Universidade Federal de Juiz de Fora

Departamento de Ciência da Computação

**Rodrigo da Cruz Alvarenga Fajardo Pontes**

# Um Modelo de Algoritmo Evolucionista Paralelo Global Distribuído

Juiz de Fora

2018

**Rodrigo da Cruz Alvarenga Fajardo Pontes**

# Um Modelo de Algoritmo Evolucionista Paralelo Global Distribuído

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Juiz de Fora, na área de computação evolucionista, como requisito parcial para obtenção do título de Cientista da Computação

Orientador: Carlos Cristiano Hasenclever Borges

Juiz de Fora

2018

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

da Cruz Alvarenga Fajardo Pontes, Rodrigo.
Um Modelo de Algoritmo Evolucionista Paralelo Global Distribuído / Rodrigo da Cruz Alvarenga Fajardo Pontes. – 2018.
32 f. : il.

Orientador: Carlos Cristiano Hasenclever Borges
Monografia – Universidade Federal de Juiz de Fora, Departamento de Ciência da Computação. , 2018.

1. Computação evolucionista. 2. distribuída. 3. Seleção. I. Hasenclever Borges, Carlos Cristiano

**Rodrigo da Cruz Alvarenga Fajardo Pontes**

**Um Modelo de Algoritmo Evolucionista Paralelo Global Distribuído**

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Juiz de Fora, na área de computação evolucionista, como requisito parcial para obtenção do título de Cientista da Computação

Aprovada em: 10/05/2018

BANCA EXAMINADORA

---

Prof. Dr. Carlos Cristiano Hasenclever Borges - Orientador
Universidade Federal de Juiz de Fora

---

Professor Dr. Leonardo Goliatt
Universidade Federal de Juiz de Fora

---

Professor Dr. Luciana Conceição Dias Campos
Universidade Federal de Juiz de Fora

## AGRADECIMENTOS

Agradeço aos professores do Departamento de Ciência da Computação e à minha família pelo apoio nesses anos de estudo.

Na longa história da humanidade (e dos animais também), aqueles que aprenderam a colaborar e improvisar mais eficientemente prevaleceram. - Charles Darwin

## RESUMO

Na computação evolucionista existem diversas técnicas de se trabalhar com sistemas distribuídos para resolver determinados problemas. Nesse trabalho em questão há uma população que será evoluída para resolver um certo problema em um sistema distribuído, onde cada processador irá ser responsável por evoluir parte dessa população, porém ele irá enxergar todos os indivíduos na hora da seleção, não apenas os indivíduos pelos quais é responsável. Sendo assim, é necessário que esses processadores comuniquem entre si para que cada um possa ser atualizado sobre os indivíduos pelos quais ele não é responsável. Serão feitas simulações com determinados atrasos para que ocorra essa atualização, ou seja, com um tempo fixo de espera para que cada processador seja atualizado sobre os outros indivíduos, para ver como a solução final é afetada. Além disso serão rodadas simulações variando o número de ilhas para que possamos ver como a solução final se comporta diante dessas diferenças.



Palavras-chave: Computação Evolucionista, Processamento Paralelo, Algoritmos Evolutivos Paralelos.

**ABSTRACT**

In the evolutionary computing field, there are several techniques of working with distributed systems in order to solve certain problems. In this work there is a population that will be evolved in order to solve a problem in a distributed system, where each processor will be responsible for evolving a part of this population, but it will be able to see all the individuals in the selection phase, not just the ones it's responsible for. So it's necessary for the processors to communicate among themselves so that each one of them can be updated on the individuals they're not responsible for. There will be simulations with certain delays for this update to occur, or in other words, with a fixed time of waiting for the processors to be updated on the other individuals, in order to see how the final solution is affected. We will also run several simulations with a different number of islands so we can see how the final solution is affected by these changes.

Key-words: Evolutionary Computing, Parallel Processing, Parallel Evolutionary Algorithms.

## SUMÁRIO


**LISTA DE ILUSTRAÇÕES** . . . . . **8**

**1 INTRODUÇÃO** . . . . . **10**

1.1 Objetivos . . . . . 12

**2 REVISÃO BIBLIOGRÁFICA** . . . . . **13**

2.1 Algoritmo Evolucionista Paralelo Global (AEP Global) . . . . . 13

2.2 AEP Distribuído . . . . . 15

**3 METODOLOGIA** . . . . . **17**

3.1 Alocação da população entre os processadores . . . . . 19

**4 IMPLEMENTAÇÃO DO CÓDIGO E FUNÇÕES TESTE** . . **20**

4.1 Problemas a serem resolvidos . . . . . 20

**5 RESULTADOS** . . . . . **21**

**6 ANÁLISE ESTATÍSTICA** . . . . . **29**

**7 CONCLUSÃO E TRABALHOS FUTUROS** . . . . . **31**

**REFERÊNCIAS** . . . . . **32**

# LISTA DE ILUSTRAÇÕES

Figura 1 – Pseudocódigo para algoritmos evolucionistas . . . . . 10
Figura 2 – Pseudocódigo para algoritmo genético geracional . . . . . 12
Figura 3 – Pseudocódigo para AEPG - MESTRE . . . . . 14
Figura 4 – Pseudocódigo para AEPG - ESCRAVOS . . . . . 15
Figura 5 – Pseudocódigo para AEPD . . . . . 16
Figura 6 – Pseudocódigo para AEPDG . . . . . 18
Figura 7 – Pseudocódigo da função SELDG . . . . . 19
Figura 8 – Divisão dos indivíduos . . . . . 19
Figura 9 – Problema Ackley: média (esquerda) e melhor solução (direita) com 2 processadores . . . . . 21
Figura 10 – Problema Ackley: média (esquerda) e melhor solução (direita) com 4 processadores . . . . . 21
Figura 11 – Problema Ackley: média (esquerda) e melhor solução (direita) com 6 processadores . . . . . 21
Figura 12 – Problema Ackley: média (esquerda) e melhor solução (direita) com 8 processadores . . . . . 22
Figura 13 – Problema Ackley: média (esquerda) e melhor solução (direita) com 10 processadores . . . . . 22
Figura 14 – Problema Ackley: média (esquerda) e melhor solução (direita) com 12 processadores . . . . . 22
Figura 15 – Problema DeJong: média (esquerda) e melhor solução (direita) com 2 processadores . . . . . 22
Figura 16 – Problema DeJong: média (esquerda) e melhor solução (direita) com 4 processadores . . . . . 23
Figura 17 – Problema DeJong: média (esquerda) e melhor solução (direita) com 6 processadores . . . . . 23
Figura 18 – Problema DeJong: média (esquerda) e melhor solução (direita) com 8 processadores . . . . . 23
Figura 19 – Problema DeJong: média (esquerda) e melhor solução (direita) com 10 processadores . . . . . 23
Figura 20 – Problema DeJong: média (esquerda) e melhor solução (direita) com 12 processadores . . . . . 24
Figura 21 – Problema Rastrigin: média (esquerda) e melhor solução (direita) com 2 processadores . . . . . 24
Figura 22 – Problema Rastrigin: média (esquerda) e melhor solução (direita) com 4 processadores . . . . . 24
Figura 23 – Problema Rastrigin: média (esquerda) e melhor solução (direita) com 6 processadores . . . . . 24

Figura 24 – Problema Rastrigin: média (esquerda) e melhor solução (direita) com 8 processadores . . . . . 25
Figura 25 – Problema Rastrigin: média (esquerda) e melhor solução (direita) com 10 processadores . . . . . 25
Figura 26 – Problema Rastrigin: média (esquerda) e melhor solução (direita) com 12 processadores . . . . . 25
Figura 27 – Problema Rosenbrock: média (esquerda) e melhor solução (direita) com 2 processadores . . . . . 25
Figura 28 – Problema Rosenbrock: média (esquerda) e melhor solução (direita) com 4 processadores . . . . . 26
Figura 29 – Problema Rosenbrock: média (esquerda) e melhor solução (direita) com 6 processadores . . . . . 26
Figura 30 – Problema Rosenbrock: média (esquerda) e melhor solução (direita) com 8 processadores . . . . . 26
Figura 31 – Problema Rosenbrock: média (esquerda) e melhor solução (direita) com 10 processadores . . . . . 26
Figura 32 – Problema Rosenbrock: média (esquerda) e melhor solução (direita) com 12 processadores . . . . . 27
Figura 33 – Problema Schwefel: média (esquerda) e melhor solução (direita) com 2 processadores . . . . . 27
Figura 34 – Problema Schwefel: média (esquerda) e melhor solução (direita) com 4 processadores . . . . . 27
Figura 35 – Problema Schwefel: média (esquerda) e melhor solução (direita) com 6 processadores . . . . . 27
Figura 36 – Problema Schwefel: média (esquerda) e melhor solução (direita) com 8 processadores . . . . . 28
Figura 37 – Problema Schwefel: média (esquerda) e melhor solução (direita) com 10 processadores . . . . . 28
Figura 38 – Problema Schwefel: média (esquerda) e melhor solução (direita) com 12 processadores . . . . . 28
Figura 39 – Variação da qualidade das soluções nos diferentes intervalos de migração 29
Figura 40 – Variação da qualidade das soluções para diferentes números de processadores . . . . . 30

# 1 INTRODUÇÃO

A computação evolucionista é uma área da Inteligência Computacional que, baseando-se em conceitos da teoria da evolução de Darwin, e em conceitos da genética natural, busca desenvolver algoritmos que sejam efetivos em problemas complexos de busca, por exemplo, otimização multimodal.

Tais algoritmos, denominados algoritmos evolucionistas (AEs), tem como principal característica serem baseados em população composta por possíveis soluções, que devem evoluir seguindo procedimentos de controle de pressão de seleção, que visam favorecer as soluções de melhor desempenho, e aplicação dos chamados operadores evolucionistas, associados ao processo de reprodução. Muitos desses algoritmos foram introduzidos nas últimas décadas [3], vários usando sistemas distribuídos para diminuir o tempo de simulação [1]. O funcionamento dos AEs se dá de acordo com algumas etapas. Inicialmente, gera-se uma população inicial, geralmente, de forma randômica. Segue-se, então, a aplicação de um ciclo evolutivo, até que que alcance um critério de parada pré-determinado, que envolve: (i) determinação da aptidão dos indivíduos (candidatos a solução) da população; (ii) aplicação de um processo de seleção via um operador de seleção; (iii) aplicação de operadores evolutivos que repliquem a reprodução, visando gerar uma população com novos indivíduos associados à próxima geração. O pseudocódigo de um algoritmo evolucionista é apresentado em Fig. 1.

**Algoritmo evolucionista**
**Início**
t=0
Inicialize a população P(t)
Avalie a população P(t)
**Enquanto** critério de parada não é alcançado **faça**
Selecione a população P(t+1) a partir da população P(t)
Aplique operadores evolutivos à população P(t+1)
$t = t + 1$
**Fim enquanto**
**Fim**

Figura 1 – Pseudocódigo para algoritmos evolucionistas

Entre os principais AEs, pode-se destacar os algoritmos genéticos (AGs), que simulam de maneira bem efetiva os conceitos de Darwin e da genética natural. Atualmente, diversos modelos baseados em população também foram enquadrados no contexto da computação evolucionista como, por exemplo, estratégia de evolução, enxame de partículas, evolução diferencial, entre diversos outros. Esses algoritmos são também muito utilizados no chamado modelo de ilhas [4], onde a população é dividida em subpopulações, cada uma em uma ilha, para evoluírem separadamente. Isso é vantajoso pois a existência de

várias ilhas ajuda a preservar a diversidade genética, pois cada ilha segue um caminho diferente no espaço de soluções [8] [5]. O modelo de ilhas possui vários parâmetros que podem ser ajustados com base no tipo de problema a ser resolvido. Esses ajustes pode influenciar muito na qualidade das soluções [7]. Por fim, o modelo de ilhas oferece boas soluções mesmo ao rodar em uma infraestrutura paralela ou distribuída não-confiável, o que significa que é um modelo tolerante a falhas [2].

Descreve-se, como referência, o algoritmo genético (AG), um dos principais representantes da CE. Baseado na teoria da evolução natural, considerado um processo robusto, simples e poderoso, tem fácil adaptação para solução computacional, principalmente, para problemas que envolvem espaços de busca complexos. Os AGs como métodos de otimização, inspiram-se em mecanismos da evolução com objetivo de varrer eficientemente a região de busca, encontrando a solução ótima ou soluções sub-ótimas para o problema em questão, onde as soluções são obtidas através da replicação do processo evolutivo simulado pelo algoritmo. A construção de um AG é feita utilizando as mesmas etapas que compõem os AEs já descritos. Sua principal característica é a utilização de dois espaços de representação das variáveis do problema: o espaço dos genótipos, geralmente utilizando o alfabeto binário e, o espaço dos fenótipos, que utiliza representação específica das variáveis de projeto. Logicamente, estas representações são inspiradas nos conceitos da genética natural.

Porém, de uma forma geral, apesar do AG apresentar os mesmos passos de construção dos AEs, utiliza uma nomenclatura específica para designar seus componentes. Assim, gerada uma população inicial de **cromossomos**, que representam possíveis soluções de um problema no espaço dos genótipos, esta população é **decodificada** no espaço dos fenótipos, obtendo **indivíduos** que são avaliados, determinando a **aptidão** para cada um deles. Baseando-se na teoria de sobrevivência dos mais aptos, um processo de **seleção** probabilística é realizado, onde indivíduos com melhores aptidão tem maior chance de serem selecionados. Os selecionados transferem suas características para seus **descendentes** através de um procedimento de **reprodução**, simulada por meio dos operadores **genéticos** de **recombinação** e **mutação**. Os **filhos** gerados irão compor a próxima população, ou seja, a próxima **geração**. Todo esse processo é repetido até que seja encontrada uma solução satisfatória. Apresenta-se, em Fig. 2, o pseudocódigo deste AG, conhecido como algoritmo genético geracional.

**Algoritmo genético geracional**
**Início**
Inicialize a população aleatória P
Avalie os indivíduos da população P
**Repita**
Selecione indivíduos em P
Aplique operadores de recombinação com probabilidade $p_r$
Aplique operadores de mutação com probabilidade $p_m$
Insira novos indivíduos em P'
**Até** população P' completa
Avalie os indivíduos na população P'
P $\leftarrow$ P'
**Até** o critério de parada ser satisfeito
**Fim**

Figura 2 – Pseudocódigo para algoritmo genético geracional

## 1.1 Objetivos

No próximo capítulo serão introduzidos os algoritmos paralelos usados como base para criar o algoritmo proposto nesse trabalho.

O objetivo principal deste trabalho é avaliar a viabilidade do algoritmo proposto em relação a sua expectativa de funcionamento. Será realizado por meio da análise dos resultados obtidos, através de simulações. Caso a qualidade dos resultados se aproxime do modelo serial, saberemos então que o modelo proposto é viável.

Na fase de testes, é rodado um número alto de simulações para que a média entre elas possa ser usada para exibir os resultados. O objetivo dessa etapa é gerar os resultados das simulações para em seguida analisá-los e verificar se o desempenho do novo modelo é aceitável em comparação com o modelo serial.

# 2 REVISÃO BIBLIOGRÁFICA

Como base para este trabalho, podemos citar os algoritmos evolutivos paralelos (AEP) que possuem vantagens em relação aos sequenciais no que diz respeito ao custo computacional. Ressalta-se que modelos paralelos também pode estar associados a outros paradigmas evolutivos, envolvendo conceitos de subpopulações e suas interações no decorrer do processo. O modelo mais próximo do modelo de AE serial é o modelo paralelo global, descrito a seguir.

## 2.1 Algoritmo Evolucionista Paralelo Global (AEP Global)

Os AEP Globais ou centralizados são baseados no envolvimento de toda a população nos procedimentos genéticos, tendo ocomo consequência a obtenção de resultado idêntico à sua versão sequencial. Geralmente, paralelizam a etapa de avaliação dos indivíduos, que usualmente é a que demanda maior esforço computacional. Os operadores de seleção, recombinação e mutação, com baixo custo computacional, são aplicados em um processador que centraliza estas tarefas, o processador mestre, também responsável por distribuir partes da população para os chamados processadores escravos, que farão a avaliação destas parcelas, devolvendo os resultados para o mestre, findas suas tarefas. Esta estratégia é conhecida como modelo mestre-escravo.

No AEP Global, a avaliação dos indivíduos, usualmente com maior custo computacional, é feita em paralelo. Como o modelo deve funcionar como uma população única, é necessário que a cada geração se tenha um procedimento de sincronia onde o processador mestre deve esperar que todos os escravos enviem os indivíduos avaliados para que possa aplicar os operadores genéticos visando gerar a próxima população e sua posterior redistribuição aos processadores escravos.

A arquitetura do ambiente paralelo tem grande influência no modelo mestre-escravo, pois se a memória for distribuída, é necessário que haja uma comunicação eficiente para manter a efetividade do método, por isso os computadores com memória compartilhada geralmente são mais adequados para esse tipo de abordagem.

Uma questão relevante do funcionamento do modelo mestre-escravo é a estrategia de distribuição da população para os escravos. A divisão uniforme da população em relação ao número de processadores escravos (modelo balanceado) tem funcionamento bastante comprometido no caso onde o ambiente paralelo é não-dedicado ou heterogêneo. Neste caso, implementação de balanceamento estático ou dinâmico é crucial para que se tenha desempenho computacional adequado. Uma outra solução são os modelos assíncronos, que flexibilizam a sincronia necessária no processador mestre no recebimento das avaliações dos processadores escravos. Porém este modelo tende a se afastar do resultado obtido pelo modelo serial associado.

Um outro problema dos AEP Global é a tendência de se ter um gargalo na comunicação com o processador mestre, quando o número de processadores escravos aumentam, o que indica uma limitação efetiva na escalabilidade do modelo. A seguir, apresenta-se o pseudocódigo do AEP Global em sua versão balanceada, para o processador mestre na Fig. 3 e para os processadores escravos na Fig. 4.

**Algoritmo evolucionista paralelo global**

**MESTRE**

**Início**
Defina tamanho da população ($npop$), número de processadores escravos ($n_{pe}$)
Inicialize aleatoriamente a população com $npop$ indivíduos
Calcule o número de índivíduos por processador: $ip = npop/n_{pe}$
**Enquanto** critério de parada não é alcançado **faça**
**Para** $i = 1, n_{pe}$ **faça**
Envie $ip$ indivíduos para $i$-ésimo processador escravo
**Fim para**
**Para** $i = 1, n_{pe}$ **faça**
Receba $ip$ indivíduos para $i$-ésimo processador escravo
**Fim para**
Organize o vetor de aptidão
Aplique operadores evolutivos gerando nova população
**Fim enquanto**
**Fim**

Figura 3 – Pseudocódigo para AEPG - MESTRE

**Algoritmo evolucionista paralelo global**

**ESCRAVOS**

**Início**
Receba $ip$ indivíduos do processador MESTRE
**Para** $i = 1, ip$ **faça**
avalie $i$-ésimo indivíduo e aloque no vetor de aptidão
**Fim para**
Envie o vetor de aptidão com $ip$ avaliações para o MESTRE
**Fim**

Figura 4 – Pseudocódigo para AEPG - ESCRAVOS

## 2.2 AEP Distribuído

Uma outra estratégia para construção de um AEP é baseada no conceito de subpopulações memores que devem evoluir separadamente utilzando o menor nível de material genético presente e, em uam segunda etapa viabiliza a comunicação entre as subpopulações visando a inserção de novo material genético nas mesmas que irão proporcionar um no processo evolutivo. Este modelo é conhecido como AEP Distribuído ou em ilhas.

Sua implementação em uma máquina com ambiente paralelo não necessita de um processador centralizador, sendo que as subpopulações podem ser geradas diretamente nos processadores ilhas para gerar o ciclo evolutivo que se segue. Desta forma, permite uma maior escalabilidade em relação ao número dfe processadores utilizados devido a descentralização.

Por representar outra abordagem evolutiva, o resultado do AEP Distribuído pode ser bastante distinto dos modelos seriais ou do AEP Global. Pode-se dizer que não se tem, na literatura, questão fechada de qual modelo é mais eficiente em termos da qualidade da solução obtida.

Um inconveniente dos AEP Distribuídos é a necessidade de novos operadores e o grande número de parâmetros adicionais a serem definidos para viabilizar sua execução, com alguns sendo dependeres diretamente do ambiente de alto desempenho utilizado [**?**]. Um dos principais operadores necessários é o operador de migração, que indica a estratégia a ser utilizada para a comunicação entre as subpopulações. Estudos mostram que a escolha desse operador pode ser mais importante do que a escolha de como será a substituição dos indivíduos da população [6]. É aplicado por meio de dois parâmetros principais, a saber:

- taxa de migração ($TM$): indica a quantidade de indivíduos a serem enviados para outros processadores;
- intervalo de migração ($I$): define o padrão temporal para a comunicação.

Além deste operador com seus parâmetros principais, diversas outras variáveis precisam serem definidas para aplicação do AEP Distribuído, que, dependendo dos valores e padrões adotados, tornam o seu uso um tanto mais complexo e voláteis em relação aos modelos de AEP Globais. A seguir na Fig. 6, apresenta-se o pseudocódigo do modelo distribuído.

**Algoritmo evolucionista distribuído**
**Início**
Defina número de processadores ($np$), população por processador ($npop_p$)
Defina intervalo de migração ($I$) e taxa de migração ($TM$)
Defina demais parâmetros do modelo distribuído
Inicialize o ambiente paralelo com $np$ processadores
Gere a população com $npop_p$ indivíduos
**Enquanto** critério de parada não é alcançado **faça**
**Para** $i = 1, I$ **faça**
Avalie os $npop_p$ indivíduos da população do processador
Aplique os operadores evolutivos e gere nova população do processador
**Fim para**
Envie$TM$ indivíduos selecionados para migração de acordo com topologia
Receba e aloque $TM$ indivíduos migrados de acordo com topologia
**Fim enquanto**
Concatene o resultado dos $npop_p.np$ indivíduos dos processadores
Finalize o ambiente paralelo
**Fim**

Figura 5 – Pseudocódigo para AEPD

Finalizando, deve-se ressaltar que a descrição dos principais modelos de AEP neste processo introdutório, com destaque para suas principais qualidades e dificuldades tanto em termos de funcionamento quanto em relação ao procedimentos de implementação, servirá de base para o desenvolvimento do modelo de AEP a ser proposto.

Na continuidade do trabalho apresenta-se, a abordagem a ser utilizada na construção de um modelo específico de AEP. Segue-se a definição do objetivo principal a ser atingido, como será feita a divisão dos processadores, as ferramentas utilizadas. Então, os resultados obtidos são apresentados e nalisados. Finalmente, algumas conclusóes e trabalhos futuros são delineados.

# 3 METODOLOGIA

Nesse trabalho, primeiramente é elaborado um novo modelo, utilizando o chamado algoritmo evolucionista paralelo distribuído global, utilizando características de algoritmos já existentes na literatura, com o intuito de manter as vantagens de cada um deles. Será desenvolvido um algoritmo evolucionista distribuído visando simular, dentro do possível, um modelo serial, onde será utilizada apenas uma população, e não subpopulações divididas em ilhas, onde cada processador será responsável por evoluir somente uma parte dessa população (uma parcela específica que será atribuída ao mesmo). Apesar do processador evoluir apenas certos indivíduos, ele conseguirá ter acesso a todos os indivíduos da população no momento da seleção, ou seja, o processador irá gerar sua parcela de novos indivíduos com acesso ao material genético de toda a população, não apenas dos indivíduos pelos quais o processador é responsável. Este modelo pode ser enquadrado em AEP Distribuído com comportamento Global (AEPD Global), ou seja, apresenta características de implementação de um modelo distribuído mas busca um processo evolutivo similar a um modelo global [1].

Para realizar essa implementação, precisa-se fazer com que cada processador possua uma cópia idêntica da população global no ínico do processo evolutivo. Porém, cada processador será responável pela evolução de somente uma parcela da população de indivíduos mas, com todos os indivíduos da população global sendo considerados no processo seletivo. Desta forma, a parte da população de cada processador não sujeita a evolução, vai ficando desatualizada no decorrer das gerações. A população de todos os processadores envolvidos só é completamente atualizada durante a aplicação do operador de migração, onde cada processador envia sua parcela da população atualizada para todos os outros processadores envolvidos. Da mesma maneira, cada processador recebe os indivíduos atualizados dos outros processadores. Assim, na aplicação do operador de migração, tem-se a re-unificação atualizadada população global em todos os processadores envolvidos. Ressalta-se que, neste caso, o parâmentro taxa de mutação ($TM$) não se aplica.

A questão central deste modelo distribuído está na influência que a defasagem da população em cada processador, reflete na qualidade do processo evolutivo em relação ao modelo de um AE serial. Assim, a variável intervalo de migração ($I$), do operador de migração, assume papel fundamental no comportamento deste algoritmo evolutivo paralelo. O pseudocódigo da abordagem proposta é mostrado abaixo, na Fig. 6.

**Algoritmo evolucionista distribuído global**
**Início**
Defina população ($npop$), número de processadores ($np$), intervalo de migração ($I$)
Gere a população com $npop$ indivíduos
Avalie a população de $npop$ indivíduos
Inicialize o ambiente paralelo com $np$ processadores
envie a população de $npop$ indivíduos para os $np$ processadores
Defina número indivíduos a evoluir $n_{ie} = npop/np$ por processador
**Enquanto** critério de parada não é alcançado **faça**
**Para** $i = 1, I$ **faça**
Aplique o operador de seleção SELDG($j, n_{ie}, npop$) em cada $j$-ésimo processador
Avalie $n_{ie}$ os indivíduos da população do processador
Aplique os operadores evolutivos e gere nova população do processador
**Fim para**
Envie os $n_{ie}$ indivíduos para os outros processadores
Receba os $n_{ie}$ indivíduos dos outros processadores
Atualize a população do processador
**Fim enquanto**
Finalize o ambiente paralelo
**Fim**

Figura 6 – Pseudocódigo para AEPDG

Para a viabilização deste modelo distribuído, é necessário a implementação de um procedimento de seleção diferenciado para um modelo distribuído com comportamento global (SELDG) A função SELDG deve permitir atualizar alguns indivíduos previamente determinados, porém, levando-se em conta toda a população global. Além disto, dado o perfil de aplicação em ambiente paralelo, é necessário que o procedimento de seleção gerencie, para cada processador, qual parcela da população deve ser evoluída e atualizada. Para a estratégia de inserção de pressão de seleção no processo pode-se adotar qualquer um dos métodos padrão de seleção amplamente conhecidos como, por exemplo, seleção por torneio, seleção por roleta, etc. A função que descreve este operador de seleção aplicado a um AEP distribuído é apresentada na Fig. 7. Uma descrição mais pormenorizada do funcionamento desta rotina será apresentada na próxima seção.

**Função operador de seleção SELDG do AEPDG**
**Início**
**Entrada** ID do processador ($j$), número de indivíduos a evoluir ($n_{ie}$), *npop*
**Para** $i = 1, n_{ie}$ **faça**
Selecione 2 indivíduos aplicando operador de seleção em *npop*
Aloque os indivíduos selecionados na população parcial nos índices $j.n_{ie} + i$
**Fim para**
**Retorne** população parcial
**Fim função**

Figura 7 – Pseudocódigo da função SELDG

## 3.1 Alocação da população entre os processadores

Para dividir a população entre os processadores, foi utilizado um método de partição no qual cada parcela de indivíduos é atribuída a um determinado processador, de modo que cada um é responsável por uma parte da população. Nos testes com 4 processadores e uma população de 100 indivíduos, o primeiro fica responsável pelos indivíduos de 1 a 25, o segundo de 26 a 50, o terceiro de 51 a 75 e o quarto de 76 a 100, como mostra a figura abaixo:

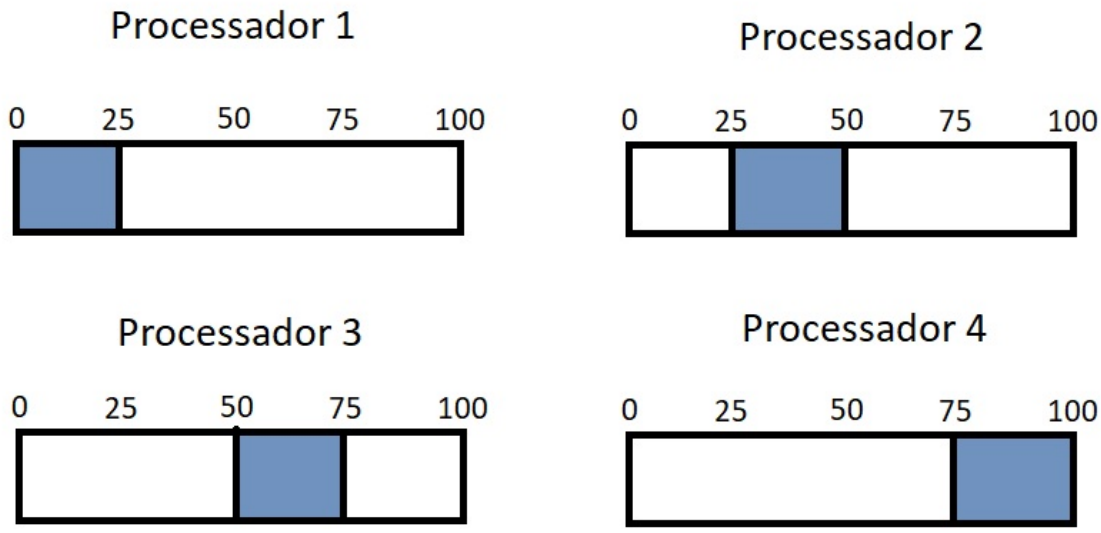


Figura 8 – Divisão dos indivíduos

Ao fazer uma evolução dos indivíduos, o processador irá ficar com os outros indivíduos desatualizados (o processador 1 ficará com os indivíduos de 26 a 100 desatualizados, por exemplo). Por isso é necessário que haja uma atualização para que que cada processador receba dos outros os indivíduos pelos quais ele não é responsável. Após realizar a atualização, todos os processadores ficarão com as populações idênticas novamente.

# 4 IMPLEMENTAÇÃO DO CÓDIGO E FUNÇÕES TESTE

Para escrever o código desse modelo, foi utilizada uma bilioteca científica para linguagem Python chamada PyGMO. Essa biblioteca possui uma grande quantidade de problemas de optimização e também de algoritmos, construidos em um ambiente altamente paralelizado, fazendo assim com que o código seja executado rapidamente.

Essa biblioteca dá a liberdade ao usuário de implementar seus próprios problemas ou algoritmos, apenas derivando das classes já prontas em seu código. A biblioteca nos oferece tanto problemas comuns como também problemas multi-objetivos, assim como a opção de fazê-los nós mesmos.

A biblioteca possui muitos métodos que envolvem o sistema de ilhas e arquipélagos, funcionando totalmente em paralelo, onde o usuário pode escolher a topologia a ser utilizada.

## 4.1 Problemas a serem resolvidos

O foco desse trabalho não é em resolver um problema específico, mas sim o método utilizado para resolvê-los. Portanto foi utilizado esse modelo para simular a resolução de vários problemas diferentes de otimização de funções que estão implementados com a ajuda da biblioteca PyGMO. A maior parte dos problemas escolhidos são de minimização, nos quais a solução real é zero, portanto a aptdão dos indivíduos começa com valores altos, e à medida que o oprocesso evolutivo avança,a busca vai achando valores cada vez menores, se aproximando cada vez mais da melhor solução. As funções utilizadas foram:

- Ackley
- De Jong
- Rastrigin
- Rosenbrock
- Schwefel

Para cada um dos problemas acima, foram feitas simulações com uma população de 120 indivíduos, variando o número de gerações que se passam entre cada atualização.

# 5 RESULTADOS

Seja $I$ o intervalo de migração (o número de gerações que se passam entre cada atualização). Quando $I$ for igual a 1, nunca haverá indivíduos desatualizados, pois após qualquer evolução dos indivíduos, eles são automaticamente atualizados. Ao aumentar esse número, as soluções vão piorando cada vez mais. Foram então feitas simulações com atrasos iguais a 1, 10, 15 e 30 gerações, esperando-se que quanto maior este número, pior seja a solução final. Além disso, para cada um dos casos acima foram feitas várias simulações variando o número de processadores entre 2, 4, 6, 8, 10 e 12, para saber qual será o efeito dessa variação nas soluções finais.

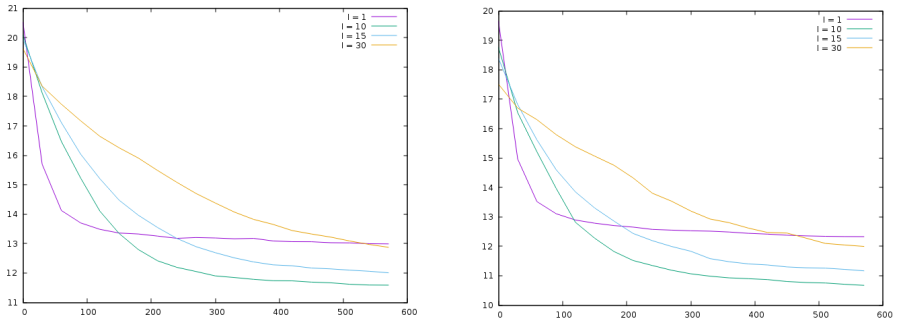


Figura 9 – Problema Ackley: média (esquerda) e melhor solução (direita) com 2 processadores

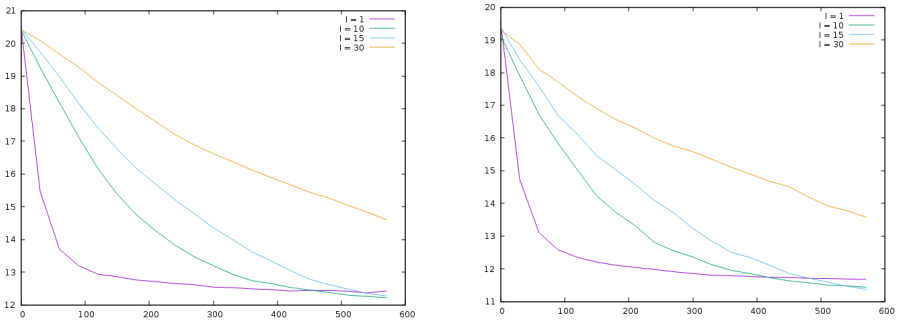


Figura 10 – Problema Ackley: média (esquerda) e melhor solução (direita) com 4 processadores

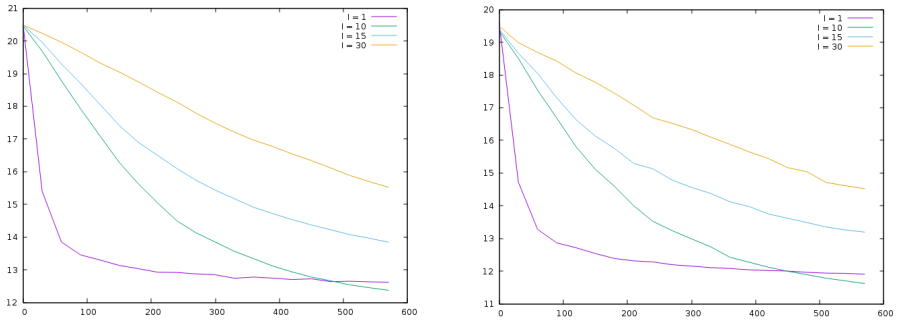


Figura 11 – Problema Ackley: média (esquerda) e melhor solução (direita) com 6 processadores

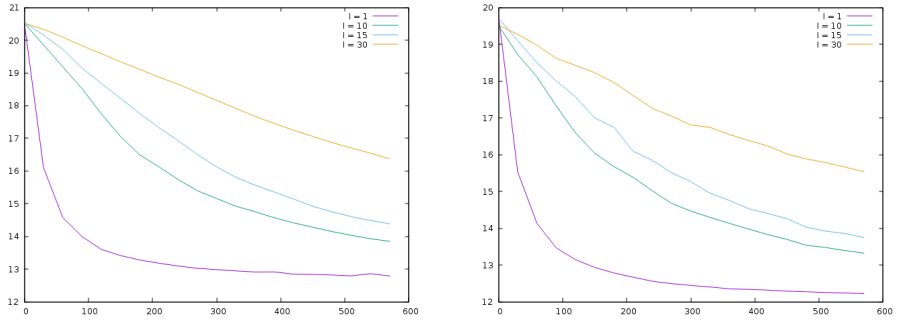


Figura 12 – Problema Ackley: média (esquerda) e melhor solução (direita) com 8 processadores

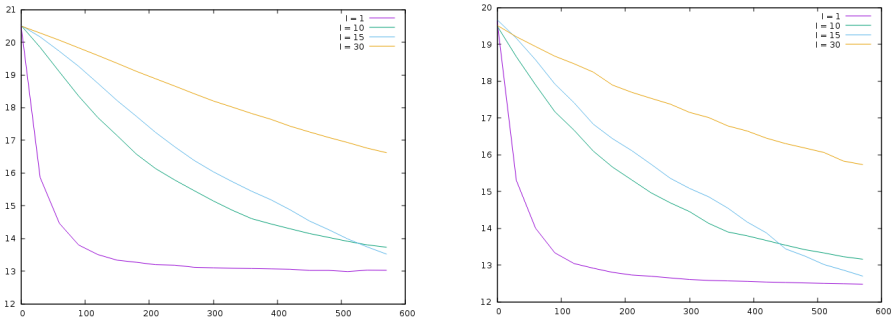


Figura 13 – Problema Ackley: média (esquerda) e melhor solução (direita) com 10 processadores

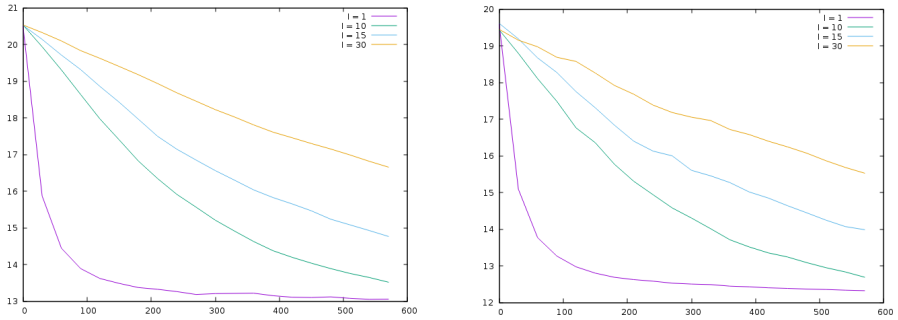


Figura 14 – Problema Ackley: média (esquerda) e melhor solução (direita) com 12 processadores

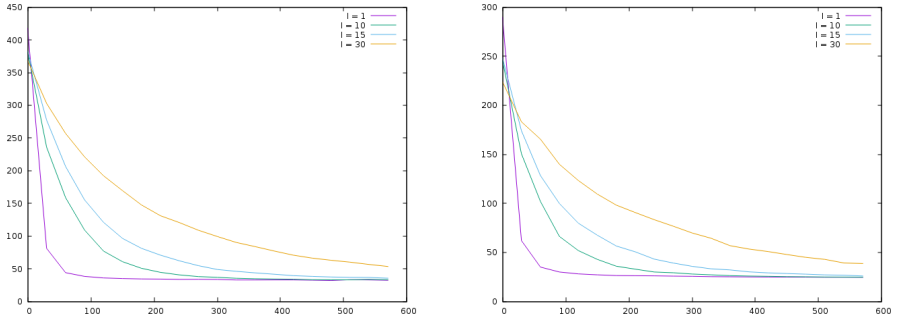


Figura 15 – Problema DeJong: média (esquerda) e melhor solução (direita) com 2 processadores

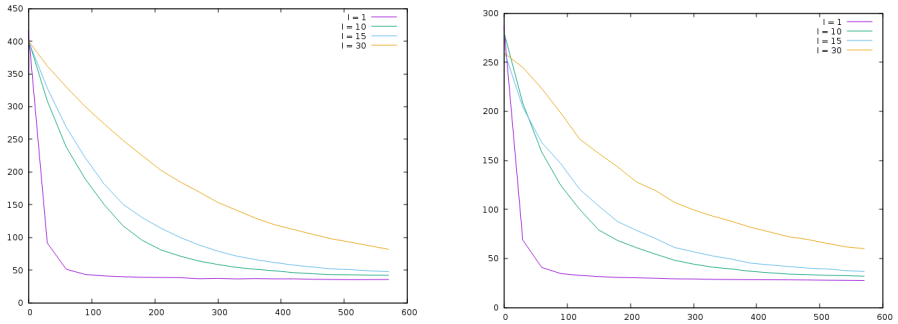


Figura 16 – Problema DeJong: média (esquerda) e melhor solução (direita) com 4 processadores

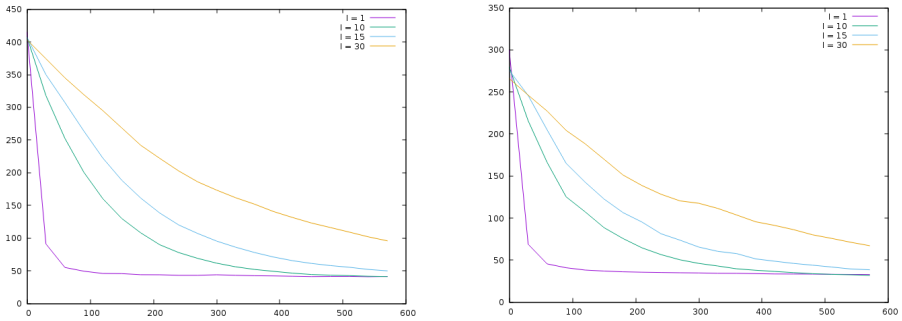


Figura 17 – Problema DeJong: média (esquerda) e melhor solução (direita) com 6 processadores

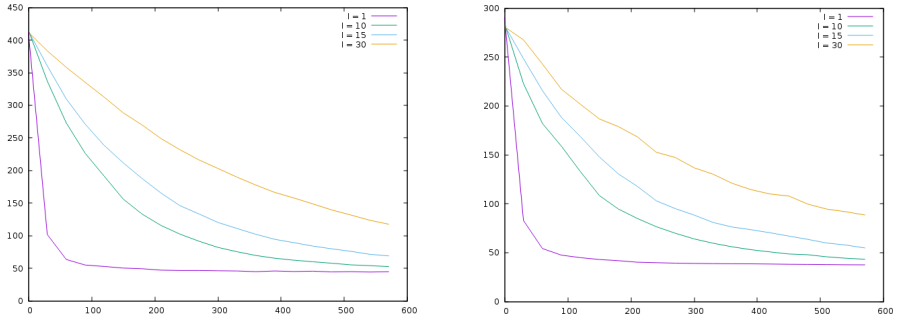


Figura 18 – Problema DeJong: média (esquerda) e melhor solução (direita) com 8 processadores

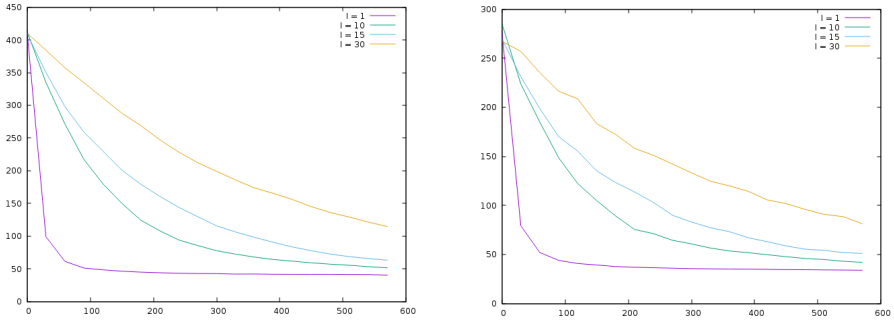


Figura 19 – Problema DeJong: média (esquerda) e melhor solução (direita) com 10 processadores

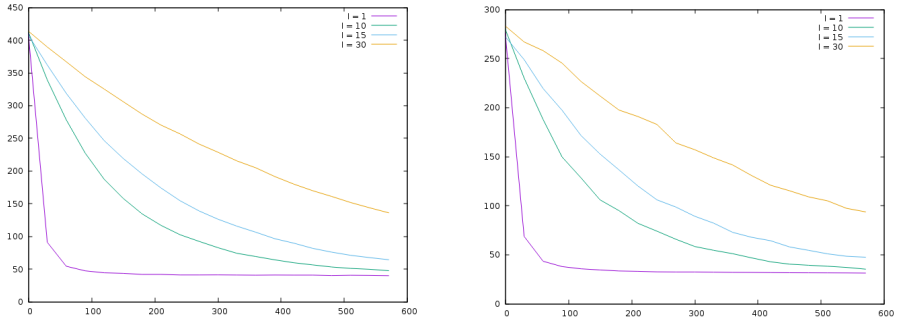


Figura 20 – Problema DeJong: média (esquerda) e melhor solução (direita) com 12 processadores

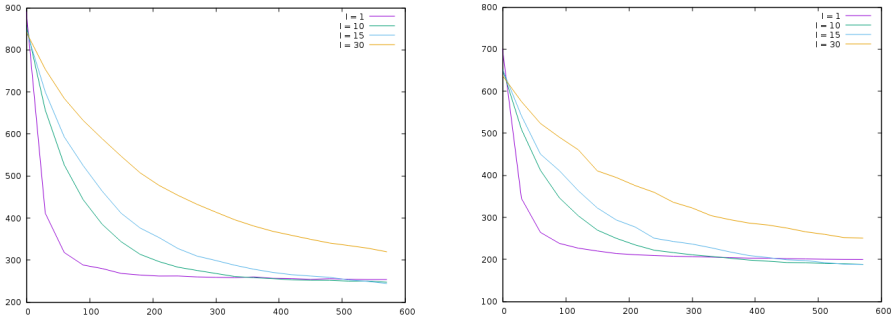


Figura 21 – Problema Rastrigin: média (esquerda) e melhor solução (direita) com 2 processadores

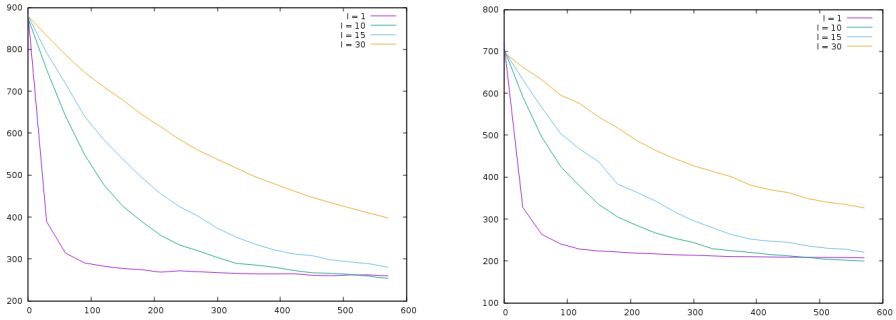


Figura 22 – Problema Rastrigin: média (esquerda) e melhor solução (direita) com 4 processadores

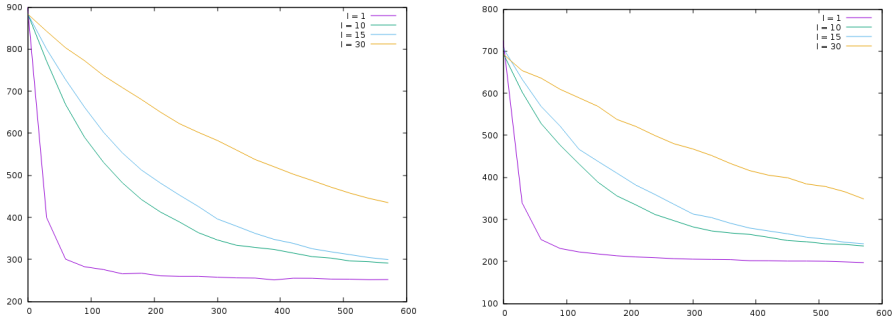


Figura 23 – Problema Rastrigin: média (esquerda) e melhor solução (direita) com 6 processadores

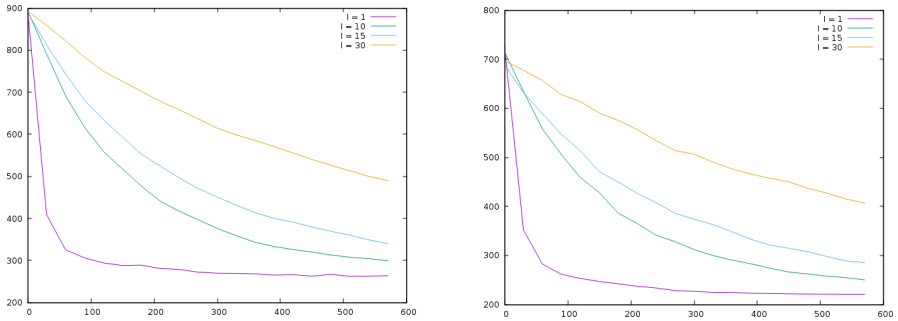


Figura 24 – Problema Rastrigin: média (esquerda) e melhor solução (direita) com 8 processadores

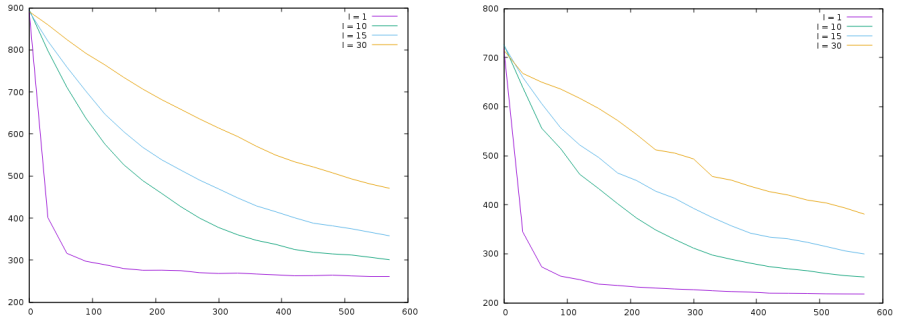


Figura 25 – Problema Rastrigin: média (esquerda) e melhor solução (direita) com 10 processadores

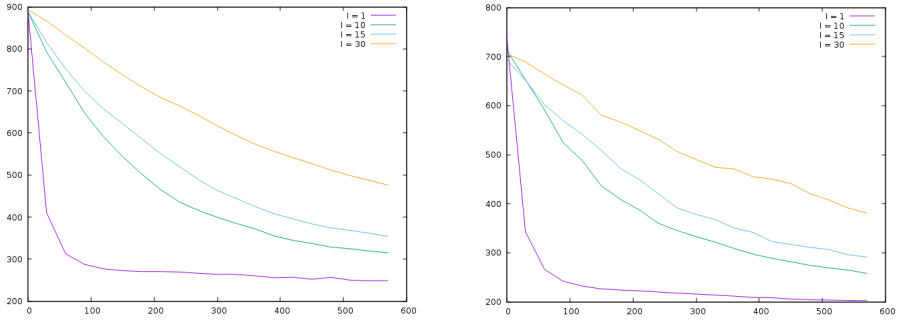


Figura 26 – Problema Rastrigin: média (esquerda) e melhor solução (direita) com 12 processadores

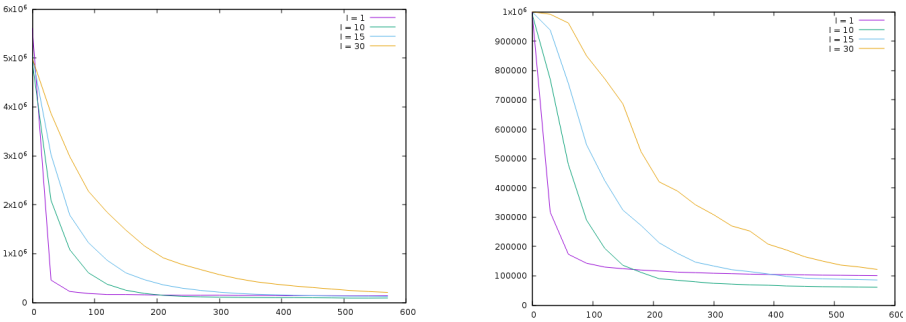


Figura 27 – Problema Rosenbrock: média (esquerda) e melhor solução (direita) com 2 processadores

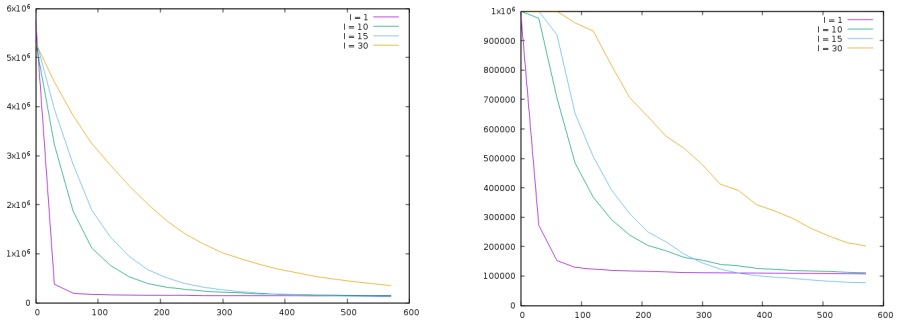


Figura 28 – Problema Rosenbrock: média (esquerda) e melhor solução (direita) com 4 processadores

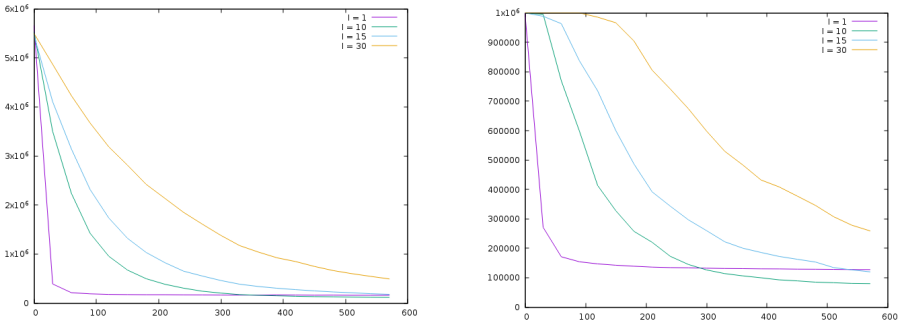


Figura 29 – Problema Rosenbrock: média (esquerda) e melhor solução (direita) com 6 processadores

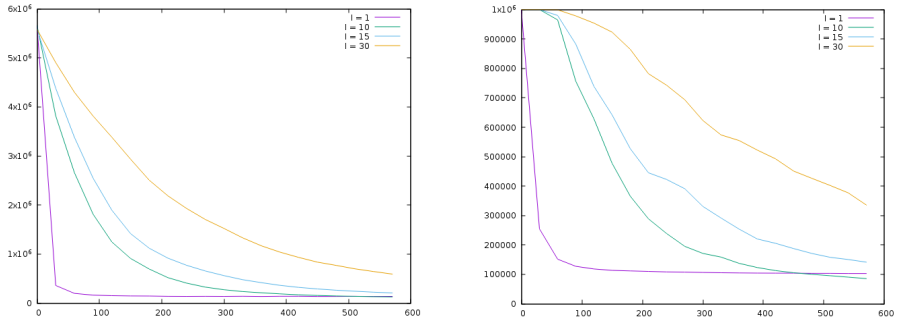


Figura 30 – Problema Rosenbrock: média (esquerda) e melhor solução (direita) com 8 processadores

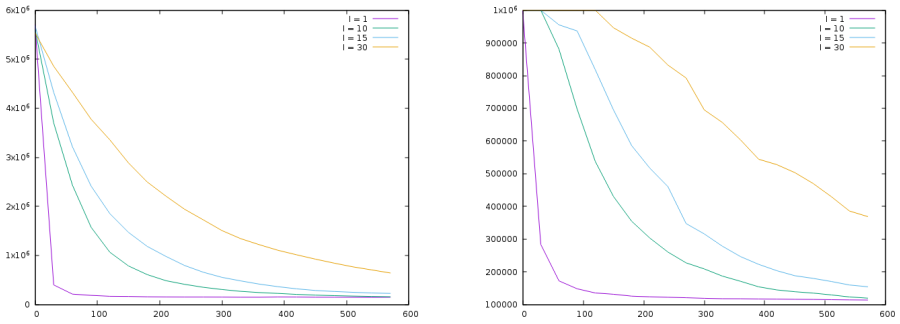


Figura 31 – Problema Rosenbrock: média (esquerda) e melhor solução (direita) com 10 processadores

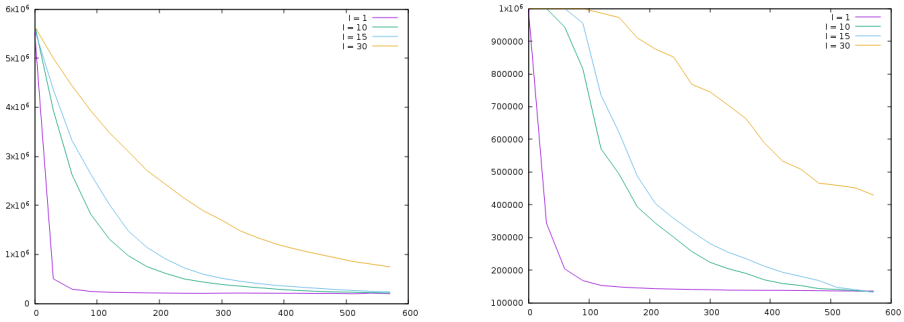


Figura 32 – Problema Rosenbrock: média (esquerda) e melhor solução (direita) com 12 processadores

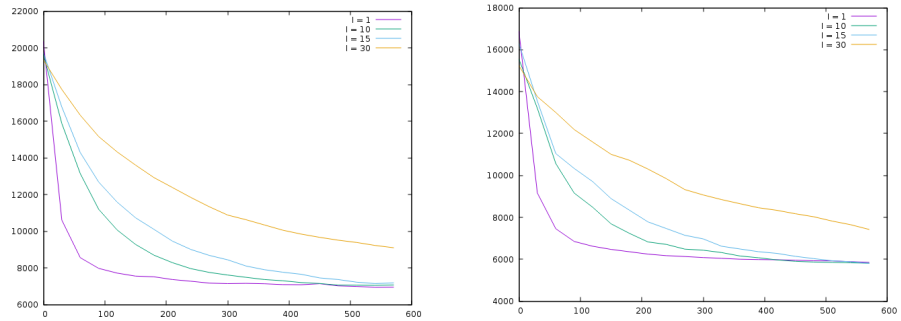


Figura 33 – Problema Schwefel: média (esquerda) e melhor solução (direita) com 2 processadores

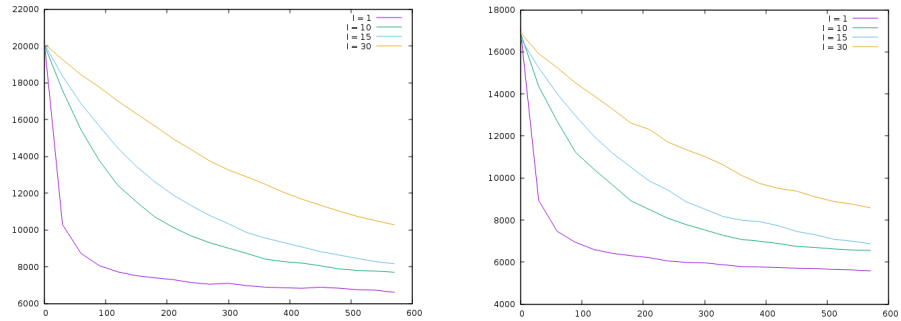


Figura 34 – Problema Schwefel: média (esquerda) e melhor solução (direita) com 4 processadores

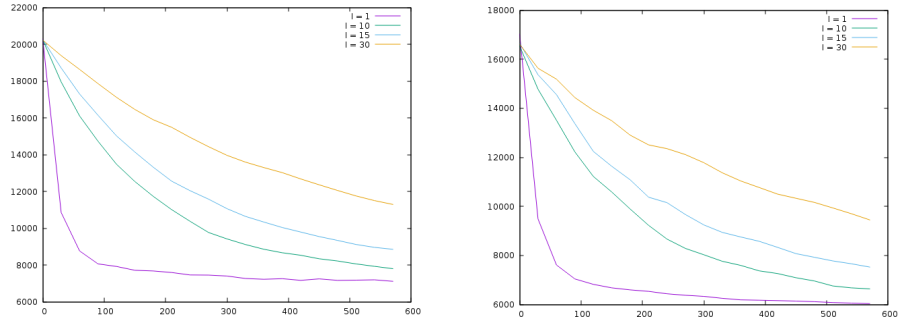


Figura 35 – Problema Schwefel: média (esquerda) e melhor solução (direita) com 6 processadores

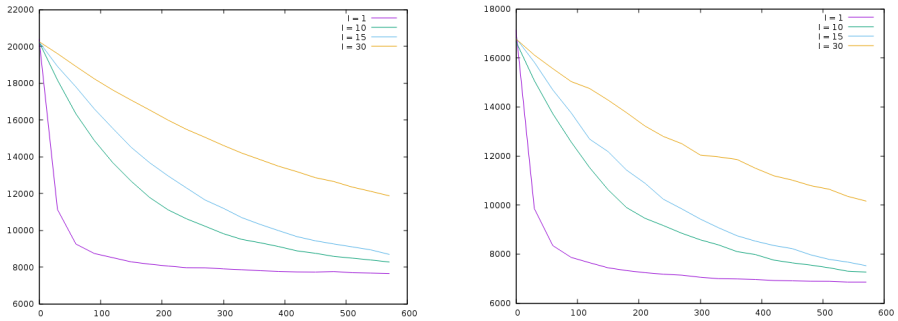


Figura 36 – Problema Schwefel: média (esquerda) e melhor solução (direita) com 8 processadores

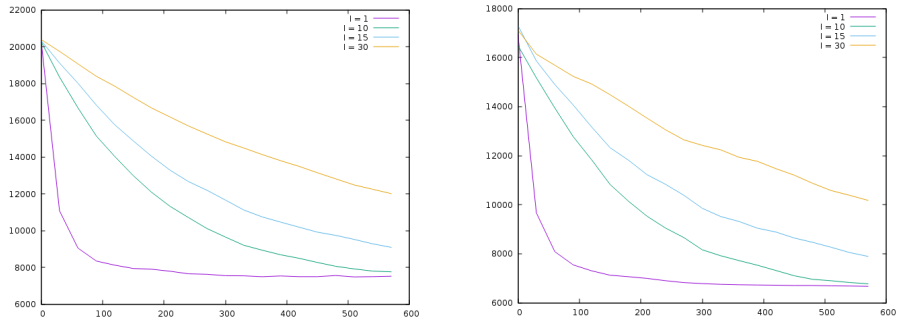


Figura 37 – Problema Schwefel: média (esquerda) e melhor solução (direita) com 10 processadores

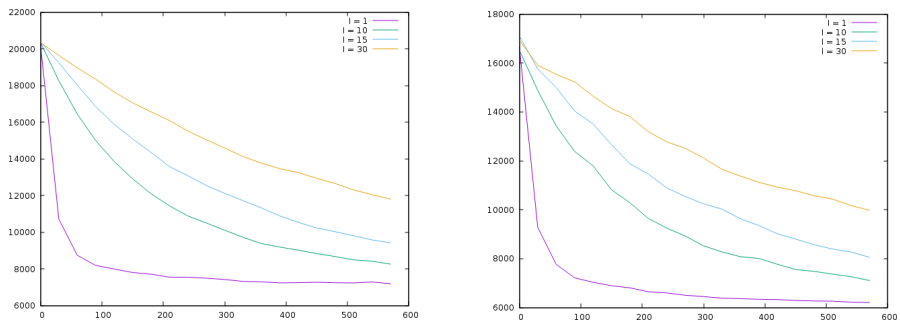


Figura 38 – Problema Schwefel: média (esquerda) e melhor solução (direita) com 12 processadores

# 6 ANÁLISE ESTATÍSTICA

Foi feita uma análise estatística para determinar como os 2 parâmetros (número de processadores e intervalo de migração) estão afetando a qualidade das soluções. Utilizando-se do Software RStudio, foram plotados gráficos do tipo BoxPlot para ver o quanto as soluções ficam piores ao aumentar o número de processadores ou o intervalo de migração:

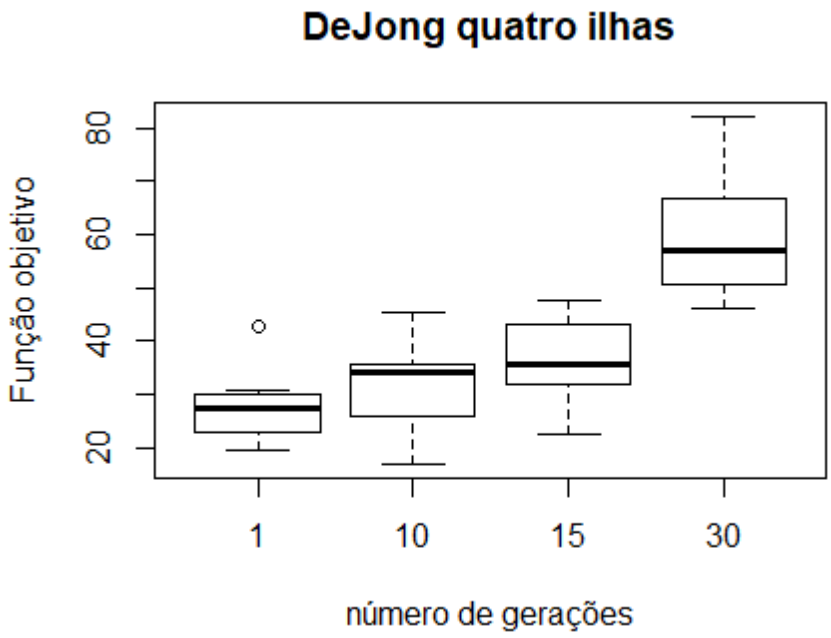


Figura 39 – Variação da qualidade das soluções nos diferentes intervalos de migração

Como pode-se observar, quanto maior o intervalo de migração pior é a solução final da simulação. Podemos ver também que as soluções finais pioram de uma maneira visível com o aumento do número de processadores:

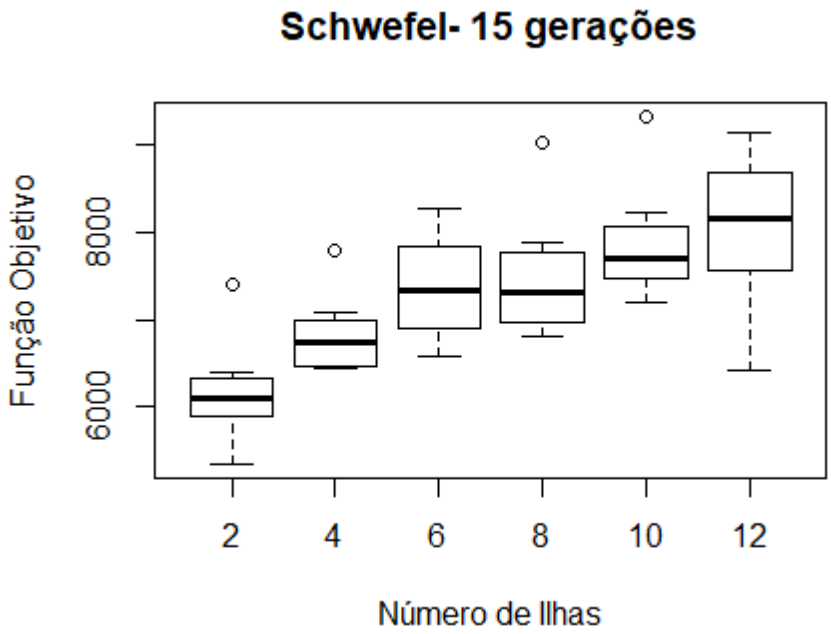


Figura 40 – Variação da qualidade das soluções para diferentes números de processadores

# 7 CONCLUSÃO E TRABALHOS FUTUROS

Com base nos resultados podemos perceber que quanto mais aumentamos o valor de I (intervalo de migração), pior é a solução final. As simulações com I = 1 foram as primeiras a atingir um estado próximo do ótimo e as primeiras a estabilizarem. É possível perceber que esse sistema distribuído entre vários processadores seria vantajoso, pois os resultados com atrasos não estão muito longe do serial em relação à solução obtida no fim da simulação, o que faria com que houvesse um ganho na solução final. Esse trabalho então mostrou que é possível que tal modelo seja eficiente ao ser rodado com vários processadores, ou seja, o objetivo foi alcançado em estabelecer essa comparação entre o serial e o nosso modelo paralelo.

Como trabalhos futuros, pode-se pensar em desenvolver uma estratégia para minimizar a defasagem da população, baseada em modelos de aproximação. Pode-se também implementar esse modelo para ser executado em um hardware paralelo para que se possa medir o quão grande é o ganho no desempenho computacional desse modelo.

# REFERÊNCIAS

[1] Yue-Jiao Gong, Wei-Neng Chen, Zhi-Hui Zhan, Jun Zhang, Yun Li, Qingfu Zhang, and Jing-Jing Li. Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. 2015.

[2] J. I Hidalgo, F. Fernández de Vega, J. Lanchares, and D. Lombraña-González. Is the island model fault tolerant? 2007.

[3] Nandar Lynn, Mostafa Z. Ali, and Ponnuthurai Nagaratnam Suganthan. Population topologies for particle swarm optimization and differential evolution. 2017.

[4] Thiago T. Magalhães, Eduardo Krempster, and Helio J. C. Barbosa. Migration policies to improve exploration in parallel island models for optimization via metaheuristics. 2015.

[5] Thiago T. Magalhães, Luiz Henrique C. Nascimento, Eduardo Krempster, Douglas A. Augusto, and Helio J. C. Barbosa. Hybrid metaheuristics for optimization using a parallel islands model. 2014.

[6] Rafael Nogueras and Carlos Cotta. An analysis of migration strategies in island-based multimemetic algorithms. 2014.

[7] Ryoji Tanabe and Alex Fukunaga. Evaluation of a randomized parameter setting strategy for island-model evolutionary algorithms. 2013.

[8] Darrel Whitley, Soraya Rana, and Robert B. Heckendorn. The island model genetic algorithm: on separability, population size and convergence. 1999.