20.

FERREIRA, L. P. da S. *Políticas de Migração Assíncronas em Algoritmos Genéticos Paralelos Aplicado a Otimização Multimodal*. Monografia (Trabalho de Conclusão de Curso) — Faculdade de Computação, Universidade Federal do Pará, Belém, 2017. Citado 12 vezes nas páginas 12, 13, 14, 17, 20, 22, 31, 33, 34, 36, 38 e 39.

FUQIANG, D.; MINQING, Z.; JIA, L. Virus-evolutionary genetic algorithm based selective ensemble for steganalysis. In: *2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. [S.l.: s.n.], 2014. p. 553–558. Citado 2 vezes nas páginas 11 e 13.

GAVANA, A. *Global Optimization Benchmarks and AMPGO*. 2013. Disponível em: <http://infinity77.net/global_optimization/>. Acesso em: 28.6.2018. Citado 4 vezes nas páginas 27, 28, 29 e 30.

GRAFOS: conceitos, algoritmos e aplicações. Elsevier, 2012. ISBN 9788535257168. Disponível em: <https://books.google.com.br/books?id=9Fq7NAEACAAJ>. Citado na página 21.

HERDA, M. Parallel genetic algorithm for capacitated p-median problem using openmp protocol. In: *2016 IEEE 17th International Symposium on Computational Intelligence and Informatics (CINTI)*. [S.l.: s.n.], 2016. p. 000347–000352. Citado 3 vezes nas páginas 11, 12 e 13.

KUBOTA, N.; FUKUDA, T.; SHIMOJIMA, K. Virus-evolutionary genetic algorithm for a self-organizing manufacturing system. *Computers & Industrial Engineering*, v. 30, n. 4, p. 1015 – 1026, 1996. ISSN 0360-8352. Disponível em:

<http://www.sciencedirect.com/science/article/pii/0360835296000496>. Citado na página 11.

KUMAR, J.; KUMAR, R.; DIXIT, M. Result merging in meta-search engine using genetic algorithm. In: *International Conference on Computing, Communication Automation*. [S.l.: s.n.], 2015. p. 299–303. Citado na página 11.

LI, N. n.; GU, J. h.; LIU, B. y. A new genetic algorithm based on negative selection. In: *2006 International Conference on Machine Learning and Cybernetics*. [S.l.: s.n.], 2006. p. 4297–4299. ISSN 2160-133X. Citado 2 vezes nas páginas 12 e 13.

MORADY, R.; DAL, D. A multi-population based parallel genetic algorithm for multiprocessor task scheduling with communication costs. In: *2016 IEEE Symposium on Computers and Communication (ISCC)*. [S.l.: s.n.], 2016. p. 766–772. Citado 2 vezes nas páginas 11 e 12.

MOREIRA, R. S.; TEIXEIRA, O. N.; OLIVEIRA, R. C. L. de. Mixing theory of retroviruses and genetic algorithm to build a new nature-inspired meta-heuristic for real-parameter function optimization problems. In: *2010 10th International Conference on Hybrid Intelligent Systems*. [S.l.: s.n.], 2010. p. 181–184. Citado 3 vezes nas páginas 13, 23 e 24.

NVIDIA. *CUDA Zone*. 2018. Disponível em: <https://developer.nvidia.com/cuda-zone>. Acesso em: 13.06.2018. Citado na página 12.

OpenMP Architecture Review Board. *OpenMP Application Program Interface Version 4.0*. 2013. Disponível em: <https://www.openmp.org/wp-content/uploads/OpenMP4.0.0.pdf>. Acesso em: 11.06.2018. Citado na página 15.

PEREIRA, G. T. M. *NOVO OPERADOR PARA ALGORITMOS GENÉTICOS BASEADO NA RECOMBINAÇÃO POR TRANSFORMAÇÃO DE BACTÉRIAS: Um estudo comparativo em otimização multimodal*. Monografia (Trabalho de Conclusão de Curso) — Faculdade de Computação, Universidade Federal do Pará, Belém, 2017. Citado 11 vezes nas páginas 11, 12, 13, 14, 15, 25, 26, 31, 37, 38 e 40.

SURJANOVIC, S.; BINGHAM, D. *Virtual Library of Simulation Experiments: Test Functions and Datasets*. 2013. Disponível em: <http://www.sfu.ca/~ssurjano>. Acesso em: 28.6.2018. Citado 2 vezes nas páginas 26 e 27.

TARIQUE, T. A.; ZAMEE, M. A.; KHAN, M. I. A new approach for pattern recognition with neuro-genetic system using microbial genetic algorithm. In: *2014 International Conference on Electrical Engineering and Information Communication Technology*. [S.l.: s.n.], 2014. p. 1–4. Citado 2 vezes nas páginas 11 e 12.

UFRJ. *Esteganálise*. 2008. Disponível em: <https://www.gta.ufrj.br/grad/08_1/estegano/Esteganlise.html>. Acesso em: 11.06.2018. Citado na página 11.

UPADHYAYULA, S.; KOBTI, Z. Population migration using dominance in multi-population cultural algorithms. In: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. [S.l.: s.n.], 2015. p. 614–617. Citado 3 vezes nas páginas 11, 12 e 21.

WIRIYASERMKUL, N.; BOOBJING, V.; CHANVARASUTH, P. A meiosis genetic algorithm. In: *2010 Seventh International Conference on Information Technology: New Generations*. [S.l.: s.n.], 2010. p. 285–289. Citado na página 12.