EGIDIO DE CARVALHO RIBEIRO JUNIOR

UM ALGORITMO GENÉTICO PARALELO AUTOADAPTÁVEL MISTO PARA A OTIMIZAÇÃO MULTIMODAL COM E SEM RESTRIÇÕES

São Luis - Maranhão

2017, v-1.0.0

EGIDIO DE CARVALHO RIBEIRO JUNIOR

UM ALGORITMO GENÉTICO PARALELO AUTOADAPTÁVEL MISTO PARA A OTIMIZAÇÃO MULTIMODAL COM E SEM RESTRIÇÕES

Dissertação apresentada à Universidade Estadual do Maranhão, como parte das exigências do Programa de Pós-Graduação em Engenharia de Computação e Sistemas, para obtenção do título de mestre em engenharia de computação e sistemas.

Universidade Estadual do Maranhão – UEMA

Engenharia de Computação e Sistemas

Programa de Pós-Graduação

Orientador: Prof. Dr. Omar Andres Carmona Cortes

São Luis - Maranhão

2017, v-1.0.0

EGIDIO DE CARVALHO RIBEIRO JUNIOR
UM ALGORITMO GENÉTICO PARALELO AUTOADAPTÁVEL MISTO PARA A OTIMIZAÇÃO MULTIMODAL COM E SEM RESTRIÇÕES/ EGIDIO DE CARVALHO RIBEIRO JUNIOR. – São Luis - Maranhão, 2017, v-1.0.0-
83 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Omar Andres Carmona Cortes

Dissertação de Mestrado – Universidade Estadual do Maranhão – UEMA
Engenharia de Computação e Sistemas
Programa de Pós-Graduação, 2017, v-1.0.0.

1. Algoritmos Genéticos. 2. OpenMP. 2. Algoritmos autoajustáveis. I. Orientador: Prof. Dr. Omar Andres Carmona Cortes II. Universidade Estadual do Maranhão. III. Engenharia de Computação e Sistemas. IV. Um Algoritmo Genético Paralelo Autoadaptável Misto para a Otimização Multimodal com e sem restrições.

EGIDIO DE CARVALHO RIBEIRO JUNIOR

UM ALGORITMO GENÉTICO PARALELO AUTOADAPTÁVEL MISTO PARA A OTIMIZAÇÃO MULTIMODAL COM E SEM RESTRIÇÕES

Dissertação apresentada à Universidade Estadual do Maranhão, como parte das exigências do Programa de Pós-Graduação em Engenharia de Computação e Sistemas, para obtenção do título de mestre em engenharia de computação e sistemas.

Trabalho aprovado. São Luis - Maranhão, 14 de agosto de 2017:

Prof. Dr. Omar Andres Carmona Cortes
Orientador

Prof. Dr. Bruno Feres de Souza
(UFMA)

Prof. Dr. Osvaldo Ronald Saavedra Mendez
(UFMA)

São Luis - Maranhão
2017, v-1.0.0

A Deus que como mestre maior sempre guia meus passos.

Agradecimentos

Agradeço às minhas avós que com seus exemplos moldaram meu caráter.

Aos meus pais, pelo incentivo aos estudos.

Em especial, ao meu orientador, pela paciência, apoio e confiança.

Resumo

Os algoritmos genéticos são algoritmos de busca que utilizam uma série de parâmetros para encontrar a melhor solução de um dado problema. A escolha desses parâmetros é um desafio, pois diferentes problemas podem exigir diferentes configurações. Na maioria das vezes, essa escolha é feita pelo método da tentativa-e-erro que além de demandar tempo pode não levar à melhor configuração possível. Outra possibilidade é a utilização de design de experimentos (fator n^k) que além de ser limitada à quantidade de fatores k, exige que muitas simulações sejam feitas antes da determinação da melhor configuração que também é dependente do problema a ser resolvido. É nesse contexto que surgem os algoritmos autoadaptáveis, permitindo a melhor seleção tanto de taxas quanto de operadores genéticos em tempo de execução. Dessa forma, esta dissertação desenvolve um algoritmo autoajustável que além de escolher entre 4 tipos diferentes de operadores de cruzamento e 4 operadores de mutação, permite também adaptar em tempo de execução as taxas de cruzamento e mutação. Superado o desafio de encontrar a melhor configuração, resta aumentar a velocidade da execução, pois à medida que mais parâmetros são ajustados, maior poder computacional é exigido. Para solucionar esse problema, lança-se mão da utilização da computação paralela que se tornou acessível e popular após a criação e redução dos custos dos processadores multi-núcleos. Nesta dissertação, o paralelismo é obtido através da utilização do OpenMP, que é uma interface de programação de aplicativo (API) que fornece um modelo portável e escalável para programadores criarem aplicações paralelas. Para avaliar o desempenho do algoritmo genético autodaptativo foram utilizados benchmarks padrão, como as funções Schwefel, Griewank, Rastringin, Rosenbrock, variações das duas últimas utilizadas (Shifted Rastring e Shifted Rosenbrock) e uma aplicação real denominada despacho econômico dinâmico de energia elétrica. A avaliação foi feita em termos de qualidade da solução utilizando-se ANOVA e teste de Tukey. Em termos de paralelismo, a avaliação foi feita utilizando-se as métricas de speedup e eficiência.

Palavras-chave: Algoritmos Genéticos. OpenMP. Algoritmos autoajustáveis.

Abstract

Genetic algorithms are search algorithms that use a set of parameters to find the best solution to a given problem. These parameter's choice is a challenge because different problems can demand different sets. Most of the times, this choice is made using the trial and error method that demands time and may not lead to the best possible setting. Other option is to use the experiment design (factor n^k) that further on limited to the amount of k factors, demands that multiple simulations are done before the definition of the best setting, which is also dependent of the problem that is being solved. In this context, the self-adaptive algorithms arise, allowing the best selection of rates and genetic operators within execution time. Thereby, this work develops a self-adaptive algorithm that in addition of choosing between four different kinds of crossover and four mutation operators also adapts in execution's time the rates of crossover and mutation. Having overcome the challenge of finding the best configuration, remains increasing the execution time, because as more parameters are adjusted a higher computational power is required. To solve this problem, it is used the parallel computing, which became acessible and popular after the criation and cost reduction of multicore processors. In this work, the parallelism is achieved using the OpenMp, which is an application programming interface (API) that supplies a portable and scalable model that allows programmers to create parallel applications. To evaluate the self-adaptive algorithm performance standard benchmark functions were used, such as Schwefel, Griewank, Rastringin, Rosenbrock, variations of the last two ones (Shifted Rastringin and Shifted Rosenbrock) and a real application called Dynamic Economic Dispacth. The evaluation was made in terms of the solution's quality using ANOVA and Tukey's test. In terms of parallelism, the evaluation was done using speedup and eficiency metrics.

Keywords: Genetic Algorithms. OpenMp . Self-Adaptive Algorithms.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AG	Algoritmos Genéticos
API	Interface de Programação de Aplicativo
DED	Despacho Econômico Dinâmico de Energia Elétrica
DMS	Diferença Mínima Significativa
FA	Fator de Aceleração
GCC	Coleção de Compiladores GCC
GRI	Função Griewank
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
MOD	Modo de Execução do Algoritmo Genético
OPENMP	Especificações Abertas para Multiprocessamento
RAS	Função Rastringin
ROS	Função Rosenbrock
SCW	Função Schwefel
SRAS	Função Shifted Rastringin
SROS	Função Shifted Rosenbrock

Sumário

1 Introdução

Os algoritmos genéticos são algoritmos de busca que utilizam conceitos da teoria da seleção natural de Darwin na tentativa de encontrar a melhor solução em um espaço de busca complexo. Utilizando mecanismos como cruzamento, mutação e seleção, esses algoritmos exploram o espaço de busca de forma heurística na tentativa de encontrar regiões promissoras.

O campo de aplicação dos algoritmos genéticos é muito vasto. Oliveira, Oliveira e Gomes (2016) utilizaram algoritmos genéticos para encontrar a melhor localização para a instalação de geradores de energia eólica e de biomassa, minimizando assim perdas de energia, emissões e os custos de operação. A análise do mercado de ações motivou Carvalho, Almeida e Silva (2015) a usar os algoritmos genéticos para encontrar a melhor combinação de estratégias para aumentar o lucro de um investidor. Lin e Ku (2014) utilizaram algoritmos genéticos para otimizar padrões de paradas para o transporte ferroviário de passageiros. O refinamento de funções de pertinência Fuzzy foi estudado por Cortes (2011) . Segundo Linden (2008), os algoritmos genéticos podem ser usados para modelar a estrutura do subsolo. O trabalho de Montesinos, Arnoso e Vieira (2005, páginas 301 a 316), na mesma área, utilizou os algoritmos genéticos para inversão de dados gravitacionais. Alsaeedan e Menai (2015) resolveram o problema de retirar a ambiguidade de palavras de um determinado contexto através dos algoritmos genéticos. Segundo Herrera, M.Lozano e Verdegay (1998):

> "Os algoritmos genéticos tiveram um grande êxito nos problemas de busca e otimização. A razão de grande parte desse sucesso é a habilidade deles em usar as informações acumuladas sobre um espaço de busca inicialmente desconhecido para influenciar as próximas buscas em espaços úteis".

Para realizar essa busca, os algoritmos genéticos utilizam uma série de parâmetros, como por exemplo, as taxas e os operadores de cruzamento e mutação. Encontrar o melhor conjunto desses parâmetros é um desafio, pois diferentes problemas podem exigir diferentes configurações. Essa escolha costuma ser feita pelo método da tentativa-e-erro, no entanto, esse método demanda muito tempo e pode não levar à melhor configuração possível. Outra possibilidade é a utilização de projeto de experimentos (fator n^k) que além de ser limitada a um número normalmente pequeno de possibilidades, exige que muitas simulações sejam feitas antes da determinação da melhor configuração.

É nesse contexto que os algoritmos autoajustáveis ou autoadaptáveis se tornam atrativos, sendo sua maior vantagem em relação aos algoritmos genéticos tradicionais o ajuste automático de seus parâmetros, o que leva a descoberta de melhores resultados. Por

esse motivo o autoajuste tem sido o foco de trabalhos como Serpell e Smith (2010) e Korejo et al. (2013) que implementaram respectivamente a autoadaptação e a adaptação apenas do operador e da taxa de mutação. Abbass (2002) desenvolveu o autoajuste apenas da taxa de cruzamento e de mutação. Deb e Beyer (2001) não usa a mutação no seu algoritmo autoadaptável. Esta dissertação desenvolveu um algoritmo autoajustável diferente dos demais pois implementa o autoajuste para as taxas de cruzamento e mutação e o ajuste automático para os operadores de cruzamento e mutação. Essa diferença será explicada em detalhes no Capítulo 4. O algoritmo autoajustável misto conseguiu resultados melhores que a versão autoajustável simples.

Muito embora esse autoajuste permita que uma série de problemas sejam solucionados por um algoritmo genético, dependendo do tipo de aplicação, o custo de processamento pode se tornar proibitivo, em especial aquelas aplicações cuja resposta precisa ser rápida ou em tempo real. Por exemplo, em uma bolsa de valores um tempo de execução alto pode inviabilizar a solução proposta. Essa foi a motivação para esta dissertação buscar a otimização da performance pela execução em paralelo do algoritmo. Utiliza-se a interface de programação de aplicativos OpenMp para realizar esta tarefa. Segundo Mehta e Gabriel (2014) :

> "Processar grandes quantidades de dados é um cenário comum para aplicações paralelas. Enquanto aplicações de memória distribuída são capazes de melhorar a performance de suas operações de entrada e saída utilizando bibliotecas de entrada e saída paralela, não existe um suporte para operações de entrada e saída para aplicações que usam o modelo de programação de memória compartilhada como o OpenMP disponíveis hoje."

Essa paralelização de algoritmos genéticos autoadaptáveis ainda é um tema pouco explorado. Nenhum trabalho com este escopo foi encontrado na literatura. Por exemplo, Lu et al. (2015), Xiao et al. (2015) e Miranda, Montevechi e Pinho (2015) desenvolveram algoritmos genéticos adaptáveis ou autoadaptáveis, sem no entanto paralelizá-los.

Pessini (2003) e Silva, Oliveira e Veras (2013) fizeram dissertações sobre algoritmos genéticos paralelos, mas estes não eram autoadaptáveis. O trabalho de Kalegari (2015) trata da paralelização do algoritmo de evolução diferenciada. O algoritmo não é autoadaptativo. A paralelização, também diferiu pois utiliza clusters de computadores enquanto esta dissertação cria múltiplas threads de execução dentro do mesmo computador, utilizando sua arquitetura de vários núcleos.

As funções de benchmark utilizadas nesta dissertação são usadas com frequência na literatura para medir o desempenho de algoritmos. Trabalhos como o de Cortes e Silva (2010), Mirjalili (2016), Koupaei, Hosseini e Ghaini (2016), Tsoulos, Tzallas e Tsalikakis (2016) , Trivedi et al. (2017) e Yadav, Yadav e Kim (2017) utilizaram pelo uma ou mais das funções de benchmark deste estudo para avaliar seus algoritmos.

A função de Despacho econômico dinâmico de energia foi escolhida por se tratar de função de difícil solução, onde alguns milésimos podem ocasionar uma variação de milhares de unidades no resultado final. Os resultados desta dissertação serão comparados com os resultados dos artigos Niknam e Golestaneh (2012), página 429, tabela 3, Balamurugan e Subramanian (2007) página 156 e Chakraborty et al. (2012) página 201, tabela 4. Estes artigos foram escolhidos por apresentar seus parâmetros de configuração, permitindo assim uma comparação mais adequada.

1.1 Objetivos

A escolha manual de parâmetros de execução de algoritmos genéticos é uma tarefa de tentativa e erro. Existem vários parâmetros de configuração como os operadores de cruzamento e mutação, taxas de cruzamento, mutação, nascimento e morte. O objetivo desta dissertação é desenvolver um algoritmo autoajustável mais eficiente que sua versão sem ajuste e que rode em paralelo para entregar o resultado mais rápido que sua versão sequencial. Para alcançar esse resultado, sete objetivos específicos devem ser alcançados.

1. Estudar o comportamento dos operadores genéticos.
2. Estudar o comportamento das probabilidades de cruzamento e mutação.
3. Desenvolver o algoritmo genético autoadaptável misto, visando a automatização da escolha dos parâmetros de configuração (operadores e probabilidades) que encontre as melhores respostas para as funções propostas.
4. Comparar o resultado do algoritmo desenvolvido com a escolha aleatória de parâmetros.
5. Estudar a programação paralela e a a interface de programação de aplicativo chamada OpenMP.
6. Paralelizar o algoritmo autoadaptável misto usando OpenMp.
7. Comparar a diferença de performance do algoritmo autoadaptável misto paralelizado e do sequencial.

1.2 Hipóteses

Esta dissertação busca testar duas hipóteses. A primeira hipótese deseja confirmar que o algoritmo autoadaptativo misto gerará resultados melhores que o código sem autoadaptação na maioria das vezes que a diferença for considerada significativa .

A segunda hipótese busca confirmar que a paralelização do algoritmo autoadaptável misto usando OpenMp irá melhorar a performance do algoritmo em comparação a versão sequencial sem perda significativa de qualidade.

1.3 Justificativa

As aplicações dos algoritmos genéticos são extremamente variadas. Para cada aplicação, existem parâmetros que mais se adequam ao caso. Essa escolha é fundamental para o sucesso do algoritmo. Um conjunto inadequado de parâmetros pode fazer que o software nunca encontre seu ponto ótimo. Para definir melhor a magnitude desse problema, o design de experimentos certamente pode ajudar. Segundo Cavazzuti (2013) no seu livro Design de Experimentos:

> "Dentro da teoria de otimização, um experimento é uma série de testes nos quais as variáveis de entrada são alteradas de acordo com uma regra dada para identificar as razões das alterações nas saídas."

A quantidade de experimentos é definido pela quantidade de entradas em estudo possíveis. Para um experimento factorial 2^K, com dois fatores (2^2) existirão quatro possibilidades. Caso a probabilidade de cruzamento e a de mutação estejam sendo avaliadas, onde P_c é a probabilidade de cruzamento e P_m é a de mutação, as quatro possibilidades poderiam ser $P_c = 0,5$, $P_c = 0,9$, $P_m = 0,01$ e $P_m = 0,05$. Logo, à medida que se aumenta a quantidade de variáveis aumenta-se a quantidade de experimentos exponencialmente. Como para diferentes problemas pode haver fatores intermediários entre $P_c = 0,5$ e $P_c = 0,9$ que sejam mais interessantes para os mesmos, como por exemplo $P_c = 0,75$. Basta que esse nível ou fator seja acrescentado para o projeto de experimento passar a ser 3^k.Em outras palavras, a medida que possibilidades são adicionadas ao experimento a quantidade de execuções cresce em demasia.

O objetivo da melhoria da performance dessas soluções, não perde em importância. Patel e Gulati (2014) ao analisar a performance de aplicações, escrevem:

> "Baixa performance custa à indústria de software milhões em dinheiro anualmente na forma de receitas perdidas, custos de hardware relações com clientes prejudicadas e produtividade diminuída."

Dada a importância da performance, trabalhos como Pessini (2003), Silva, Oliveira e Veras (2013) , Kalegari (2015) e Devos, Downey e Duponchel (2014) buscaram melhorar o tempo de resposta de algoritmos genéticos através da paralelização.

1.4 Descrição do Trabalho

Esta dissertação é dividida em seis capítulos. O primeiro capítulo trata desta introdução, dos objetivos, das hipóteses e da justificativa. O segundo capítulo inicia discorrendo sobre os algoritmos genéticos, sua representação, e como se dá seu funcionamento. Cada passo do algoritmo será detalhado, como o processo de seleção utilizado, os operadores de cruzamento e mutação, o elitismo e das formas possíveis de trabalharmos com restrições e em especial da utilizada nesta dissertação. No terceiro capítulo serão discutidos os conceitos da programação paralela, os tipos de compartilhamento de memória, os índices de desempenho, as limitações e a interface de programação de aplicativos OpenMP.

O quarto capítulo descreve os algoritmos genéticos autoajustáveis e os ajustáveis para em seguida apresentar o algoritmo autoajustável misto desenvolvido nesta dissertação. Também será apresentada a solução paralelizada. O quinto capítulo tem foco nos experimentos, nas suas configurações, nas funções de benchmark usadas e em especial na função de despacho econômico dinâmico de energia elétrica. Esse capítulo também mostra os resultados desta dissertação. Inicialmente elencará os resultados da comparação do algoritmo autoajustável misto desenvolvido nesta dissertação com o algoritmo genético com operadores aleatórios e com todas as combinações de operadores possíveis. Por fim, irá comparar a performance do algoritmo autoajustável misto paralelo a do algoritmo autoajustável misto de thread única. O sexto capítulo mostra as conclusões desta dissertação.

2 Algoritmos Genéticos

Os algoritmos genéticos foram propostos pela primeira vez em 1975 por John Holland da Universidade de Michigan. Inspirados em regras descobertas da biologia, mimetizam o processo de evolução das espécies. Segundo esse processo, entre vários indivíduos presentes em determinado ambiente, alguns devido a certas características específicas, possuem uma maior chance de sobrevivência e consequente reprodução (indivíduos mais aptos). Como graças ao processo de reprodução as características dos pais tendem a ser repassadas, as características que os tornaram mais aptos tendem a ser mais frequentes nas gerações futuras, aumentando assim a aptidão da população como um todo. Durante o processo de reprodução podem ocorrer falhas, as mutações. Esse fenômeno tanto pode ser prejudicial quanto pode acabar incorporando uma característica desejável não contida no conjunto de genes dos seus pais.

Os algoritmos genéticos são uma classe de Algoritmos Evolutivos. Segundo BENTLEY (1999), os algoritmos evolutivos são uma coleção de métodos de busca, otimização, aprendizagem e modelagem de dados inspirados na evolução biológica. Cortes (2004) deixa esse conceito ainda mais claro:

> "A teoria dos algoritmos evolutivos baseia-se na teoria da evolução de Darwin, onde os indivíduos mais fortes interagem entre si propagando seu genes e gerando descendentes mais aptos às novas condições."

A Figura 1 mostra o funcionamento de um algoritmo genético. O passo inicial do algoritmo é a geração da população inicial. Cada gene de cada indivíduo é gerado aleatoriamente dentro do domínio do problema. Cada solução, ou indivíduo, precisa ser avaliada, ou seja, sua qualidade precisa ser mensurada. Para essa tarefa utiliza-se a função de aptidão. Em outras palavras, a função de aptidão, usando os genes como variáveis, permite mensurar o quão bom ou ruim um indivíduo pode ser. Quanto melhor a solução mais apto é o indivíduo. Nesse contexto, um conjunto de soluções ou indivíduos é denominado população e pode conter indivíduos em qualquer parte do espaço de busca. Cabe ao algoritmo genético a busca da melhor solução. Muito embora um tempo infinito garanta que a solução ótima será encontrada, nem sempre isso é possível. O critério de parada destes algoritmos costuma ser o número de gerações. Enquanto o critério de parada não é atingido, o algoritmo executa as operações de seleção, cruzamento e mutação que serão detalhadas nas próximas seções.

Algoritmo 1 Algoritmo Genético

```
Populacao ← geraPopulacaoInicial();
enquanto Criterio parada não for alcançado faça
    PopulacaoTemp ← Selecao(Populacao);
    PopulacaoTemp ← Cruzamento(PopulacaoTemp);
    PopulacaoTemp ← Mutacao(PopulacaoTemp);
    Elitismo(PopulacaoTemp, Populacacao);
    Populacacao ← PopulacaoTemp;
fim enquanto
```

Figura 1 – Algoritmo Genético

2.1 Representação

Nos algoritmos genéticos, os indivíduos são formados por um conjunto de genes, representados normalmente por um vetor. Esse vetor pode ser codificado utilizando números binários ou reais, conforme a Figura 2. Linden (2012) descreve uma terceira forma, a representação em árvore.

John Holland iniciou os estudos sobre algoritmos genéticos utilizando números binários. Ele acreditava que o desempenho do algoritmo melhoraria quando o paralelismo implícito inerente a ele fosse maximizado e demonstrou que um alfabeto binário conseguiria maximizar esse paralelismo. Essa codificação também possui outras vantagens. Ela é mais fácil de implementar, manipular e analisar. No entanto, ela possui alguns problemas. Primeiro, para representar valores contínuos com alta precisão são necessários por vezes muitos genes como mostram Michalewicz (1996) e Herrera, M.Lozano e Verdegay (1998). Segundo, também esclarecido por Herrera, M.Lozano e Verdegay (1998), quando uma variável tem um número discreto de valores válidos que não é potência de dois, alguns códigos binários serão redundantes e poderão gerar soluções inválidas caso os genes extras não sejam adequadamente tratados. Terceiro, dois indivíduos com um único bit de diferença podem não estar próximos no espaço de soluções.

Já a codificação real permite diminuir o tamanho dos cromossomos e ainda assim conseguir uma maior precisão numérica. Essa representação também entrega um desempenho superior dos resultados para aplicações que necessitem de precisão. Segundo Balieiro et al. (2008), Golub (1996) e Janikow e Michalewicz (1991), a codificação real em comparação com a binária é mais rápida, produz valores melhores e tem um menor desvio padrão. Como os problemas utilizados nesta dissertação demandam precisão numérica com até 120 genes, no caso do despacho econômico dinâmico de energia elétrica, decidiu-se pela utilização da representação real.

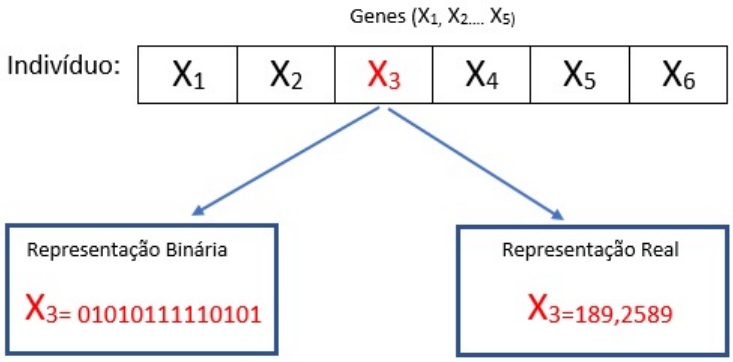

Figura 2 – Representação Cromossomial

2.2 Seleção

O primeiro mecanismo utilizado nos algoritmos genéticos é a seleção. Ele seleciona os indivíduos que poderão sofrer cruzamento e posteriormente mutação. O objetivo aqui é imitar a seleção natural, fazendo que indivíduos mais aptos tenham mais chances de passar seus genes para as próximas gerações.

Existem várias técnicas de seleção como roleta, ranking, dentre outras podem ser vistas nos trabalhos de Cortes (2004), Lehrer e Borges (2002) e Linden (2012). Neste estudo optou-se pelo método do torneio por ter apresentado melhores resultados que o da roleta. Neste método, inicialmente, são sorteados três indivíduos. Um torneio entre eles escolhe o mais apto. Vários torneios são feitos para escolher o grupo que formará a próxima geração através de cruzamentos e mutações (Figura 3).

2.3 Cruzamento

Os operadores de cruzamento utilizam os genes de dois ou mais pais para gerar descendentes. Seu propósito é permitir que os melhores indivíduos escolhidos na etapa de seleção troquem informações (genes) entre si, gerando assim descendentes possivelmente ainda mais aptos. O cruzamento ocorre depois da geração da população temporária por seleção. A ocorrência ou não de cruzamento será decidida através da utilização da probabilidade de cruzamento, ou seja, para cada indivíduo sorteia-se um número entre 0 e 1, se esse número for menor do que a probabilidade de cruzamento então esse indivíduo será cruzado. A forma de como o cruzamento ocorre pode variar. Ela pode, por exemplo, ser feita aos pares em sequencia, ou em ordem aleatória. Nesta dissertação, serão explorados quatro operadores de cruzamento na representação real: simples, aritmético uniforme, aritmético não uniforme e linear.

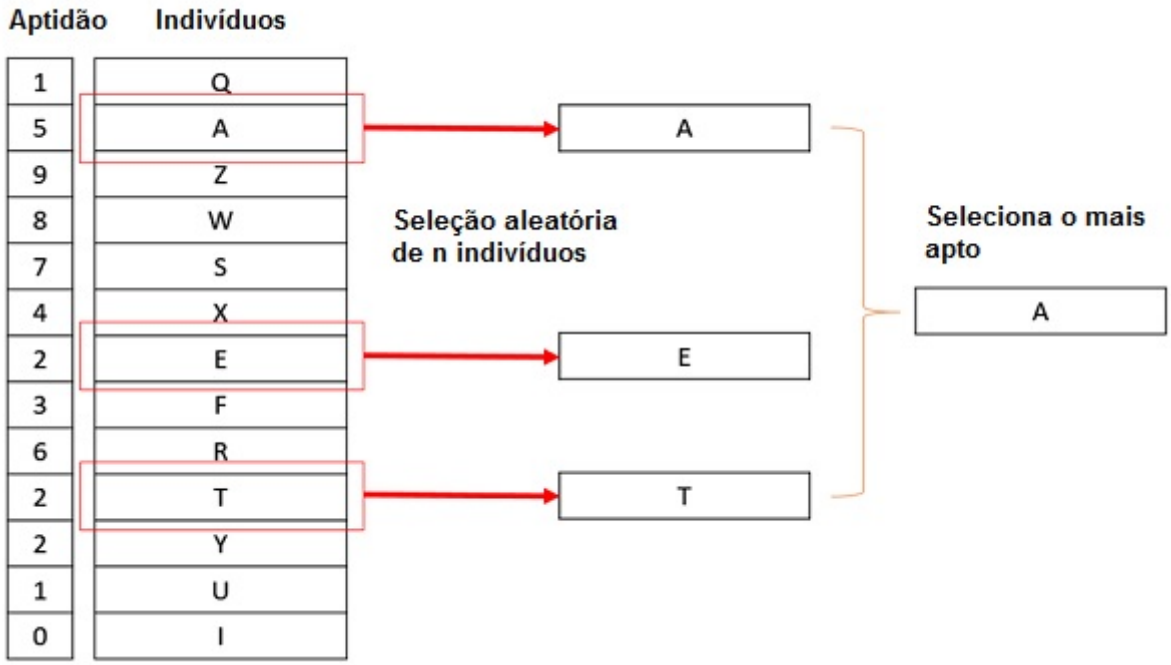

Figura 3 – Seleção por Torneio

Adaptado de: <http://www.w3ii.com/pt/genetic_algorithms/genetic_algorithms_quick_guide.html>

O cruzamento simples produz dois filhos a partir de dois indivíduos. Dados dois cromossomos (p^1 e p^2), e considerando que g_i^k é o gene i de p^k onde k individualiza o indivíduo. Os dois filhos (s_i^1 e s_i^2) serão gerados conforme as Equações 2.1 e 2.2 .

$$s_i^1 = \begin{cases} g_i^1 & \text{if } i \mod 2 = 0 \\ g_i^2 & \text{if } i \mod 2 = 1 \end{cases} \tag{2.1}$$

$$s_i^2 = \begin{cases} g_i^2 & \text{if } i \mod 2 = 0 \\ g_i^1 & \text{if } i \mod 2 = 1 \end{cases} \tag{2.2}$$

No cruzamento aritmético uniforme, os descendentes são gerados dentro da reta que liga cada gene dos dois pais. Dados dois cromossomos (p^1 e p^2), e considerando que g_i^k é o gene i de p^k, os dois filhos serão gerados obedecendo as Equações 2.3 e 2.4, onde r está no intervalo entre 0 e 1.

$$s_i^1 = r \times g_i^1 + (1 - r)g_i^2 \tag{2.3}$$

$$s_i^2 = (1 - r) \times g_i^1 + (r)g_i^2 \tag{2.4}$$

Tanto o cruzamento simples quanto o aritmético uniforme não geram indivíduos inválidos, ou seja, todos os genes estarão dentro do domínio do problema.A diferença do cruzamento aritmético uniforme e não uniforme está no r que varia de acordo com a geração, conforme a Equação 2.5, assim, há uma tendência de produzir indivíduos mais próximos do primeiro genitor no caso da Equação 2.3 e do segundo genitor na Equação 2.4.

$$r = \frac{GeracaoAtual}{NumeroTotaldeGeracoes} \tag{2.5}$$

O cruzamento linear gera 3 filhos e descarta o menos apto. Ela explora regiões além dos genes dos pais. O primeiro descendente é gerado como se o cruzamento aritmético com r igual a 0,5 fosse usado. Ele é a média dos genes do pais de acordo com a Equação 2.6. O segundo filho explora regiões acima do primeiro pai e o terceiro acima do segundo pai, como pode ser visto nas equações 2.7 e 2.8, o que eventualmente pode levar a regiões fora do domínio. Quando uma solução ultrapassa o domínio inferior ou superior normalmente se aumenta, caso tenha ultrapassado o limite inferior, ou se decresce o gene caso tenha passado o limite superior, para deixar seu valor dentro do domínio do problema.

$$s_i^1 = 0.5 * s_i^1 + 0.5 * s_i^2 \tag{2.6}$$

$$s_i^2 = 1.5 * s_i^1 - 0.5 * s_i^2 \tag{2.7}$$

$$s_i^3 = -0.5 * s_i^1 + 1.5 * s_i^2 \tag{2.8}$$

2.4 Mutação

Esses operadores são responsáveis pela tentativa de manter a diversidade da população, alterando um ou mais genes das novas proles. Eles foram concebidos para evitar a convergência prematura para um mínimo local. A decisão de aplicação desses operadores cabe à probabilidade de mutação que poderá ser aplicada sobre toda a população gerada pelo processo de cruzamento. Para cada gene da população gerada pelo cruzamento um número aleatório será gerado, se o mesmo for menor do que a probabilidade de mutação, esse gene será mutado. Nesta dissertação são estudados quatro tipos de mutação.

A mutação uniforme simplesmente troca um gene por outro sorteado dentro do domínio do problema. A mutação não uniforme (Equações 2.9 e 2.10) difere da primeira variando a magnitude da mudança. Quando maior for a geração, menor será a mudança, convergindo assim para valores pequenos a medida que a quantidade de gerações se aproxima do final. Para atingir esse objetivo, a equação usa t e T, na qual, t é a geração

atual e T o número total de gerações. Ainda nessa equação, b é um número definido pelo usuário que informa o grau de dependência do número de gerações, normalmente b é um valor entre 2 e 5. Nesta dissertação o grau de dependência do número de gerações foi definido como 2.

$$s_i^{'} = \begin{cases} s_i + \Delta(t, LS - s_i) & \text{if } \theta = 0 \\ s_i + \Delta(t, s_i - LI) & \text{if } \theta = 1 \end{cases} \tag{2.9}$$

$$\Delta(t, y) = y \times (1 - r^{(1-\frac{t}{T})^b}) \tag{2.10}$$

Nas referidas Equações 2.9 e 2.10, r é num número aleatório entre 0 e 1. LS e LI são respectivamente o limite superior e o limite inferior do gene sendo mudado s_i. Δ é a função que retorna um número entre $[0, y]$, onde y dependerá de θ, podendo ser $LS - s_i$ caso seja 0 ou $s_i - LI$ caso seja 1. Um sorteio definirá se θ recebe o valor 0 ou 1.

A terceira mutação estudada é a mutação creep como apresentado na Equação 2.11. A mudança no gene será a adição ou subtração de um valor pequeno a um dos genes. A decisão se a perturbação será adicionada ou subtraída depende de um sorteio que definirá se θ receberá o valor 0 ou 1.

$$s_i^1 = \begin{cases} s_i^1 + perturbacao & \text{if } \theta = 0 \\ s_i^1 - perturbacao & \text{if } i\theta = 1 \end{cases} \tag{2.11}$$

A última mutação estudada é a mutação apso evoluída. Proposta por Niknam e Golestaneh (2012), ela diminui a probabilidade que a melhoria das soluções pare ou fique mais devagar antes do ponto ótimo global. Este operador foi adaptado para esta dissertação e teve seus primeiros três passos da proposta original implementados aqui. Os outros passos levam em consideração características do enxame, como a inércia e a velocidade, que não são utilizados neste algoritmo. Os passos estão enumerados abaixo. P_j^k é o indivíduo atual onde j é a quantidade de genes e a k é a posição do mesmo dentro da população e P_j^k, t referencia o gene da posição t dentro do indivíduo.

1. Selecione cinco indivíduos da população (m_1, m_2, m_3, m_4 e m_5) onde $m_1 \neq m_2 \neq m_3 \neq m_4 \neq m_5$. Todos diferentes do indivíduo selecionado para mutação j.

2. O vetor de mutação ($P_{mut,j}^{k,t}$) é criado conforme a fórmula:

$$P_{mut,j}^{k,t} = P_{m1}^{k,t} + U(0,1) * (P_{m2}^{k,t} - P_{m3}^{k,t}) + U(0,1) * (P_{m4}^{k,t} - P_{m5}^{k,t}) \tag{2.12}$$

   onde t representa a posição do gene afetado.

3. Um novo indivíduo s é obtido:

$$P_s^{k,t} = \begin{cases} P_{mut,j}^{k,t}, & \text{Se } U_1(0,1) \leq U_2(0,1) \\ P_j^{k,t}, & \text{senão} \end{cases} \tag{2.13}$$

onde: $U_1(0,1)$ e $U_1(0,1)$ são números aleatórios entre 0 e 1.

2.5 Elitismo

O elitismo foi criado para garantir uma melhoria contínua da população. Com ele pode-se garantir que os indivíduos mais aptos de cada população sobrevivam para passar seus genes adiante. Essa alteração quase não altera o tempo de processamento e certifica que a nova geração terá, no pior caso, a mesma aptidão da anterior.

O elitismo é implementado comparando os melhores indivíduos da população atual com os melhores indivíduos da população nova. Se os indivíduos da população atual forem mais aptos, eles serão reinseridos na nova população, retirando da nova população os indivíduos menos aptos.

2.6 Trabalhando com Restrições

Uma função de otimização com restrições é comumente descrita por uma função como a da Equação 2.14 que possui restrições de igualdade (Equação 2.15) e de não-igualdade (Equação 2.16) onde $\overline{x}$ é o vetor de soluções, F é a região de soluções factíveis e S é o espaço de busca. O vetor de soluções $\overline{x}$ que satisfaz todas as restrições está no região F de soluções factíveis.

$$Minf(\overline{x}) \ , \ \overline{x} = (x_1, ..., x_n)^t \in F \subseteq S \subseteq R^n \tag{2.14}$$

sujeito a

$$h_j(\overline{x}) = 0 \ , \text{ onde } j = q+1, ..., m \tag{2.15}$$

$$g_i(\overline{x}) \leq 0 \ , \text{ onde } i = 1, ..., q \tag{2.16}$$

Existem muitos métodos para resolver essas funções de otimização multimodais com restrições. Segundo Yeniay (2005) esses métodos podem ser divididos em quatro categorias:

a) Métodos baseados em funções de penalidade conforme Fletcher (2013), e Xu et al. (2013) .

b) Métodos baseados na busca de soluções factíveis conforme Michalewicz e Nazhiyath (1995) e D.Powell e Skolnick (1993) .

c) Métodos baseados na preservação de soluções factíveis conforme Michalewicz e Janikow (1993) e Schoenauer e Michalewicz (1996).

d) Métodos híbridos conforme Belur (1997) e Hajela e Lee (1997).

Segundo o trabalho de Berhe (2012) a classificação dos métodos para tratar funções com restrições pode ser mais simples, focada apenas na forma explícita ou implícita que serão tratadas todas as restrições:

> "Todos os muitos métodos disponíveis para solucionar um problema de programação não linear com restrições podem ser classificados em duas categorias mais amplas: enfoque dos métodos diretos e indiretos. Nos métodos diretos as restrições são tratadas de uma maneira explícita enquanto na maioria dos métodos indiretos, o problema com restrições é resolvido como uma sequência de problemas de minimização sem restrições ou como um único problema de minimização sem restrições."

2.6.1 Método da Função de Penalidade

O método utilizado nesta dissertação para resolver as funções multimodais é o método da função de penalidade. O tipo de penalidade aplicada varia e serve como classificação deste método. O primeiro tipo é a aplicação da penalidade de morte do indivíduo que não obedece as restrições e fica fora da região factível.A desvantagem deste método aparece quando existem muitas restrições pois a obtenção apenas de indivíduos válidos pode ser uma tarefa dispendiosa e complexa. Por conta das restrições do problema de despacho econômico dinâmico de energia elétrica, a quantidade de tentativas para gerar um indivíduo nas primeiras gerações era extremamente numerosa e, aliada a um cálculo complexo, tornava os tempos de execução proibitivos.

O segundo tipo de penalidade é a função de penalidade estática. Nela, os parâmetros de penalidade não dependem do número da geração. Existem níveis de violação para cada restrição. As penalidades são aplicadas para cada nível de violação de cada restrição. A população é gerada usando tanto indivíduos dentro da região factível quanto fora. Este foi o tipo utilizado nesta dissertação. No tipo de penalidade de função dinâmica, a penalidade dependerá da geração em curso. Ele aumentará a penalidade a medida que as gerações forem se sucedendo.

A escolha do método de penalidade estática se deveu primeiramente a quantidade muito pequena de indivíduos válidos para um número muito grande de tentativas. Outro fator importante é que dois indivíduos com penalidades podem gerar filhos sem penalidades. A morte dos pais diminuiria o espaço de busca. Por fim, a própria definição do despacho econômico dinâmico levou a escolha da penalidade estática. Nesta função, a penalidade não depende da geração e sim da violação de restrições.

Esse método é implementado, para problemas de minimização, adicionando ou multiplicando a penalidade, caso o vetor de soluções encontrado não obedeça a alguma restrição, ou seja, não esteja na região de soluções factíveis. Esse aumento prejudica a aptidão das soluções, como mostra a Equação 2.17 .

$$Aptidao(\overline{x}) = \begin{cases} f(\overline{x}) & \text{if } \overline{x} \in F \\ f(\overline{x}) + p(\overline{x}) & \text{if } \overline{x} \notin F \end{cases} \tag{2.17}$$

onde $\overline{x}$ é o vetor de soluções candidatas e F é o conjunto de soluções factíveis.

2.7 Considerações Finais

Neste capítulo foram apresentados os algoritmos genéticos, sua representação, seleção, cruzamento, mutação e elitismo. O algoritmo desenvolvido nesta dissertação usa a representação real, a seleção por torneio, 4 operadores de cruzamento e 4 operadores de mutação descritos neste capítulo.

A Tabela 1 mostra um resumo dos operadores de cruzamento e mutação. A multiplicidade de operadores de cruzamento e mutação, mostrada nessa tabela, permite que o algoritmo escolha o operador mais adequado para cada problema. O algoritmo que executa essa escolha será tratado em detalhes no Capítulo 4.

Tabela 1 – Resumos dos Operadores de Cruzamento e Mutação Utilizados nos Algoritmos Genéticos

Operador	Resumo
Cruzamento Simples	Gera 2 indivíduos com metade dos genes do pai e metade dos genes da mãe
Cruzamento Arit. Uniforme	Produz 2 descendentes dentro da reta que liga cada gene dos dois pais. Não geram indivíduos inválidos. r entre 0 e 1.
Cruzamento Arit. Não Uniforme	Produz 2 descendentes dentro da reta que liga cada gene dos dois pais. Não geram indivíduos inválidos. r depende da geração atual para gerar alterações menores nas últimas gerações.
Cruzamento Linear	Gera 3 filhos e descarta o menos apto. O primeiro é gerado como se o cruzamento aritmético com r igual a 0,5 fosse usado. O segundo filho explora regiões acima do primeiro pai e o terceiro acima do segundo pai.
Mutação Uniforme	Troca um gene por outro sorteado dentro do domínio do problema
Mutação Não Uniforme	Difere da mutação uniforme variando a magnitude da mudança. Quando maior for a geração, menor será a mudança.
Mutação Creep	A mudança no gene será a adição ou subtração de um valor pequeno a um dos genes. A decisão se a perturbação será adicionada ou subtraída depende de um sorteio que definirá se θ receberá o valor 0 ou 1.
Mutação Apso Evoluída	Diminui a probabilidade que a melhoria das soluções pare ou fique mais devagar antes do ponto ótimo global.

3 Programação Paralela

Na programação sequencial o trabalho a ser executado é dividido em tarefas que são executas uma após a outra no processador. Na programação paralela podem ser utilizadas múltiplas linhas de execução simultâneas para resolver um problema. Para que isso seja possível, o programa é dividido em partes menores que executam ao mesmo tempo em diferentes unidades de processamento. Assim, a computação paralela pode melhorar a utilização dos recursos computacionais, permitindo que os processadores com vários núcleos sejam aproveitados por aplicações que demandam um poder maior de processamento. A relação entre a demanda por processamento, a evolução dos computadores e a programação paralela é bem sintetizada por Kirk e Wen-Mei (2016):

> "Microprocessadores baseados numa única unidade central de processamento (CPU), como os da família Intel Pentium e da família Opteron da AMD, levaram a incrementos rápidos de performance e redução de custos em aplicações computacionais nas duas últimas décadas. (...) Essa busca incansável pela melhoria da performance permitiu que as aplicações de software provessem mais funcionalidades, tivessem interfaces melhores e gerassem resultados mais úteis. Os usuários a seu turno demandaram ainda mais aperfeiçoamentos, assim que ficaram acostumados com essas melhorias, criando um ciclo positivo (virtuoso) para a indústria de computadores. Essa busca entretanto ficou mais devagar a partir de 2003 devido a problemas de consumo de energia e dissipação de calor que limitaram o aumento da frequencia de clock e do nível de atividades produtivas que podem ser feitas em cada ciclo de clock dentro de uma única CPU. Desde então todos os fabricantes de microprocessadores mudaram para modelos onde múltiplas unidades de processamento, referidas como núcleos de processador, são usadas em cada chip para incrementar o poder de processamento. Essa mudança teve um impacto tremendo na comunidade de desenvolvimento de software. "

Em seu livro, Fox, Williams e Messina (2014), descrevem uma série de aplicações alavancadas pela programação paralela como monitoramento em tempo real de satélites da NASA, exploração de petróleo, identificação de novos materiais com propriedades interessantes como supercondutores e citam especificamente a simulação de distribuição de energia e de gás para otimizar a produção e a resposta a falhas já na introdução e no capítulo 19. Essa aplicação é uma das funções de teste para o algoritmo criado nesta dissertação.

Nesse contexto, os computadores atuais favorecem os programas paralelos, pois são montados com múltiplos núcleos. No entanto, trabalhos como o de Geyer et al. (2001) , Chandra (2001) e Moldovan (2014) citam que sua implementação aumenta a complexidade do código em relação à programação sequencial. Este aumento de complexidade reflete diretamente nas horas gastas pelo desenvolvedores e, por fim, nos custos do desenvolvimento.

A decisão sobre a paralelização ou não do software deve ser bem fundamentada. Nesta dissertação, a programação paralela servirá para diminuir o tempo de processamento da aplicação, permitindo inclusive a resolução de problemas complexos em tempo adequado às características de cada problema. Um dos problemas resolvidos aqui, o despacho econômico dinâmico de energia elétrica, apresenta tempos de processamento na versão sequencial do algoritmo desenvolvido até 80 vezes maior que as funções de benchmark testadas.

Para melhor definir computação paralela, inicialmente é importante entender os vários tipos de computadores paralelos. A classificação de Flynn (1972), embora antiga, ainda abrange grande parte das arquiteturas paralelas disponíveis. Nesse sentido, ainda pode ser utilizada para uma melhor compreensão de como dados e instruções podem ser processadas ao mesmo tempo, ou seja, paralelizadas. O algoritmo que será desenvolvido nesta dissertação utilizará um tipo específico de computador paralelo, o tipo MIMD. Os computadores segundo Flynn são classificados em:

- SISD (Single Instruction Single Data). É o tipo mais antigo de computador, consegue processar apenas um fluxo de instrução e um fluxo de dados por vez. Este é um computador sequencial.
- SIMD (Single Instruction Multiple Data). Todas as unidades de processamento trabalham com a mesma instrução, mas com dados diferentes. Usados para problemas especializados com alto grau de regularidade.
- MISD (Multiple Instruction Single Data) . Cada processador opera de forma independente, usando um único fluxo de dados. São computadores paralelos, no entanto, poucos exemplos de computadores que utilizam esta arquitetura foram desenvolvidos. Como exemplo, é possível citar a execução de vários algoritmos tentando decifrar a criptografia de uma mensagem.
- MIMD (Multiple Instruction Multiple Data). Este computador tem vários processadores. Cada processador utiliza um fluxo diferente de dados. A execução pode ser síncrona, assíncrona, determinística ou não determinística.

3.1 Tipos de Compartilhamento de Memória

Uma importante classificação para a arquitetura do paralelismo é o compartilhamento de memória. Esse compartilhamento define como serão trocadas as informações entre os processadores.

Nos computadores paralelos com memória compartilhada todos os processadores operam de maneira independente, mas acessam uma memória comum com espaço de endereçamento global. É classificada em uniforme e não uniforme.

Na memória compartilhada uniforme, as formas e tempos de acesso às memórias são iguais entre os processadores. Quando uma posição de memória é atualizada por um processador, todos os outros processadores que acessarem aquela posição logo após já visualizarão as mudanças. Esse espaço de endereçamento global torna a programação mais fácil. O problema pode ser a escalabilidade, pois quanto mais processadores forem adicionados, maior será o tráfego de dados entre os processadores e a memória.

Na memória compartilhada não uniforme os processadores não têm tempo de acesso igual a todas as memórias. É construído ligando dois ou mais computadores de memória compartilhada uniforme. Essa ligação torna o acesso à memória mais lento. Na Figura 4 essa construção, feita a partir de 4 computadores, de memória compartilhada uniforme é visualizada.

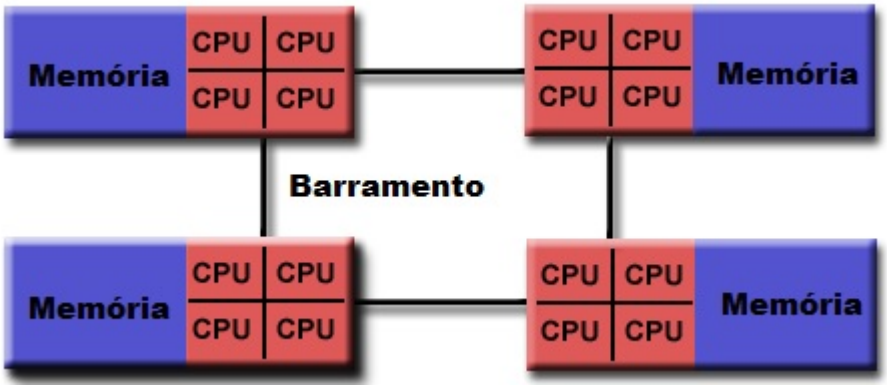

Figura 4 – Memória Compartilhada Não Uniforme

Nos computadores paralelos com memória distribuída (Figura 5), cada processador acessa endereços privativos de memória. Não existe um endereço global de memória que mapeie a memória de todos os processadores. Neste caso, o intercâmbio de informações é feito através de uma rede. Comumente, utiliza-se troca de mensagens para acessar a memória de outro computador. A vantagem do uso da memória distribuída é que ela é escalável e que como cada processador tem acesso a seu espaço privativo de memória, não precisa se preocupar com o acesso concorrente. As principais desvantagens são a inexistência de um mapeamento único de todas as memórias e a forma como se dará a comunicação entre os processadores.

A última classificação se refere a computadores paralelos de memória híbrida (compartilhada e distribuída). Neste caso temos vários computadores de memória compartilhada unidos pela rede. A arquitetura é similar a estrutura de computadores distribuídos, entretanto cada nó da Figura 6 é substituído por um computador de memória compartilhada. Atualmente, estes são os computadores mais rápidos. A grande escalabilidade é sua principal vantagem.

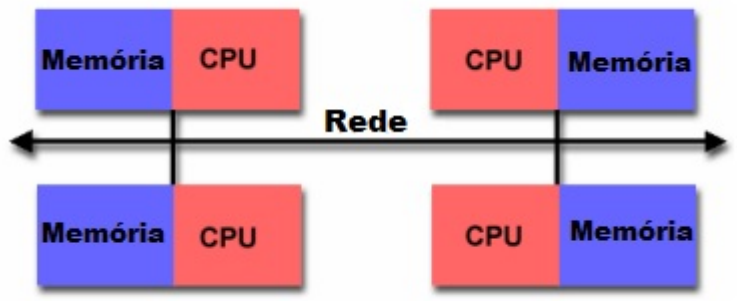

Figura 5 – Memória Distribuída

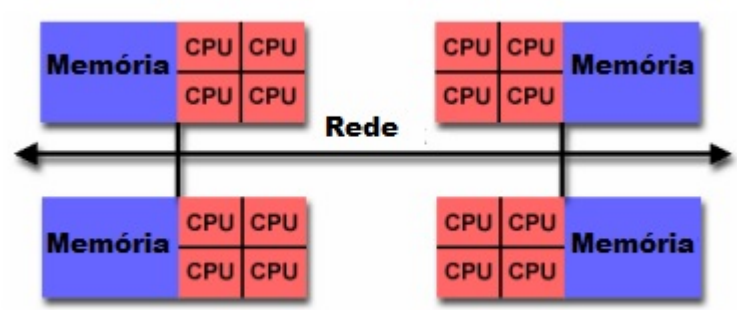

Figura 6 – Memória Híbrida

3.2 Índices de Desempenho

Segundo Navaux, Rose e Pilla (2011) dois índices demonstram o desempenho das aplicações paralelas. O primeiro é o fator de aceleração (speedup) que mostra quantas vezes a aplicação paralela é mais rápida que a aplicação sequencial. Se esse fator de aceleração for maior que um, a versão paralela reduziu o tempo de execução, caso contrário ela deixou a aplicação mais lenta. A Equação 3.1 mostra o cálculo do fator de aceleração (FA).

$$FA = \frac{TempoVersaoSequencial}{TempoVersaoParalela} \tag{3.1}$$

Esse fator depende da aplicação que está sendo paralelizada. No caso da aplicação desenvolvida nesta dissertação, cada geração do algoritmo genético, depende da geração anterior, o que limita a porção de código que poderá executar de forma concorrente.

O segundo índice de desempenho é a eficiência. Ela demonstra a taxa de utilização média das unidades de processamento utilizadas conforme a Equação 3.2.

$$Eficiência = \frac{FA}{NumeroDeProcessadores} \tag{3.2}$$

3.3 Limitações

Os índices de desempenho vistos trazem novas questões. A primeira é qual será o fator de aceleração esperado para um dado programa. A lei de Amdahl prevê que o fator de aceleração do programa (FA) é determinado pela fração do código (p) que irá executar em paralelo, como revela a Equação 3.3.

$$FA = \frac{1}{1-p} \tag{3.3}$$

De forma simples, paralelizando-se metade do código, o programa irá executar duas vezes mais rápido, como mostrado na Figura 7. Considerando-se o número de processadores n, p é a fração do código paralela e s é a fração serial, a fórmula fica mais abrangente como ilustrado na Equação 3.4. Com esta nova fórmula fica mais claro que existe uma limitação para o incremento da velocidade. Na Figura 8 é possível se notar que mesmo paralelizando 95 por cento do código e executando em 65536 processadores só seria atingido um fator de aceleração igual a 20. O incremento do número de processadores é capaz de melhorar o fator de aceleração só até certo ponto. Por exemplo, se tivermos 95% do código paralelizado, 5% serial e 1024 processadores, seguindo a fórmula teremos um speedup de 19,635. Caso dobrássemos o número de processadores o speedup será de 19,8161 e para 4096 será de 19,908 tendendo assim para a estabilidade mostrada na Figura 8.

$$FA = \frac{1}{\frac{p}{n} + s} \tag{3.4}$$

3.4 OpenMP

OpenMp (OpenMP (2013)) é um sigla que significa especificações abertas para multiprocessamento através de trabalho colaborativo da indústria de hardware e software, universidades e governo. É uma API de programação paralela que permite que os programadores apenas explicitem que regiões do código serão paralelizadas, sem se preocupar com detalhes de implementação como o sincronismo. Utiliza o modelo de memória compartilhada aqui descrito na Seção 3.1. Portável, serve para um conjunto amplo de compiladores, entre eles o compilador C que é utilizado nesta dissertação.

Uma diretriz do pré-processador sinaliza as partes do código que funcionarão em paralelo. Ela será responsável por, antes da execução desse trecho de código, criar as threads, que são fluxos únicos e sequenciais dentro de um programa. Cada programa possui uma ou mais threads. Cada thread recebe um número, ficando atribuído o zero para a thread principal.

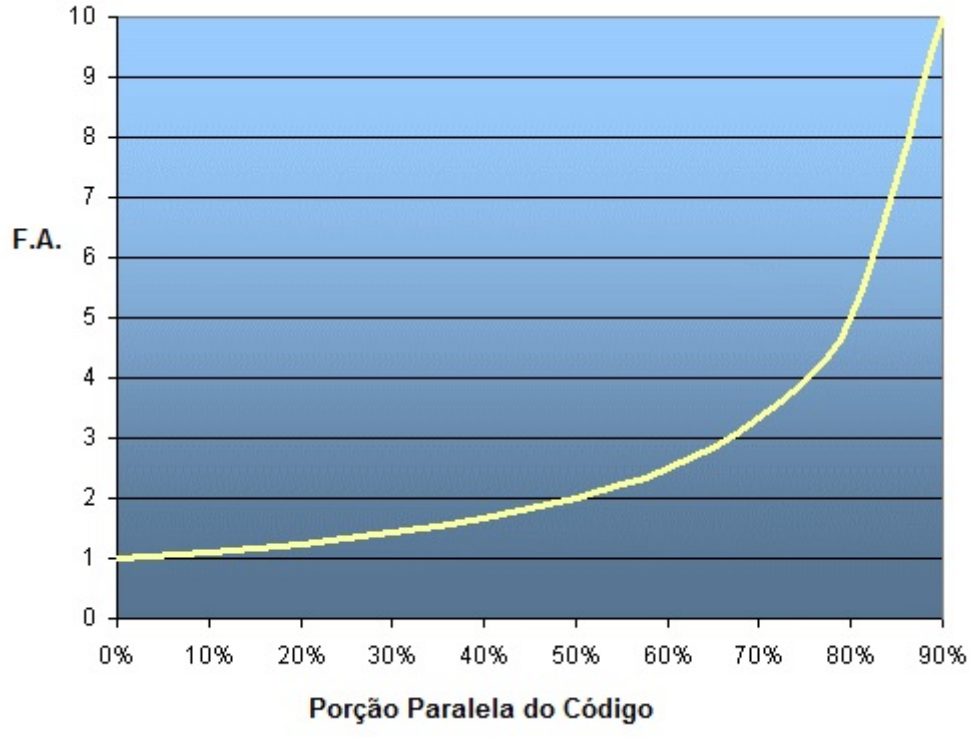

Figura 7 – Gráfico da Lei de Amdahl

Adaptado de:<https://goo.gl/Zv6B5M>

As variáveis utilizadas no programa, normalmente estão visíveis a todas as threads. Para evitar problemas de concorrência deve ser definida uma forma diversa de compartilhamento das mesmas.

Apesar da relativa facilidade de uso, Muddukrishna, P.Jonsson e Brorsson (2015) alerta sobre a complexidade da programação para um ganho efetivo de performance:

> "Outrossim, melhorar a performance do programa balanceando o paralelismo de tarefa coma hierarquização da utilização de memória é um processo iterativo. Modificadores de código e escalonamento alteram a composição baseada em tarefas. O recentemente descoberto paralelismo de tarefas tem de ser tediosamente compreendido de novo para cada iteração. O número de iterações gastas no processo de ajuste da performance varia dependendo da experiência do programador, mas múltiplas iterações são normalmente necessárias até mesmo para programadores experientes."

3.4.1 Funcionamento do OpenMP

Os programas que utilizam a API OpenMP começam de forma sequencial com uma thread mestre até encontrar a primeira região paralela. Nessa região, os comandos são executados por diversas threads paralelas. Quando as threads paralelas finalizarem a

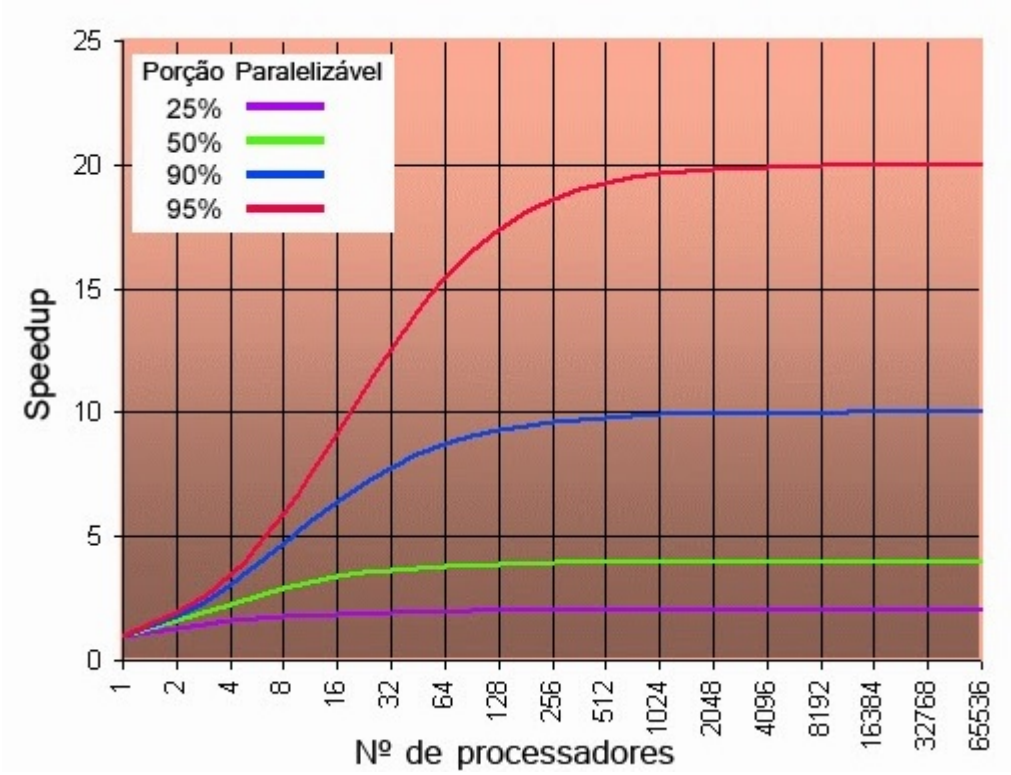

Figura 8 – Gráfico da Lei de Amdahl com Porção do Código paralelizado

Fonte: <https://goo.gl/mKigk5>

execução dos comandos, ocorre a sincronização, permanecendo apenas a thread mestre.

A OpenMP é formada por bibliotecas de funções, variáveis de ambiente e diretivas de compilação. Bibliotecas de funções são conjuntos de funções escritas por um programador e adicionados por meio de uma linha escrita no código-fonte. As variáveis de ambiente são variáveis que podem afetar a forma de comportamento do processo. As diretivas de compilação são anotações que assinalam pontos e formas de paralelização. Todas as diretivas em C para o OpenMp começam com "#pragma omp". A diretiva parallel é a construção mais importante do OpenMP (Figura 9) . Ela identifica o bloco de código que será executado por várias threads.

A cláusula if define uma condição para que o grupo de threads seja criado. Caso a expressão lógica seja falsa, apenas a thread mestre executará o trecho de código. Caso seja verdadeira, várias threads serão criadas.

Na claúsula private, cada variável da lista será copiada em cada thread. Ela não é inicializada e seu valor não poderá ser usado fora da região paralela. A claúsula firstprivate funciona de modo parecido exceto que a inicialização será feita pelo valor original, ou seja, aquele antes de iniciar a região paralela.

A cláusula shared descreve as variáveis que serão compartilhadas, ou seja, visíveis

```
#pragma omp parallel cláusula (numThreads, tid)
      [if (expressão lógica)]
      [private (lista de variáveis)]
      [shared (lista de variáveis)]
      [default (shared | none)]
      [firstprivate (lista de variáveis)]
      [reduction (operador: lista de variáveis)]
      [copyin (lista de variáveis)]
      { .... Bloco de código; }
```

Figura 9 – Diretiva Parallel

para todas as threads simultaneamente. Por padrão todas as variáveis dentro da região paralela são compartilhadas, exceto as controladoras de laço. A claúsula default estabelece como será o compartilhamento em toda a região paralela. O padrão é compartilhado. A opção none força a declaração de cada variável da região paralela.

Na claúsula reduction, cada variável tem uma cópia por thread, mas os valores das cópias são resumidos numa variável global compartilhada. Ela é útil se uma variável depender do valor da iteração anterior. Ou seja, alguns passos são paralelizados, mas os threads reúnem-se e esperam antes de fazer o update das variáveis. Esse comportamento é requerido ao paralelizar funções de integração numérica e equações diferenciadas que .

A diretiva parallel possui três tipos de contruções, que formam outras diretivas: For, Sections e Single. A diretiva for (Figura 10) determina que as iterações de um laço sejam executadas em paralelo por um grupo de threads. A região paralela deve ser identificada antes.

As claúsulas já explicadas possuem a mesma função. A claúsula SCHEDULE informa como as iterações do laço serão divididas por entre as threads. Ela possui quatro opções:

a) STATIC n - O laço será dividido em n partes e distribuído entre as threads. Se o parâmetro n não for definido, o compilador dividirá o laço em partes iguais.

b) DYNAMIC n- O laço é distribuído dinamicamente para cada thread disponível. Cada thread recebe n iterações. Quando uma thread finaliza o conjunto n de iterações, outro é atribuído para processamento. Por padrão o valor de n é 1.

c) GUIDED – O número de iterações para cada thread irá variar, começando com um valor grande e sendo reduzido exponencialmente.

d) RUNTIME - A decisão sobre a divisão é tomada durante a execução do programa,

```
#pragma omp for cláusula
 schedule (tipo [,tamanho])
 ordered
 private (lista de variáveis)
 firstprivate (lista de variáveis)
 lastprivate (lista de variáveis)
 shared (lista de variáveis)
 reduction (operador: lista de variáveis)
 nowait
 Laço For
```

Figura 10 – Diretiva For

com a definição da variável de ambiente OMP SCHEDULE.

Sobre essas claúsulas Kalogerakos M. Gourma (2014) esclarece:

> "O construtor de compartilhamento de trabalho padrão fornecido pelo OpenMp pode não ser bom o suficiente na maioria dos casos ao distribuir igualmente o trabalho entre as threads disponíveis. De fato, o modo de distribuição padrão é pegar o número de iterações e dividir da melhor maneira entre todas as threads. Isso pode criar intâncias onde algumas threads estarão muito ocupadas enquanto outras estarão inativas. OpenMp no entanto, fornece algumas opções adicionais para o seu construtor que podem alterar este comportamento. O exemplo de distribuição citado supõe um STATIC SCHEDULE. Existe outra opção na claúsula SCHEDULE construct que pode ajudar neste caso: DYNAMIC. A opção DYNAMIC faz o compartilhamento do trabalho ser feito em tempo de execução, enquanto a opção STATIC o faz durante a compilação."

A claúsula nowait permite que as threads não sejam sincronizadas no final do laço, continuando a execução para o próximo comando.

A diretiva sections (Figura 11) divide o trabalho em seções independentes. Cada seção será executada por uma thread. Implementa o paralelismo funcional. No final dessa diretiva existe um ponto de sincronização implícita, a menos que, seja incluído o atributo nowait.

A diretiva single (Figura 12) estipula que a execução do código deve ser feita somente por uma thread. As threads do grupo que não executam a diretiva SINGLE, esperam o fim do processamento dela, a menos que se utilize o atributo nowait. A diretiva barrier cria um ponto de sincronização. A thread que chega nesse ponto para de executar até que todas as outras threads tenham chegado nessa diretiva.

```
#pragma omp sections
private (lista de variáveis)
firstprivate (lista de variáveis)
lastprivate (lista de variáveis)
reduction (operador: lista de variáveis)
nowait
 {
 #pragma omp section
      bloco de comandos;
 #pragma omp section
      bloco de comandos;
 }
```

Figura 11 – Diretiva Sections

```
#pragma omp single
private (lista de variáveis)
firstprivate (lista de variáveis)
nowait
 bloco de comandos;
```

Figura 12 – Diretiva Single

3.5 Considerações Finais

O aumento do número de núcleos dos processadores e a demanda por aplicações mais rápidas e com mais recursos tornaram a paralelização da execução dos programas desenvolvidos mais importante. Essa importância só tem crescido a medida que a indústria de processadores lança novos produtos, pois por conta de limitações técnicas, a frequência de clock teve um aumento limitado, mas a quantidade de núcleos de processamento segue sendo incrementada.

A programação paralela tem uma papel relevante no algoritmo desenvolvido nesta dissertação. O processamento de uma das aplicações que servem de teste para o algoritmo genético misto desenvolvido, o despacho econômico dinâmico de energia elétrica, consome um tempo de processamento elevado, que pode tornar inviável sua aplicação prática. Conforme explicado neste capítulo, para paralelizar o algoritmo foi escolhida a interface de programação de aplicativos chamada OpenMp. As diretivas para a utilização do OpenMp

não são muito complexas e conseguem dar algum retorno de maneira rápida. No entanto, para atingir fatores de aceleração melhores sem perda de qualidade, uma boa experiência no uso dessa interface de programação é requerida.

4 Algoritmo Genético Autoajustável Misto Paralelo

A premissa dos algoritmos autoajustáveis é a alteração automática dos parâmetros de funcionamento do algoritmo de acordo com o problema apresentado e com a geração para assegurar seu melhor desempenho. Funcionam de forma parecida aos algoritmos genéticos clássicos. A diferença está na escolha dos parâmetros. Enquanto nos algoritmos genéticos tradicionais ela é feita uma vez antes da execução do algoritmo, nos autoadaptáveis ela é feita durante a execução do algoritmo, podendo se adequar inclusive para as diferentes fases da execução. Nos algoritmos autoadaptáveis os parâmetros a serem ajustados são codificados nos genes dos indivíduos, sendo os mais comuns a probabilidade de cruzamento e a de mutação.

A principal diferença deste para os algoritmos adaptáveis ou ajustáveis é a codificação dos parâmetros de ajuste do algoritmo, segundo Nieberg e Beyer (2007, pp 47-55). Enquanto nos algoritmos ajustáveis ela ocorre fora do genoma do indivíduo, nos autoajustáveis ela acontece dentro, ou seja fica codificado nos genes do cromossomo. Essa codificação interna irá ter uma influência decisiva nos parâmetros.

Além disso, nos algoritmos ajustáveis, como não há a codificação interna, todos os indivíduos utilizam um único conjunto de parâmetros que varia após cada cruzamento e mutação. Nos algoritmos autoajustáveis, cada indivíduo tem codificado em si os parâmetros que levaram a sua geração. Cada operação, como cruzamento e mutação, é feita com os parâmetros retirados da população selecionada de indivíduos. Dessa forma, os cromossomos mais aptos terão uma influência maior na próxima população, pois pelo processo da seleção por torneio tendem a participar mais de cruzamentos e mutações.

4.1 Algoritmo Genético Ajustável Paralelo

O desenvolvimento do algoritmo autoajustável misto paralelo iniciou pela versão ajustável e de thread única. Deste estudo foi produzido o artigo ”A Stochastic Adaptive Genetic Algorithm for Solving Unconstrained Multimodal Numerical Problems” no IEEE EAIS. A paralelização do algoritmo foi desenvolvida e gerou um segundo artigo: ”A Parallel Adaptive Multithread Genetic Algorithm for Unconstrained Multimodal Numerical Optimization” aceito para o SBAI.

A Figura 13 mostra o algoritmo ajustável desenvolvido. Ele inicia com a geração da população inicial. Em seguida para cada nova geração, ele formará uma nova população com os pais escolhidos por torneio a partir da geração anterior. A Seção 2.2 descreveu o

funcionamento da seleção por torneio. Os próximos passos escolhem os operadores e as taxas de cruzamento e mutação. Uma função usando o método ajustável decide qual o operador de cruzamento será utilizado. Nesse algoritmo tanto os operadores quanto as taxas utilizam o método ajustável. A forma como essa decisão é tomada é explicada na Seção 4.2. O próximo passo é aplicar o cruzamento nesses pais selecionados, de acordo com a taxa de cruzamento (explicado em detalhes na Seção 2.3) e aplicar a mutação usando sua própria taxa (Seção 2.4). Após o processamento de todos os indivíduos, o elitismo é então aplicado como detalhado na Seção 2.5.

A paralelização foi feita usando as diretivas **parallel** e **for**, explicadas na Seção 3.4. Essa paralelização, no entanto, havia deixado uma parte muito executada do código com thread única, a seleção por torneio. Essa exclusão gerou um impacto nos resultados e o speedup atingido foi relativamente baixo para a quantidade de threads utilizadas.

Algoritmo 2 Algoritmo Ajustável Paralelo

```
procedure AJUSTE
    Pop ← GerarPopulacaoInicial();
    para i←1 ; até i< NumeroDeGeracoes faça
        para p←1 ; até TamanhoPopulacao faça
            NewPop ← SeleçãoPorTorneio(Pop);
        fim para
        OMP Parallel Directive
        OMP For Directive
        para p←1 ; até TamanhoPopulacao faça
            tecnicaCruzamento ← decideTecnicaCruzamento(NovaPopulacao);
            tecnicaMutacao ← decideTecnicaMutacao(NovaPopulacao);
            taxaCruzamento ← decideTaxaCruzamento(NovaPopulacao);
            taxaMutacao ← decideTaxaMutacao(NovaPopulacao);
            aplicaCruzamento(NovaPopulacao, tecnicaCruzamento);
            aplicaMutacao(NovaPopulacao,tecnicaMutacao);
        fim para
        End OMP Parallel For Directive
        Elitismo();
        Pop ← NewPop;
    fim para
fim procedure
```

Figura 13 – Algoritmo Genético Ajustável Paralelo

4.2 Algoritmo Genético Autoajustável Misto

Esta dissertação utilizou os dois métodos (autoajustável e ajustável) para tirar o melhor proveito de cada um, sendo por isso denominado de algoritmo autoajustável misto.

Os operadores de cruzamento e mutação são definidos de maneira ajustável e as taxas utilizam o autoajuste.

Na parte adaptável, a decisão dos tipos de operadores de cruzamento e mutação que irão gerar os novos filhos dependerá da probabilidade de execução deles. Quanto maior o percentual de um tipo de operador, maior será a probabilidade dele ser usado na geração de novos indivíduos. Os operadores iniciam a execução com uma distribuição uniforme de probabilidades, ou seja, cada operador de cruzamento n inicialmente tem a mesma possibilidade de execução (25% cada um) calculada a partir de um valor inicial (V_n) que recebe (25.0). Cada vez que um operador gerar um filho melhor que o indivíduo mais apto da geração passada, ele terá seu valor aumentado. Esse incremento será menor nas primeiras gerações, por ser mais fácil gerar indivíduos melhores e maior nas últimas gerações, de acordo com a Equação 4.1.

$$incremento = \frac{geraçãoAtual}{quantidadeGerações} \tag{4.1}$$

Após o aumento, a taxa de cada operador n (P_n) será calculada conforme a Equação 4.2. O mesmo processo ocorrerá para os operadores de mutação.

As taxas de mutação e cruzamento são autoajustáveis, ou seja estão codificadas dentro do genoma do indivíduo. A taxa de cruzamento dos filhos será a média das taxas de cruzamento que estão no genoma dos pais. A escolha pelo autoajuste das taxas se deveu à maior velocidade que uma taxa ideal é encontrada. No caso dos operadores essa mesma velocidade poderia diminuir de maneira precoce a utilização de um operador, motivo pelo qual foi escolhido o método ajustável.

O algoritmo autoajustável misto está representado na Figura 14 e difere do algoritmo representado na Figura 13 no processo de autoajuste das taxas. Ele também inicia com a geração da população inicial. Em seguida para cada nova geração, ele formará uma nova população com os pais escolhidos através da seleção por torneio a partir da geração anterior. Essa seleção foi explicada na Seção 2.2. Os próximos passos escolhem os operadores e as taxas de cruzamento e mutação para em seguida aplicá-los. O funcionamento deles está detalhado nas Seções 2.3 e 2.4. Após o processamento de todos os indivíduos, o elitismo é então aplicado como descrito na Seção 2.5.

$$P_n = \frac{V_n}{\sum_{o=1}^{4} V_o} \tag{4.2}$$

Algoritmo 3 Algoritmo Autoajustável Misto

```
procedure AutoAjuste
    Populacao ← geraPopulacaoInicial();
    para i←1 ; até NumeroDeGeracoes faça
        para p←1 ; até #TamanhoPopulacao faça
            NovaPopulacao ← SelecaoPorTorneio(Populacao);
            tecnicaCruzamento ← decideTecnicaCruzamento(NovaPopulacao);
            tecnicaMutacao ← decideTecnicaMutacao(NovaPopulacao);
            taxaCruzamento ← decideTaxaCruzamento(NovaPopulacao);
            taxaMutacao ← decideTaxaMutacao(NovaPopulacao);
            aplicaCruzamento(NovaPopulacao, tecnicaCruzamento);
            aplicaMutacao(NovaPopulacao,tecnicaMutacao);
        fim para
        Elitismo(NovaPopulacao, Populacao);
        Populacao ← NovaPopulacao;
    fim para
fim procedure
```

Figura 14 – Algoritmo Genético Autoajustável Misto

4.3 Algoritmo Genético Autoajustável Misto Paralelo

A última tarefa do desenvolvimento do algoritmo genético autoajustável misto foi torná-lo paralelo utilizando o OpenMp, explicado na Seção 3.4. Segundo a lei de Amdahl quanto maior a porção de código paralelizada, maior é o fator de aceleração. O laço mais externo que cuida das gerações seria uma boa opção, pois abrange a maior parte do código. No entanto, para computar a próxima geração é preciso que a geração atual esteja toda computada. Se várias gerações nascessem dependendo de uma única anterior, o desempenho pioraria bastante. Já o segundo laço, engloba a quase totalidade do código e pode ser paralelizado sem perda de qualidade. A Figura 15 mostra onde as diretivas OpenMp foram colocadas.

Para o código funcionar foi necessário utilizar a biblioteca $\#include < omp.h >$. O número de threads foi setado utilizando a instrução $num\_threads(number\_of\_threads)$ na diretiva $\#pragma\ omp\ parallel$. As diretivas utilizadas foram a $\#pragma\ omp\ parallel$ e a $\#pragma\ omp\ for$ que instruem o compilador a executar em paralelo as instruções do loop. A Figura 16 mostra o trecho da função principal onde essas diretivas são colocadas. Essas diretivas são explicadas em detalhes na Seção 3.4.

Algoritmo 4 Algoritmo Autoajustável Paralelo

```
procedure AutoAjuste
    Populacao ← geraPopulacaoInicial();
    para i←1 , até NumeroDeGeracoes faça
        Diretiva OMP Parallel
        Diretiva OMP For
        para p←1 , até #TamanhoPopulacao faça
            NovaPopulacao ← SelecaoPorTorneio(Populacao);
            tecnicaCruzamento ← decideTecnicaCruzamento(NovaPopulacao);
            tecnicaMutacao ← decideTecnicaMutacao(NovaPopulacao);
            taxaCruzamento ← decideTaxaCruzamento(NovaPopulacao);
            taxaMutacao ← decideTaxaMutacao(NovaPopulacao);
           aplicaCruzamento(NovaPopulacao, tecnicaCruzamento);
           aplicaMutacao(NovaPopulacao,tecnicaMutacao);
        fim para
        OMP Barrier
        Elitismo(NovaPopulacao, Populacao);
        Populacao ← NovaPopulacao;
    fim para
fim procedure
```

Figura 15 – Algoritmo Genético Autoajustável Paralelo

```
printf("Algoritmos Geneticos\n... Comparação... auto ajuste");
gravaLog("\nGerando população inicial...",0);
gerar_populacao_inicial(populacao,liminf,limsup);
imprimePopulacao(populacao);
gravaLog("\nPopulacao inicial Gerada...\n",0);
melhorGeracao = populacao[0];
for (i=1; i < gqtdPopulacao; i++){
    if (melhorGeracao.resFunObj > populacao[i].resFunObj)
        melhorGeracao = populacao[i];
}

menorFitGeracao[0] = melhorGeracao.resFunObj;
tipo_individuo pai, mae, filho1, filho2, i1,i2,i3,i4,i5;

for (i=1;i<=GERACOES-1;i++) {

    gravaLogInt("\nCriando nova geracao... %d", i);
    #pragma omp parallel  num_threads(8) firstprivate(executouCross, executouMutacao,
    {

    #pragma omp for
    /*Utilizo as tecnicas para geracao de filhos aqui (crossover)*/
    for (contador = 0; contador<= gqtdPopulacao-1;contador=contador+2){
        pai = selecao_por_torneio(populacao, gqtdPopulacao);
        mae = selecao_por_torneio(populacao, gqtdPopulacao);
```

Figura 16 – Trecho de Código da Função Principal

A Figura 17 mostra o trecho do código fonte do programa desenvolvido que são escolhidos o tipo e a taxa de cruzamento e mutação. As taxas que são autoajustáveis

dependem das taxas gravadas no genoma dos pais. As técnicas dependem de uma probabilidade gravada em variáveis globais. Cada variável recebe o valor inicial de 25.0. O ajuste automático aumenta a probabilidade de execução das técnicas que gerarem filhos melhores que o indivíduo mais apto da geração passada. Para os modos sem ajuste automático que utilizam apenas um tipo de mutação e um de cruzamento (Tabela 3) apenas uma das variáveis $crossptip1, crossptip2, crossptip3, crossptip4$ recebe o valor 100 e as outras zero. O mesmo acontece com as variáveis $mutptip1, mutptip2, mutptip3, mutptip4$.

```
sorteio = gera_numero_inteiro(0,1000);
sorteio1 = gera_numero_inteiro(0,1000);

if (selfAdaptive == 1){
    tempTecCross = decideTecCrossMut(crossptip1, crossptip2, crossptip3,crossptip4);
    tempTecMut = decideTecCrossMut(mutptip1,mutptip2,mutptip3,mutptip4);
    tempTaxCross = decideTaxaCrossMutacao(pai.taxCrossover, mae.taxCrossover);
    tempTaxMut= decideTaxaCrossMutacao(pai.taxMutacao, mae.taxMutacao);

} else {
    if (modoExecucao == 2){
        tempTecCross = gera_numero_inteiro(1,4);
        tempTecMut = gera_numero_inteiro(1,4);
    }else   {
        tempTecCross = decideTecCrossMut(crossptip1, crossptip2, crossptip3,crossptip4);
        tempTecMut = decideTecCrossMut(mutptip1,mutptip2,mutptip3,mutptip4);
    }
    tempTaxCross = 75;
    tempTaxMut = 65;
}
```

Figura 17 – Trecho de Código que mostra a escolha dos operadores e taxas de cruzamento e mutação

```
if (selfAdaptive == 1){
    if (executouCross==1){
        if (filho1.resFunObj < menorFitGeracao[i-1]) {
            incrementaTecCrossMut(tempTecCross,i, &crossptip1, &crossptip2, &crossptip3, &crossptip4);
        }
        if (filho2.resFunObj < menorFitGeracao[i-1]) {
            incrementaTecCrossMut(tempTecCross,i, &crossptip1, &crossptip2, &crossptip3, &crossptip4);
         }
    }
}
```

Figura 18 – Trecho de Código com o incremento da técnica de cruzamento

Após a execução do cruzamento a Figura 18 mostra como acontece o incremento da técnica de cruzamento caso ela gere filhos melhores que o indivíduo mais apto da geração anterior. Após a execução do laço de indivíduos, a claúsula *#pragma omp barrier* força a sincronização das threads. A Figura 19 mostra o trecho de código onde essa cláusula foi posicionada. O elitismo acontece em seguida. Se o melhor indivíduo da nova população for pior que o melhor indivíduo da geração anterior, ele será recolocado na nova geração.

```
#pragma omp barrier
gravaLog("************* \n\n Geracao %lf \n ********",(double)i);
// Elitismo
mergeSort(populacao,gqtdPopulacao);
mergeSort(novapopulacao,gqtdPopulacao);
if (populacao[0].resFunObj <= novapopulacao[0].resFunObj) {
    novapopulacao[gqtdPopulacao-1] = populacao[0];
    gravaLogInt("\n\n Não Houve melhoria. ",0);
    menorFitGeracao[i] = populacao[0].resFunObj;
} else {
    gravaLogInt("\n\n Houve melhoria. ",0);
    menorFitGeracao[i] = novapopulacao[0].resFunObj;
}

for (ig = 0; ig < gqtdPopulacao; ig++)
    populacao[ig] = novapopulacao[ig];
```

Figura 19 – Trecho de Código com a Cláusula Barrier

4.4 Considerações Finais

Neste capítulo foi descrito o algoritmo genético autoajustável misto paralelo criado. Os conceitos de algoritmos genéticos usados aqui foram mostrados no Capítulo 2, incluindo os operadores e taxas de cruzamento e mutação. A paralelização do algoritmo foi feita utilizando a interface de programação de aplicativos OpenMp, apresentada na Seção 3.4.

O autoajuste dos parâmetros de funcionamento do algoritmo genético é essencial para a execução eficiente do mesmo. O algoritmo genético autoajustável misto utilizou os métodos autoajustável e ajustável. Com o método autoajustável foram escolhidos os operadores de cruzamento e mutação que entregassem os melhores resultados e com o método ajustável foram definidos as taxas de cruzamento e mutação mais adequadas. A escolha pelo autoajuste das taxas se deveu à sua característica de encontrar uma taxa ideal com maior velocidade. Essa velocidade, desejada no caso das taxas, poderia diminuir de maneira precoce a utilização de um operador, motivo pelo qual foi escolhido o método ajustável.

5 Experimentos

5.1 Configurações

O algoritmo genético autoajustável foi desenvolvido utilizando a linguagem de programação C. O compilador utilizado foi o Intel. Esse compilador foi escolhido por melhorar a utilização e consequente desempenho do processador Core i7 e possuir um conjunto de ferramentas de desenvolvimento e testes chamada Parallel Studio. A API do OpenMP, utilizada para criar o paralelismo, está disponível no site oficial do OpenMP <www.openmp.org>.

Todos os testes foram executados no mesmo computador Core i7 2.8GHZ, 8Gb de Ram, Windows 7 de 64 bits e 1TB de disco rígido. O processador tem 4 núcleos e suporta 8 threads simultâneas.

Para testar a qualidade das respostas produzidas pelos algoritmos autoajustáveis paralelos serão construídas duas linhas principais de testes. A primeira linha irá comparar o algoritmo genético autoajustável com a versão sem o autoajuste, combinando todas as probabilidades de cruzamento e mutação. A segunda linha de testes irá medir o fator de aceleração do programa paralelizado com a versão sequencial.

Cada teste será executado 50 vezes com 600 e 2000 gerações. De acordo com o Teorema do Limite Central 30 testes tendem a criar uma distribuição normal. Os resultados serão comparados usando o teste estatístico ANOVA para validar as hipóteses e o teste de Tukey para atestar se as diferenças são ou não significativas.

Entre os ajustes da aplicação, o tamanho da população tem influência direta na execução do software. Uma população pequena demais não permite uma variedade genética ampla, limitando o espaço de buscas. A solução ótima neste caso pode não ser encontrada. Por outro lado, uma população muito grande aumenta o tempo necessário para a busca, podendo inclusive aproximar o algoritmo de uma busca exaustiva. O tamanho da população desta dissertação foi de 25 indivíduos, a mesma quantidade utilizada no trabalho de Niknam e Golestaneh (2012) que serviu de comparação. Para as comparações de tempo de execução a população foi aumentada para 200 indivíduos para que o tempo de execução pudesse ser medido melhor.

5.2 Simulação Computacional

Para testar o algoritmo desenvolvido foram utilizadas funções com e sem restrições. As funções de benchmark mostradas na subseção 5.2.1 não possuem restrições, além do

domínio de cada variável. A função do Despacho Econômico dinâmico de energia, detalhado na subseção 5.2.2, é um problema numérico dinâmico com restrições.

5.2.1 Funções de Benchmark

As funções de benchmark utilizadas nesta dissertação são multimodais, ou seja, elas têm como objetivo buscar o ponto de mínimo global entre os vários pontos de mínimo local existentes.

Como esclarece Cortes e Silva (2010) , essas funções têm duas classificações importantes: separabilidade e multimodalidade. É possível afirmar que uma função é multimodal, quando tem dois ou mais pontos ótimos locais. O ótimo global é o ponto de mínimo (quando o objetivo é a minimização da função) ou o de máximo (quando o objetivo é a maximização). Quanto a separabilidade, se uma função for classificada como separável, então ele pode ser reescrita como a soma de p funções de uma só variável.

Funções não separáveis são mais difíceis de otimizar pois para procurar a solução ótima no vetor de soluções depender-se-á de duas ou mais variáveis. Nas funções multimodais esta procura fica ainda mais difícil pois existem pontos ótimos locais que podem fazer a solução convergir prematuramente para um ponto diferente do ponto ótimo global. As seis funções de benchmark utilizadas nesta dissertação são apresentadas na Tabela 2.

Tabela 2 – Funções de Benchmark

Código	Nome	Fórmula	Domínio	Mínimo
SCW	Schwefel	$f(x) = 418.9829n - \sum_{i=1}^{n} x_i * \sin(\sqrt{(\lvert x_i \rvert)})$	[-500,500]	0
RAS	Rastringin	$f(x) = 10n + \sum_{i=1}^{n}[x_i^2 - 10\cos(2\pi x_i)]$	[-5.12,5.12]	0
ROS	Rosenbrock	$f(x) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_{i-1})^2]$	[-5,10]	0
GRI	Griewank	$f(x) = \sum_{i=1}^{n}(\frac{x(i)^2}{4000}) - \prod_{i=1}^{n}(\frac{x(i)}{\sqrt{(i)}}) + 1;$	[-600,600]	0
SROS	Shifted Rosenbrock	$f(x) = \sum_{i=1}^{n-1}[100(z_{i+1} - z_i^2)^2 + (z_{i-1})^2] + fbias_6, z = x - o + 1$	[-100,100]	o
SRAS	Shifted Rastringin	$f(x) = 10n + \sum_{i=1}^{n}[z_i^2 - 10\cos(2\pi z_i)] + fbias_9, z = x - o$	[-5.0,5.0]	o

A função Schewefel (Figura 20) é uma função complexa com muitos picos e vales. Multimodal e separável, o segundo ponto de mínimo é distante do primeiro, levando a muitos algoritmos de busca se perderem.

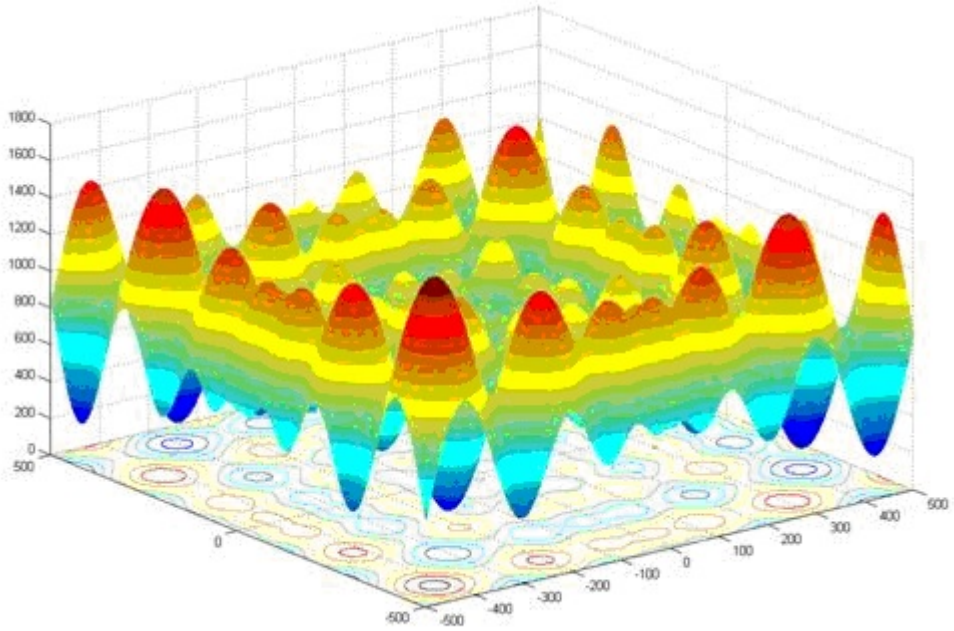

Figura 20 – Função Schewefel

Fonte: <http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page2530.htm>

A função Rastringin (Figura 21) é não convexa, multimodal e separável, muito usada para otimização de algoritmos. Encontrar o mínimo desta função é difícil por conta do amplo espaço de busca e do grande número de mínimos locais, com apenas um mínimo global.

A função Rosenbrock também é não convexa, multimodal e não separável. A dificuldade aqui é encontrar o ponto de mínimo global. Ele fica num vale muito estreito, parabólico e plano. Em duas dimensões ela é unimodal, porém acima de quatro dimensões ela é considerada multimodal, de acordo com Shang e Qiu (2006).

A função Griewank é multimodal e não separável, possuindo muitos pontos de mínimo local regularmente distribuídos. É uma função largamente utilizada para testar a convergência dos algoritmos de busca.

As funções Shifted Rosenbrock e Shifted Rastringin são funções derivadas das duas funções já citadas de mesmo nome. Foram retiradas da competição que ocorre na congresso IEEE CEC (Congresso em Computação Evolutiva) de maio de 2005. A função Shifted Rosenbrock é um função multimodal, não separável, escalável que tem um vale muito estreito e um espaço de busca amplo. A função shifted Rastringin é multimodal, separável, escalável e apresenta como dificultador uma quantidade enorme de mínimos locais.

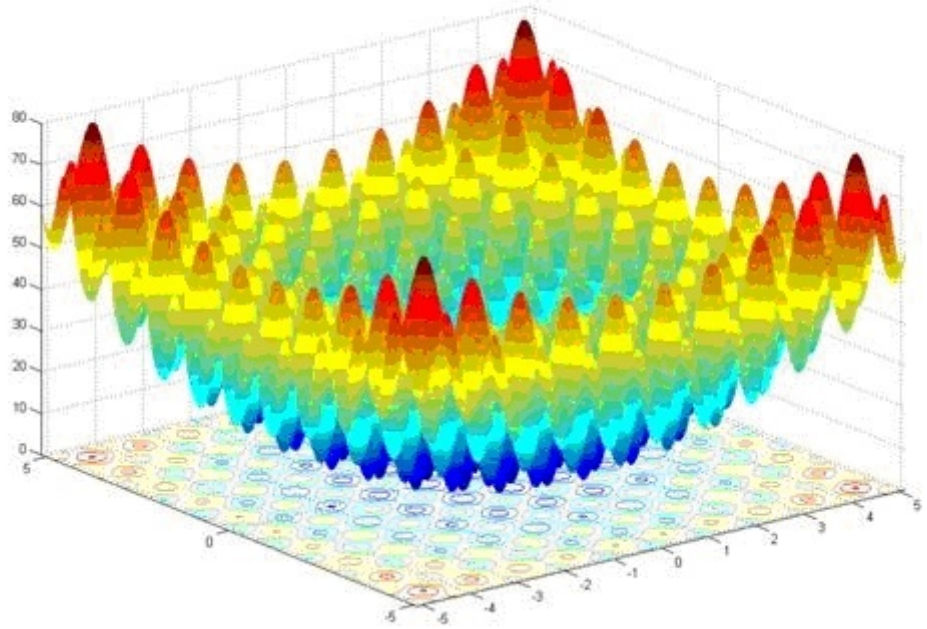

Figura 21 – Função Rastringin

Fonte: <http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page2607.htm>

5.2.2 Despacho Econômico Dinâmico de Energia Elétrica

A sétima função a ser utilizada como benchmark será a DED (Despacho Econômico Dinâmico). Ela é uma função multimodal (SRIYANYONG; LU, 2013) muito complexa. Ela tem por objetivo otimizar o despacho de energia, resolvendo assim um problema do sistema energético. A geração de saída de energia de cada unidade é calculada de acordo com a carga demandada prevista para um período. Essa geração deve obedecer algumas restrições do sistema. O objetivo é minimizar o custo de geração de energia satisfazendo todas as unidades e as restrições. Quando a demanda é alterada a geração de energia deve acompanhá-la. Na implementação da DED são levados em conta os custos dinâmicos envolvidos em sair de um nível de geração de energia para outro. De acordo com Kumar e Alwarsamy (2011):

> "DED é um problema dinâmico devido a natureza dinâmica do sistema energético, a larga variação de demanda de carga. Esse problema pode ser resolvido pela discretização de todo o período de despacho num número de intervalos menores, sobre os quais a carga é assumida como constante e o sistema é considerado como um estado temporal constante. "

A minimização desse custo de geração está sujeita a várias restrições:

a) Restrição de equilíbrio de energia real

b) Limite de geração real de energia

c) Limites de taxa de subida de unidade de geração.

Na Equação 5.1, F é o custo total de geração no período de despacho, T é o número de intervalos agendados (nesta dissertação foram utilizados 24 intervalos de 1 hora) e N é o número de unidade de geração(nesta dissertação são utilizadas 5 unidades de geração). $F_{it}(P_{it})$ é o custo do combustível de acordo com a potência de saída real P_{it} num intervalo de tempo t.

$$\min F = \sum_{t=1}^{T} \sum_{n=1}^{N} F_{it}(P_{it}) \tag{5.1}$$

Levando em consideração os efeitos de válvula, a função de custo do combustível da unidade de geração i pode ser expressa como a soma de uma função quadrática e sinusoidal como demonstra a Equação 5.2. Os termos a_i, b_i, e c_i são coeficientes de custo, e_i e fi são constantes do efeito de válvula na unidade de geração i e P_i é a potência de saída dessa unidade em MW. Essa minimização de custo esta sujeita a várias restrições.

$$F_{it}(P_{it}) = a_i P_{it}^2 + b_i P_{it} + c_i |e_i \sin f_i (P_{imin} - P_{it})| \tag{5.2}$$

A restrição de equilíbrio de carga (Equação 5.3). P_{Dt} é a demanda total de potência num período t e P_{Lt} é a perda de potência na transmissão nesse mesmo período, ambos em MW. Outra restrição é a capacidade de geração de energia de cada unidade (Equação 5.4). P_{imin} e P_{imax} são, respectivamente, os limites mínimo e máximo de potência da unidade i. As Equações 5.5 e 5.6 descrevem a restrição de limite de rampa de cada unidade de geração i. UR_i e DR_i são respectivamente o limite de aumento e o limite de diminuição de geração de energia de cada unidade i.

$$\sum_{i=1}^{N} P_{it} - P_{Dt} - P_{Lt} = 0, ondet = 1, 2...T \tag{5.3}$$

$$P_{imin} \leq P_{it} \leq P_{imax}, t = 1, 2...N, i = 1, 2...n \tag{5.4}$$

$$P_{it} - P_{it-1} \leq UR_i, ondei = 1, 2...N \tag{5.5}$$

$$P_{it-1} - P_{it} \leq DR_i, ondei = 1, 2...N \tag{5.6}$$

5.3 Resultados

O algoritmo autoajustável misto foi comparado a todas as variações possíveis de operadores, com as melhores taxas de cruzamento e mutação encontradas em testes (75% e 5% respectivamente). A Tabela 3 descreve os dezoito grupos de testes.

Tabela 3 – Modos de Comparação do Algoritmo Autoadaptável

Modo	Operadores
MOD1	Algoritmo Operadores adaptáveis e Taxas autoajustáveis
MOD2	Cruzamento e Mutação Aleatórios
MOD3	Cruzamento Simples , Mutação Uniforme
MOD4	Cruzamento Simples , Mutação Não Uniforme
MOD5	Cruzamento Simples , Mutação Creep
MOD6	Cruzamento Simples , Mutação Eapso
MOD7	Cruzamento Aritmético Uniforme , Mutação Uniforme
MOD8	Cruzamento Aritmético Uniforme , Mutação Não Uniforme
MOD9	Cruzamento Aritmético Uniforme , Mutação Creep
MOD10	Cruzamento Aritmético Uniforme , Mutação Eapso
MOD11	Cruzamento Aritmético Não Uniforme , Mutação Uniforme
MOD12	Cruzamento Aritmético Não Uniforme , Mutação Não Uniforme
MOD13	Cruzamento Aritmético Não Uniforme , Mutação Creep
MOD14	Cruzamento Aritmético Não Uniforme , Mutação Eapso
MOD15	Cruzamento Linear , Mutação Uniforme
MOD16	Cruzamento Linear , Mutação Não Uniforme
MOD17	Cruzamento Linear , Mutação Creep
MOD18	Cruzamento Linear , Mutação Eapso

Para avaliar se a diferença entre os grupos de teste é significativa será utilizado o método ANOVA. Nele, a hipótese H_0 define que as médias são iguais, ou seja, não há diferença entre as abordagens. A região de aceitação na tabela F do método ANOVA para a hipótese H_0 vai até 2,605. Valores de F calculados acima deste valor atestam que existe pelo menos uma diferença entre médias considerada significativa. Para cada função serão mostradas duas tabelas de resultado. A primeira tabela mostra o resultado da melhor e da pior aptidões encontradas nas 50 execuções do algoritmo para o modo de testes, além da média e do desvio padrão para cada modo de execução do algoritmo. A segunda tabela mostra um resumo do teste de Tukey para a comparação entre os modos indicados na primeira coluna. A segunda coluna mostra a diferença entre as médias dos dois conjuntos de teste. O sinal negativo indica que o algoritmo genético autoadaptável misto tem média menor e, portanto se saiu melhor. A terceira coluna mostra a diferença mínima significativa (DMS) calculada pelo teste de Tukey, e a última coluna mostra a conclusão para essa comparação de modos.

5.3.1 Funções de Benchmark

O primeiro conjunto de testes foi com a função Rosenbrock, utilizando 600 gerações. A Tabela 4 mostra um resumo dos resultados dos testes. O valor encontrado de F do método ANOVA é 498,0224, logo existe pelo menos uma média diferente. A Tabela 5 demonstra se a diferença entre as médias é significativa utilizando o teste de Tukey. A média dos resultados obtidos com o algoritmo autoajustável misto é melhor que a média de todos os outros resultados. Também é possível concluir que sempre que a diferença é significativa o autoajuste é melhor pois sua média é menor. Esse experimento foi repetido com 2000 gerações. Os resultados melhoraram e o teste de Tukey chegou às mesmas conclusões.

Tabela 4 – Resultados da função Rosenbrock com 600 gerações

MOD	Melhor	Pior	Média	Desvio Padrão
MOD1	**10,661**	249,625	**66,130**	**54,319**
MOD2	359,838	16083,069	1994,229	2308,602
MOD3	437,967	3574,307	1651,582	827,296
MOD4	923,74	27982,091	3915,97	3985,437
MOD5	2910,253	188310,702	46802,65	36212,203
MOD6	502701,034	1888921,012	1212031,973	372145,156
MOD7	1054,948	7237,131	2594,674	1117,641
MOD8	816,276	5939,489	2733,405	1293,926
MOD9	934,813	15978,454	4852,811	3604,537
MOD10	84091,524	413566,791	201554,697	70173,723
MOD11	548,088	5537,821	1958,144	1136,286
MOD12	1141,308	20902,151	3717,267	2998,901
MOD13	14770,229	413123,45	125087,778	72399,304
MOD14	801516,817	**2481710,641**	1639923,529	478561,761
MOD15	533,281	4056,069	1614,861	768,934
MOD16	920,165	4750,18	2188,903	861,58
MOD17	150,608	973,069	414,998	183,966
MOD18	41005,498	270805,831	142642,436	57389,561

O segundo conjunto de testes foi com a função Griewank, utilizando 600 gerações. A Tabela 6 mostra um resumo dos resultados dos testes. O valor encontrado de F do método ANOVA é 2096,079, logo existe pelo menos uma média diferente. A Tabela 7 demonstra se a diferença entre as médias é significativa utilizando o teste de Tukey. Dos resultados é possível concluir que sempre que a diferença é significativa o autoajuste é melhor pois sua média é menor. Esse experimento foi repetido com 2000 gerações. Os resultados melhoraram e o teste de Tukey chegou às mesmas conclusões.

O terceiro conjunto de testes foi com a função Rastringin, utilizando 600 gerações. A Tabela 8 mostra um resumo dos resultados dos testes. O valor encontrado de F do método ANOVA é 1666,248, logo existe pelo menos uma média diferente. A Tabela 9

Tabela 5 – Resultados do Anova para função Rosenbrock com 600 gerações

MODs	Difer. de Médias	DMS	Conclusão
1 e 2	-1928,100	74818,101	Diferença não significativa
1 e 3	-1585,453	74818,101	Diferença não significativa
1 e 4	-3849,840	74818,101	Diferença não significativa
1 e 5	-46736,520	74818,101	Diferença não significativa
1 e 6	-1211965,843	74818,101	Diferença significativa. Autoajuste é melhor
1 e 7	-2528,544	74818,101	Diferença não significativa
1 e 8	-2667,275	74818,101	Diferença não significativa
1 e 9	-4786,681	74818,101	Diferença não significativa
1 e 10	-201488,568	74818,101	Diferença significativa. Autoajuste é melhor
1 e 11	-1892,014	74818,101	Diferença não significativa
1 e 12	-3651,138	74818,101	Diferença não significativa
1 e 13	-125021,648	74818,101	Diferença significativa. Autoajuste é melhor
1 e 14	**-1639857,399**	74818,101	Diferença significativa. Autoajuste é melhor
1 e 15	-1548,731	74818,101	Diferença não significativa
1 e 16	-2122,773	74818,101	Diferença não significativa
1 e 17	-348,868	74818,101	Diferença não significativa
1 e 18	-142576,306	74818,101	Diferença significativa. Autoajuste é melhor

Tabela 6 – Resultados da função Griewank com 600 gerações

MOD	Melhor	Pior	Média	Desvio Padrão
MOD1	**0,127**	1,296	**0,533**	**0,261**
MOD2	1,281	9,31	3,036	1,834
MOD3	3,626	17,679	9,291	3,251
MOD4	5,105	27,708	14,88	5,199
MOD5	17,668	109,877	61,866	23,431
MOD6	329,458	676,035	507,25	66,532
MOD7	2,055	8,406	4,562	1,526
MOD8	1,620	10,006	4,928	1,763
MOD9	1,055	4,717	1,899	1,019
MOD10	46,976	188,287	85,232	28,12
MOD11	6,221	22,497	11,941	3,362
MOD12	6,268	44,269	19,893	9,158
MOD13	55,103	171,861	111,744	31,154
MOD14	390,827	**758,855**	609,613	77,862
MOD15	1,291	8,540	3,086	1,484
MOD16	1,300	12,189	3,403	2,074
MOD17	0,911	1,192	1,053	0,043
MOD18	21,644	107,324	62,813	20,636

demonstra se a diferença entre as médias é significativa utilizando o teste de Tukey. A média das execuções do algoritmo ajustável misto foi melhor que todas as outras. A diferença foi considerada significativa em todos os cenários e o autoajuste melhor em todas

Tabela 7 – Resultados do Anova para função Griewank com 600 gerações

MODs	Difer. de Médias	DMS	Conclusão
1 e 2	-2,503	13,993	Diferença não significativa
1 e 3	-8,758	13,993	Diferença não significativa
1 e 4	-14,347	13,993	Diferença significativa. Autoajuste é melhor
1 e 5	-61,333	13,993	Diferença significativa. Autoajuste é melhor
1 e 6	-506,717	13,993	Diferença significativa. Autoajuste é melhor
1 e 7	-4,03	13,993	Diferença não significativa
1 e 8	-4,395	13,993	Diferença não significativa
1 e 9	-1,366	13,993	Diferença não significativa
1 e 10	-84,699	13,993	Diferença significativa. Autoajuste é melhor
1 e 11	-11,408	13,993	Diferença não significativa
1 e 12	-19,36	13,993	Diferença significativa. Autoajuste é melhor
1 e 13	-111,211	13,993	Diferença significativa. Autoajuste é melhor
1 e 14	**-609,080**	13,993	Diferença não significativa
1 e 15	-2,553	13,993	Diferença não significativa
1 e 16	-2,87	13,993	Diferença não significativa
1 e 17	-0,52	13,993	Diferença não significativa
1 e 18	-62,28	13,993	Diferença significativa. Autoajuste é melhor

as comparações. Esse experimento foi repetido com 2000 gerações. As diferenças ficaram ainda maiores e o teste de Tukey chegou às mesmas conclusões.

Tabela 8 – Resultados da função Rastringin com 600 gerações

MOD	Melhor	Pior	Média	Desvio Padrão
MOD1	**1,087**	17,378	**5,844**	**2,976**
MOD2	14,150	46,156	30,314	7,028
MOD3	24,372	50,456	38,208	6,600
MOD4	40,822	90,696	63,114	11,645
MOD5	98,046	224,897	161,481	28,408
MOD6	252,559	**441,908**	358,456	34,12
MOD7	17,795	39,292	25,503	5,148
MOD8	14,205	57,981	34,921	8,565
MOD9	19,716	114,417	48,880	18,560
MOD10	178,245	283,676	239,198	26,542
MOD11	30,152	65,010	43,974	8,155
MOD12	43,634	87,751	68,208	11,148
MOD13	111,639	253,834	181,316	29,809
MOD14	307,720	439,312	394,174	27,909
MOD15	10,465	34,664	18,648	5,397
MOD16	12,884	48,07	25,913	7,228
MOD17	24,375	78,158	45,756	14,045
MOD18	124,897	326,52	205,628	47,106

Tabela 9 – Resultados do Anova para função Rastringin com 600 gerações

MODs	Difer. de Médias	DMS	Conclusão
1 e 2	-24,471	10,627	Diferença significativa. Autoajuste é melhor
1 e 3	-32,365	10,627	Diferença significativa. Autoajuste é melhor
1 e 4	-57,270	10,627	Diferença significativa. Autoajuste é melhor
1 e 5	-155,638	10,627	Diferença significativa. Autoajuste é melhor
1 e 6	-352,612	10,627	Diferença significativa. Autoajuste é melhor
1 e 7	-19,660	10,627	Diferença significativa. Autoajuste é melhor
1 e 8	-29,077	10,627	Diferença significativa. Autoajuste é melhor
1 e 9	-43,037	10,627	Diferença significativa. Autoajuste é melhor
1 e 10	-233,355	10,627	Diferença significativa. Autoajuste é melhor
1 e 11	-38,131	10,627	Diferença significativa. Autoajuste é melhor
1 e 12	-62,364	10,627	Diferença significativa. Autoajuste é melhor
1 e 13	-175,472	10,627	Diferença significativa. Autoajuste é melhor
1 e 14	**-388,330**	10,627	Diferença significativa. Autoajuste é melhor
1 e 15	-12,805	10,627	Diferença significativa. Autoajuste é melhor
1 e 16	-20,070	10,627	Diferença significativa. Autoajuste é melhor
1 e 17	-39,913	10,627	Diferença significativa. Autoajuste é melhor
1 e 18	-199,784	10,627	Diferença significativa. Autoajuste é melhor

O quarto conjunto de testes foi com a função Schwefel, utilizando 600 gerações. A Tabela 10 mostra um resumo dos resultados dos testes. O valor encontrado de F do método ANOVA é 1868,189, logo existe pelo menos uma média diferente. A Tabela 11 demonstra se a diferença entre as médias é significativa utilizando o teste de Tukey. A média das execuções do algoritmo ajustável misto foi melhor que todas as outras. A diferença foi considerada significativa em todos os cenários e o autoajuste melhor em todas as comparações. Esse experimento foi repetido com 2000 gerações. As diferenças ficaram maiores e o teste de Tukey chegou às mesmas conclusões.

Por fim, foram testadas as funções Shifted Rosenbrock e Shifted Rastringin. O quinto conjunto de testes foi com a função Shifted Rosenbrock, utilizando 600 gerações. A Tabela 12 mostra um resumo dos resultados dos testes. O valor encontrado de F do método ANOVA é 472,004, logo existe pelo menos uma média diferente. A Tabela 13 demonstra se a diferença entre as médias é significativa utilizando o teste de Tukey. Dos resultados é possível concluir que sempre que a diferença é significativa o autoajuste é melhor pois sua média é menor. Esse experimento foi repetido com 2000 gerações. Os resultados melhoraram e o teste de Tukey chegou às mesmas conclusões.

O sexto conjunto de testes foi com a função Shifted Rastringin, utilizando 600 gerações. A Tabela 14 mostra um resumo dos resultados dos testes. O valor encontrado de F do método ANOVA é 2118,644, logo existe pelo menos uma média diferente. A Tabela 15 demonstra se a diferença entre as médias é significativa utilizando o teste de Tukey. A média das execuções do algoritmo ajustável misto foi melhor que todas as outras.

Tabela 10 – Resultados da função Schewefel com 600 gerações

MOD	Melhor	Pior	Média	Desvio Padrão
MOD1	**0,605**	600,419	**80,982**	**128,862**
MOD2	798,460	2680,806	1639,720	442,684
MOD3	1083,624	2203,935	1533,091	239,477
MOD4	704,944	2171,303	1494,230	365,345
MOD5	3959,191	6598,760	5162,699	685,324
MOD6	8500,96	10670,417	9586,063	560,172
MOD7	859,694	2523,856	1569,434	378,612
MOD8	595,398	2281,879	1391,299	342,107
MOD9	3948,789	8727,799	6115,51	1157,558
MOD10	8464,206	10991,394	10113,612	600,492
MOD11	773,135	2588,265	1620,587	358,200
MOD12	974,364	2445,304	1729,356	363,898
MOD13	3557,037	7505,266	5497,103	740,829
MOD14	8962,474	**11530,162**	10208,205	656,88
MOD15	646,668	2000,464	1311,141	324,248
MOD16	722,119	2395,984	1473,366	410,582
MOD17	3547,067	6376,273	5180,51	633,589
MOD18	6580,589	10390,100	8912,902	953,468

Tabela 11 – Resultados do Anova para função Schewefel com 600 gerações

MODs	Difer. de Médias	DMS	Conclusão
1 e 2	-1558,737	295,723	Diferença significativa. Autoajuste é melhor
1 e 3	-1452,109	295,723	Diferença significativa. Autoajuste é melhor
1 e 4	-1413,247	295,723	Diferença significativa. Autoajuste é melhor
1 e 5	-5081,717	295,723	Diferença significativa. Autoajuste é melhor
1 e 6	-9505,081	295,723	Diferença significativa. Autoajuste é melhor
1 e 7	-1488,452	295,723	Diferença significativa. Autoajuste é melhor
1 e 8	-1310,316	295,723	Diferença significativa. Autoajuste é melhor
1 e 9	-6034,528	295,723	Diferença significativa. Autoajuste é melhor
1 e 10	-10032,63	295,723	Diferença significativa. Autoajuste é melhor
1 e 11	-1539,605	295,723	Diferença significativa. Autoajuste é melhor
1 e 12	-1648,374	295,723	Diferença significativa. Autoajuste é melhor
1 e 13	-5416,12	295,723	Diferença significativa. Autoajuste é melhor
1 e 14	**-10127,223**	295,723	Diferença significativa. Autoajuste é melhor
1 e 15	-1230,158	295,723	Diferença significativa. Autoajuste é melhor
1 e 16	-1392,383	295,723	Diferença significativa. Autoajuste é melhor
1 e 17	-5099,528	295,723	Diferença significativa. Autoajuste é melhor
1 e 18	-8831,919	295,723	Diferença significativa. Autoajuste é melhor

A diferença foi considerada significativa em todos os cenários e o autoajuste melhor em todas as comparações. Esse experimento foi repetido com 2000 gerações. As diferenças ficaram ainda maiores e o teste de Tukey chegou às mesmas conclusões.

Tabela 12 – Resultados da função Shifted Rosenbrock com 600 gerações

MOD	Melhor	Pior	Média	Desvio Padrão
MOD1	**-165,824**	15330,214	**1698,997**	**2873,277**
MOD2	2,931E+05	7,530E+08	6,796E+07	1,687E+08
MOD3	1,347E+06	9,644E+07	1,956E+07	1,984E+07
MOD4	2,844E+06	2,673E+08	4,123E+07	5,003E+07
MOD5	3,949E+07	1,747E+10	6,378E+09	4,413E+09
MOD6	3,512E+10	1,476E+11	7,645E+10	2,671E+10
MOD7	7,863E+05	1,879E+08	1,906E+07	2,940E+07
MOD8	2,100E+05	3,924E+07	6,684E+06	7,923E+06
MOD9	3,089E+08	1,156E+10	2,159E+09	1,766E+09
MOD10	1,670E+10	6,798E+10	3,315E+10	1,065E+10
MOD11	3,509E+06	1,390E+08	2,419E+07	2,257E+07
MOD12	4,546E+06	1,439E+08	4,847E+07	3,871E+07
MOD13	2,338E+09	3,563E+10	1,580E+10	7,518E+09
MOD14	4,184E+10	**2,016E+11**	1,204E+11	3,267E+10
MOD15	2,524E+05	1,104E+08	8,284E+06	1,973E+07
MOD16	8,970E+04	5,187E+07	4,374E+06	8,228E+06
MOD17	6,192E+05	3,542E+08	3,035E+07	5,591E+07
MOD18	8,355E+09	3,388E+10	2,133E+10	6,466E+09

Para medir a eficácia do algoritmo em relação ao mínimo de cada função a Tabela 16 mostra o menor resultado encontrado em relação ao mínimo de cada função.

5.3.2 Despacho Econômico Dinâmico de Energia Elétrica

A função de despacho econômico dinâmico de energia elétrica foi testada inicialmente utilizando 600 gerações. A Tabela 17 informa a demanda de energia por hora. A Tabela 18 mostra os parâmetros da planta de geração de energia. São 5 unidade de geração que têm um limite mínimo (P_{min}) e um limite máximo (P_{max}) de geração de energia. UR e DR são respectivamente o limite de aumento e o limite de diminuição de geração de energia de cada unidade. Por fim são mostrados os coeficientes de custo (a, b e c) e as constantes do efeito de válvula e e f para cada unidade. A Tabela 19 mostra a matriz B de coeficientes de perda usada para calcular P_{Lt} que é a perda na transmissão de energia num período t. A função e esses parâmetros são descritos na Seção 5.2.2.

As restrições foram tratadas com o método de penalidade estática conforme a Equação 5.7. Nessa equação, R_u indica o valor que ultrapassou a restrição. A constante m depende da importância daquela restrição para o problema. Para a restrição de balanceamento de carga e de limite de geração real de energia, m recebeu valor de 1. Para a restrição de limite de taxa de subida de unidade de geração, m recebeu o valor de 0,4. A razão para a escolha desses valores foi a configuração dos artigos usados para comparação,

Tabela 13 – Resultados do Anova para função Shifted Rosenbrock com 600 gerações

MODs	Difer. de Médias	DMS	Conclusão
1 e 2	-6,796E+07	5,432E+09	Diferença não significativa
1 e 3	-1,955E+07	5,432E+09	Diferença não significativa
1 e 4	-4,123E+07	5,432E+09	Diferença não significativa
1 e 5	-6,379E+09	5,432E+09	Diferença significativa. Autoajuste é melhor
1 e 6	-7,645E+10	5,432E+09	Diferença significativa. Autoajuste é melhor
1 e 7	-1,906E+07	5,432E+09	Diferença não significativa
1 e 8	-6,682E+06	5,432E+09	Diferença não significativa
1 e 9	-2,159E+09	5,432E+09	Diferença não significativa
1 e 10	-3,315E+10	5,432E+09	Diferença significativa. Autoajuste é melhor
1 e 11	-2,419E+07	5,432E+09	Diferença não significativa
1 e 12	-4,847E+07	5,432E+09	Diferença não significativa
1 e 13	-1,580E+10	5,432E+09	Diferença significativa. Autoajuste é melhor
1 e 14	**-1,204E+11**	5,432E+09	Diferença significativa. Autoajuste é melhor
1 e 15	-8,282E+06	5,432E+09	Diferença não significativa
1 e 16	-4,372E+06	5,432E+09	Diferença não significativa
1 e 17	-3,034E+07	5,432E+09	Diferença não significativa
1 e 18	-2,133E+10	5,432E+09	Diferença significativa. Autoajuste é melhor

mostrados na Tabela 24.

$$P = 1000 * m * (R_u) \tag{5.7}$$

A Tabela 20 mostra um resumo dos resultados dos testes. O valor encontrado de F do método ANOVA é 2262,298. Como a região de aceitação na tabela F para a hipótese H_0 vai até 2,605 então rejeita-se H_0 e aceita-se que existe pelo menos uma média diferente. A Tabela 21 demonstra se a diferença entre as médias é significativa utilizando o teste de Tukey. A média das execuções do algoritmo ajustável misto foi melhor que todas as outras. A diferença foi considerada significativa em todos os cenários e o autoajuste melhor em todas as comparações.

Esse experimento foi repetido com 2000 gerações. As médias melhoraram conforme mostra a Tabela 22. O valor encontrado de F do método ANOVA é 3545,329. Como a região de aceitação na tabela F para a hipótese H_0 vai até 2,605 então rejeita-se H_0 e aceita-se que existe pelo menos uma média diferente. A Tabela 23 demonstra se a diferença entre as médias é significativa utilizando o teste de Tukey. A média das execuções do

Tabela 14 – Resultados da função Shifted Rastringin com 600 gerações

MOD	Melhor	Pior	Média	Desvio Padrão
MOD1	**-328,387**	-308,091	**-322,207**	**4,339**
MOD2	-304,041	-258,169	-282,977	10,027
MOD3	-291,526	-243,114	-272,032	12,196
MOD4	-292,666	-249,309	-274,866	9,549
MOD5	-173,192	23,735	-62,872	43,655
MOD6	64,280	280,501	180,517	43,078
MOD7	-298,688	-258,074	-281,259	8,162
MOD8	-307,682	-275,089	-292,647	8,550
MOD9	-164,324	-29,318	-114,516	29,700
MOD10	-3,491	140,089	79,977	36,940
MOD11	-285,709	-239,474	-265,784	10,577
MOD12	-285,513	-244,316	-270,976	9,341
MOD13	-117,749	94,247	-23,435	42,290
MOD14	141,114	**321,275**	245,183	40,361
MOD15	-310,814	-274,784	-297,167	8,234
MOD16	-314,794	-287,360	-301,925	6,537
MOD17	-174,413	-61,809	-123,576	29,486
MOD18	-91,410	168,924	38,612	55,457

Tabela 15 – Resultados do Anova para função Shifted Rastringin com 600 gerações

MODs	Difer. de Médias	DMS	Conclusão
1 e 2	-39,230	14,408	Diferença significativa. Autoajuste é melhor
1 e 3	-50,175	14,408	Diferença significativa. Autoajuste é melhor
1 e 4	-47,341	14,408	Diferença significativa. Autoajuste é melhor
1 e 5	-259,336	14,408	Diferença significativa. Autoajuste é melhor
1 e 6	-502,725	14,408	Diferença significativa. Autoajuste é melhor
1 e 7	-40,948	14,408	Diferença significativa. Autoajuste é melhor
1 e 8	-29,561	14,408	Diferença significativa. Autoajuste é melhor
1 e 9	-207,692	14,408	Diferença significativa. Autoajuste é melhor
1 e 10	-402,184	14,408	Diferença significativa. Autoajuste é melhor
1 e 11	-56,423	14,408	Diferença significativa. Autoajuste é melhor
1 e 12	-51,231	14,408	Diferença significativa. Autoajuste é melhor
1 e 13	-298,773	14,408	Diferença significativa. Autoajuste é melhor
1 e 14	**-567,391**	14,408	Diferença significativa. Autoajuste é melhor
1 e 15	-25,041	14,408	Diferença significativa. Autoajuste é melhor
1 e 16	-20,283	14,408	Diferença significativa. Autoajuste é melhor
1 e 17	-198,632	14,408	Diferença significativa. Autoajuste é melhor
1 e 18	-360,820	14,408	Diferença significativa. Autoajuste é melhor

algoritmo ajustável misto foi melhor que todas as outras. A diferença foi considerada significativa em todos os cenários e o autoajuste melhor em todas as comparações.

Tabela 16 – Comparativo dos Resultados Funções de Benchmark

Nome	Mínimo	MOD1 600 gerações	MOD1 2000 gerações
Schwefel	0	0,605	**0,028**
Rastringin	0	1,087	0,056
Rosenbrock	0	10,661	1,573
Griewank	0	0,127	0,086
Shifted Rosenbrock	o	-165,824	-280,731
Shifted Rastringin	o	-328,387	-329,981

Tabela 17 – Demanda de Energia por Hora

Hora	Demanda (MW)	Hora	Demanda (MW)
1	410	2	435
3	475	4	530
5	558	6	608
7	626	8	654
9	690	10	704
11	720	12	740
13	704	14	690
15	654	16	580
17	558	18	608
19	654	20	704
21	680	22	605
23	527	24	463

Tabela 18 – Dados das Unidades de Geração

Unidade	P_{min}	P_{max}	UR	DR	a	b	c	e	f
1	10	75	30	30	0,0080	2,0	25	100	0,042
2	20	125	30	30	0,0030	1,8	60	140	0,040
3	30	175	40	40	0,0012	2,1	100	160	0,038
4	40	250	50	50	0,0010	2,0	120	180	0,037
5	50	300	50	50	0,0015	1,8	40	200	0,035

Para medir a eficácia dos resultados, os resultados produzidos pela função de Despacho Econômico Dinâmico de energia nesta dissertação (1) foram comparados com os resultados dos artigos Niknam e Golestaneh (2012), página 429, tabela 3 (2), Balamurugan e Subramanian (2007) página 156 (3) e Chakraborty et al. (2012) página 201, tabela 4 (4) na Tabela 24.

5.3.3 Comparação AG Autoajustável Paralelo com Thread única

A segunda linha de testes comparou o tempo de execução do algoritmo autoajustável paralelo com o algoritmo autoajustável de thread única. O tempo foi medido em

Tabela 19 – Matriz B de Coeficientes de perda

0,000049	0,000014	0,000015	0,000015	0,000020
0,000014	0,000045	0,000016	0,000020	0,000018
0,000015	0,000016	0,000039	0,000010	0,000012
0,000015	0,000020	0,000010	0,000040	0,000014
0,000020	0,000018	0,000012	0,000014	0,000035

Tabela 20 – Resultados da função Despacho Econômico Dinâmico com 600 gerações

MOD	Melhor	Pior	Média	Desvio Padrão
MOD1	**43033,396**	44145,354	**43508,565**	**267,858**
MOD2	44739,187	46777,094	45666,455	447,813
MOD3	44947,823	46792,516	45955,839	365,487
MOD4	45254,125	47083,895	46265,956	406,646
MOD5	46985,017	49807,796	48293,546	708,029
MOD6	50507,091	53266,839	51747,328	586,930
MOD7	45782,635	47732,233	46700,023	469,022
MOD8	46064,331	47669,766	46788,025	453,361
MOD9	47973,116	50197,533	49246,654	522,190
MOD10	50793,328	52741,426	51943,302	504,483
MOD11	46774,862	48913,740	48085,977	413,210
MOD12	48076,383	49987,787	48950,502	475,592
MOD13	51691,106	54283,460	52862,064	551,949
MOD14	54838,061	**56867,487**	55780,407	467,042
MOD15	44345,968	45870,094	45021,371	287,278
MOD16	44724,122	45898,430	45292,000	325,259
MOD17	45719,850	47819,367	46787,815	429,319
MOD18	48545,007	51191,313	49787,084	601,141

milisegundos e representa uma execução de 600 gerações. O algoritmo paralelo usou 2, 4 e 8 threads.

O fator de aceleração depende diretamente da quantidade de código executada em paralelo. O algoritmo paralelo desenvolvido optou por paralelizar uma porção menor do código que não afetasse os resultados aumentado o máximo possível do fator de aceleração. A Tabela 25 mostra o fator de aceleração desta opção. A primeira coluna indica a função de benchmark utilizada nos testes e as três próximas colunas descrevem o fator de aceleração obtido para 2, 4 e 8 threads. A Figura 22 mostra um gráfico com os fatores de aceleração obtidos, com os mesmos resultados da Tabela 25. A Tabela 26 mostra as eficiência calculada a partir do speedup e do número de processadores para 2, 4 e 8 threads.

A Tabela 27 compara as médias entre o algoritmo autoajustável de thread única e a versão com 4 threads no experimento de 600 gerações. O método ANOVA e o teste de tukey foram usados para testar se a diferença de médias foi significante. O resultado

Tabela 21 – Resultados do Anova para função Despacho Econômico Dinâmico com 600 gerações

MODs	Difer. de Médias	DMS	Conclusão
1 e 2	-2157,890	242,708	Diferença significativa. Autoajuste é melhor
1 e 3	-2447,275	242,708	Diferença significativa. Autoajuste é melhor
1 e 4	-2757,391	242,708	Diferença significativa. Autoajuste é melhor
1 e 5	-4784,981	242,708	Diferença significativa. Autoajuste é melhor
1 e 6	-8238,764	242,708	Diferença significativa. Autoajuste é melhor
1 e 7	-3191,458	242,708	Diferença significativa. Autoajuste é melhor
1 e 8	-3279,460	242,708	Diferença significativa. Autoajuste é melhor
1 e 9	-5738,090	242,708	Diferença significativa. Autoajuste é melhor
1 e 10	-8434,737	242,708	Diferença significativa. Autoajuste é melhor
1 e 11	-4577,412	242,708	Diferença significativa. Autoajuste é melhor
1 e 12	-5441,937	242,708	Diferença significativa. Autoajuste é melhor
1 e 13	-9353,499	242,708	Diferença significativa. Autoajuste é melhor
1 e 14	**-12271,842**	242,708	Diferença significativa. Autoajuste é melhor
1 e 15	-1512,806	242,708	Diferença significativa. Autoajuste é melhor
1 e 16	-1783,435	242,708	Diferença significativa. Autoajuste é melhor
1 e 17	-3279,251	242,708	Diferença significativa. Autoajuste é melhor
1 e 18	-6278,519	242,708	Diferença significativa. Autoajuste é melhor

Tabela 22 – Resultados da função Despacho Econômico Dinâmico com 2000 gerações

MOD	Melhor	Pior	Média	Desvio Padrão
MOD1	**42767,573**	43245,864	**42906,757**	**104,277**
MOD2	43019,313	44237,415	43559,208	337,673
MOD3	43234,213	45109,266	43834,154	345,072
MOD4	43130,079	45376,238	44141,758	464,060
MOD5	45450,687	47597,517	46491,233	579,420
MOD6	50210,717	52830,333	51689,640	591,785
MOD7	43470,334	44614,110	44075,960	237,773
MOD8	43300,878	44620,560	44005,227	281,200
MOD9	44943,132	46985,304	46115,605	467,918
MOD10	50328,768	53227,408	51894,550	494,512
MOD11	43854,935	45843,169	44744,837	453,699
MOD12	44377,233	46303,106	45229,291	448,603
MOD13	47559,731	50055,640	49005,567	606,409
MOD14	54689,069	**56883,415**	55852,117	501,519
MOD15	42981,929	43933,902	43300,202	219,038
MOD16	42842,028	43843,120	43336,296	251,725
MOD17	43324,205	45164,966	44130,372	429,116
MOD18	47704,689	49868,205	48777,485	620,822

atestou que a diferença das médias não foi significante em nenhuma das funções.

Tabela 23 – Resultados do Anova para função Despacho Econômico Dinâmico com 2000 gerações

MODs	Difer. de Médias	DMS	Conclusão
1 e 2	-652,451	224,800	Diferença significativa. Autoajuste é melhor
1 e 3	-927,398	224,800	Diferença significativa. Autoajuste é melhor
1 e 4	-1235,001	224,800	Diferença significativa. Autoajuste é melhor
1 e 5	-3584,477	224,800	Diferença significativa. Autoajuste é melhor
1 e 6	-8782,884	224,800	Diferença significativa. Autoajuste é melhor
1 e 7	-1169,203	224,800	Diferença significativa. Autoajuste é melhor
1 e 8	-1098,470	224,800	Diferença significativa. Autoajuste é melhor
1 e 9	-3208,849	224,800	Diferença significativa. Autoajuste é melhor
1 e 10	-8987,793	224,800	Diferença significativa. Autoajuste é melhor
1 e 11	-1838,080	224,800	Diferença significativa. Autoajuste é melhor
1 e 12	-2322,534	224,800	Diferença significativa. Autoajuste é melhor
1 e 13	-6098,810	224,800	Diferença significativa. Autoajuste é melhor
1 e 14	**-12945,360**	224,800	Diferença significativa. Autoajuste é melhor
1 e 15	-393,445	224,800	Diferença significativa. Autoajuste é melhor
1 e 16	-429,540	224,800	Diferença significativa. Autoajuste é melhor
1 e 17	-1223,616	224,800	Diferença significativa. Autoajuste é melhor
1 e 18	-5870,728	224,800	Diferença significativa. Autoajuste é melhor

Tabela 24 – Comparativo dos Resultados Despacho Econômico Dinâmico

Trabalho	Mínimo	Média
1 (Este Trabalho)	43033,395	43508,564
2 (Niknam e Golestaneh (2012))	43784	43794
3 (Balamurugan e Subramanian (2007))	45800	Não publicado
4 (Chakraborty et al. (2012))	43210,95	Não publicado

Tabela 25 – Fator de aceleração obtido para o MOD1 de acordo com o número de threads

Funtion	Speedup 2T	Speedup 4T	Speedup 8T
SCW	1,525	2,546	3,400
RAS	1,717	2,858	4,974
ROS	1,880	3,160	**5,177**
GRI	1,657	2,808	3,968
SROS	1,777	2,965	5,006
SRAS	1,692	2,800	4,135
DED	1,829	2,635	3,914

5.4 Considerações Finais

A Tabela 28 resume por função os testes que compararam o algoritmo genético autoajustável misto e todas as combinações de operadores de cruzamento e mutação. Essa

Tabela 26 – Eficiência obtida para o MOD1 de acordo com o número de threads

Função	Eficiência 2T	Eficiência 4T	Eficiência 8T
SCW	0,762	0,637	0,425
RAS	0,859	0,715	0,622
ROS	**0,940**	0,790	0,647
GRI	0,829	0,702	0,496
SROS	0,889	0,741	0,626
SRAS	0,846	0,700	0,517
DED	0,915	0,659	0,489

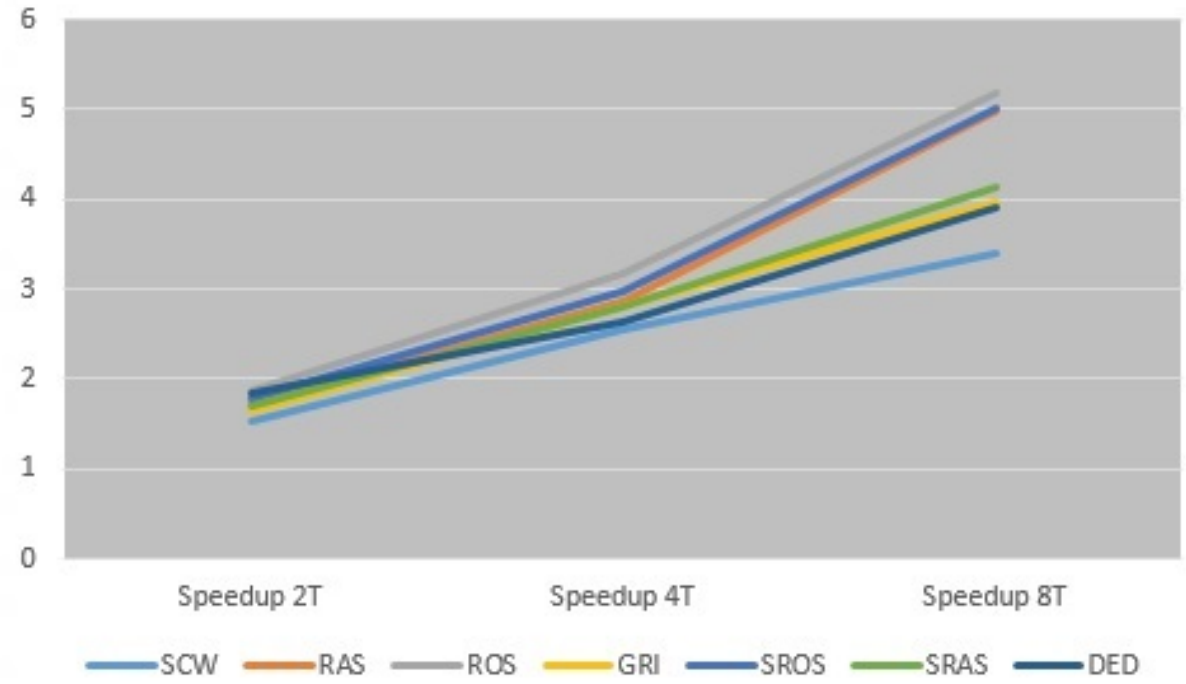

Figura 22 – Fator de aceleração para o MOD1 com 2, 4 e 8 threads

tabela tem 4 colunas. A primeira informa a função objeto da comparação, a segunda mostra a quantidade de comparações feitas. São 17 pois foram comparados o algoritmo autoajustável misto com um algoritmo que escolhe os operadores aleatoriamente e com 16 combinações de operadores de cruzamento e mutação. A terceira coluna conta a quantidade de vezes que a média das 50 execuções do algoritmo autoajustável misto foi menor que a média das execuções do algoritmo comparado. A quarta coluna mostra quantas vezes a diferença entre as médias é considerada significativa pelo método ANOVA e pelo teste de Tukey. A comparação entre a versão de thread única e as versões com 2 , 4 e 8 threads produziu resultados igualmente satisfatórios. A Tabela 29 mostra um resumo dos melhores resultados de speedup e eficiência encontrados por quantidade de threads.

Tabela 27 – Comparação entre Algoritmo de Thread única e Multithread - Qualidade das soluções

Função	Diferença de Médias Single/4T	DMS	Conclusão
SCW	7,939	39,574	Diferença não significativa
GRI	-0,004	0,061	Diferença não significativa
RAS	-0,622	1,299	Diferença não significativa
ROS	-2,492	21,101	Diferença não significativa
SRAS	-1,610	1,6167	Diferença não significativa
SROS	-542,825	1717,629	Diferença não significativa
DED	29,351	106,898	Diferença não significativa

Tabela 28 – Resumo da Comparação entre o Algoritmo Ajustável Misto e todas as combinações de operadores

Função	Qtd. de Comparações	Qtd. Média Menor	Qtd. Diferença Significativa
SCW	17	17	17
GRI	17	17	7
RAS	17	17	17
ROS	17	17	5
SRAS	17	17	17
SROS	17	17	6
DED	17	17	17

Tabela 29 – Melhores resultados de Speedup e Eficiência encontrados por quantidade de threads

Fator	2 Threads	4 Threads	8 Threads
Speedup	1,880	3,160	**5,178**
Eficiência	**0,940**	0,790	0,648

6 Conclusão

Esta dissertação apresentou um algoritmo genético autoajustável misto paralelo. Autoajustável misto pois as taxas de cruzamento e mutação são autoajustáveis e os operadores de cruzamento e mutação são ajustáveis. Para melhorar o desempenho, a solução foi paralelizada utilizando o OpenMp. Esse novo algoritmo foi testado em 6 funções de benchmark multimodais e em uma função com restrições, o despacho econômico dinâmico de energia elétrica. Esta dissertação enumerou duas hipóteses na sua introdução. A primeira, que o algoritmo autoadaptativo misto gera resultados melhores que o código sem autoadaptação na maioria das vezes que a diferença for considerada significativa . A segunda hipótese afirma que a paralelização do algoritmo autoadaptável misto usando OpenMp melhora a performance do algoritmo em comparação a versão sequencial sem perda significativa de qualidade.

Nas funções de benchmark, o algoritmo autoajustável misto se saiu melhor em todos os cenários testados e sempre que a diferença era significativa o algoritmo adaptável se saia melhor pois tinha uma média menor. A função de Despacho Econômico Dinâmico de energia também se saiu melhor em todos os cenários testados e alcançou resultados melhores que os artigos Niknam e Golestaneh (2012), página 429, tabela 3 , Balamurugan e Subramanian (2007) página 156 e Chakraborty et al. (2012) página 201, tabela 4. Com esses resultados a primeira hipótese está confirmada.

A paralelização do algoritmo foi feita utilizando o OpenMP. Essa API, aparentemente simples, revelou algumas dificuldades para se obter o melhor desempenho. Meses de testes foram necessários para ajustar seus parâmetros. A simples colocação ou não de uma variável nas claúsulas das diretivas pode impactar a eficiência ou a qualidade final da solução. O algoritmo paralelizado obteve um fator de aceleração de 5,18 e uma eficiência de 0,64 para 8 threads, e como não houve perda significativa de qualidade da solução, mais threads podem ser adicionadas para alcançar uma aceleração ainda maior. A segunda hipótese está confirmada então.

Como resultado destes estudos, foram publicados os trabalhos "A Stochastic Adaptive Genetic Algorithm for Solving Unconstrained Multimodal Numerical Problems"no IEEE EAIS e "A Parallel Adaptive Multithread Genetic Algorithm for Unconstrained Multimodal Numerical Optimization"no SBAI, qualis B2 e B5 respectivamente e um artigo para a revista Applied Soft Computing da editora Elsevier , qualis A2, foi enviado e está em avaliação. Para trabalhos futuros apontam-se:

1. Adicionar a lógica Fuzzy ao controle de adaptatividade – utilizar a lógica nebulosa para selecionar os operadores genéticos, mantendo a autoadaptação das taxas, criando

assim um algoritmo fuzzy-autoadaptativo.

2. Implementar a paralelização usando GPUs – implementar o AG autoadaptativo misto em uma arquitetura com muitos núcleos (many-core), em particular em uma General Purpose GPU (Processadores Gráficos de Propósito Geral) que é facilmente encontrada em computadores voltados para jogos.

3. Ainda em GPU, por ser uma arquitetura com alto potencial de paralelismo, pode-se adicionar outros operadores, como o de seleção, por exemplo; enquanto no autoajuste parâmetros tais como tamanho variável da população, taxa de nascimento, taxa de morte e quantidade de indivíduos participantes do elitismo, podem ser adicionados.

4. Adicionar à adaptatividade características de algoritmos híbridos, como por exemplo, adicionar características da evolução diferencial ou de enxame de partículas, paralelizando-os através de passagem de mensagem usando MPI (Message Passing Interface).

5. Adicionar aos Algoritmos Híbridos Paralelos adaptativos características de competição e cooperação, ou seja, diferentes algoritmos podem competir pela seleção de serem usados ou podem cooperar entre si para gerar soluções de maior qualidade. Sua paralelização pode se dar tanto em utilizando-se OpenMP, GPUs ou MPI.

Referências

ABBASS, H. A. The self-adaptive pareto differential evolution algorithm. In: *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*. [S.l.: s.n.], 2002. v. 1, p. 831–836. Citado na página 22.

ALSAEEDAN, W.; MENAI, M. E. B. A self-adaptive genetic algorithm for the word sense disambiguation problem. In: _____. *Current Approaches in Applied Artificial Intelligence*. [S.l.]: Springer, 2015. Citado na página 21.

BALAMURUGAN, R.; SUBRAMANIAN, S. An improved differential evolution based dynamic economic dispatch with nonsmooth fuel cost function. *Journal of Electrical Systems*, v. 3, n. 3, p. 151–161, 2007. Citado 4 vezes nas páginas 23, 71, 74 e 77.

BALIEIRO, A. M. et al. Estudo comparativo em algoritmo genético: Codificação real e codificação binária. In: *Anais do CNMAC, SBMAC, Sociedade Brasileira de Matemática Aplicada*. [S.l.: s.n.], 2008. Citado na página 28.

BELUR, S. Core: Constrained optimization by random evolution. *Late Breaking Papers at the Genetic Programming Conference, Stanford University*, 1997. Citado na página 34.

BENTLEY, P. J. *Evolutionary Design by Computers*. San Francisco: Morgan Kaufmann Publishers, 1999. Citado na página 27.

BERHE, H. W. Penalty function methods using matrix laboratory (matlab). *African Journal of Mathematics and Computer Science Research*, Novembro 2012. Citado na página 34.

CARVALHO, A. S. T.; ALMEIDA, M. R. M.; SILVA, J. C. P. Um estudo sobre a aplicação de algoritmos genéticos no investimento em ações utilizando análise técnica. *Proceeding Series of the Brazilian Society of Applied and Computational Mathematics*, n. 1, 2015. Citado na página 21.

CAVAZZUTI, M. Design of experiments. In: *Optimization Methods*. [S.l.]: Springer, 2013. p. 13–42. Citado na página 24.

CHAKRABORTY, P. et al. Dynamic economic dispatch using harmony search algorithm with modified differential mutation operator. *Electrical Engineering*, v. 94, n. 4, p. 197–205, 2012. ISSN 1432-0487. Citado 4 vezes nas páginas 23, 71, 74 e 77.

CHANDRA, R. *Parallel programming in OpenMP*. [S.l.]: Morgan kaufmann, 2001. Citado na página 37.

CORTES, O. A. C. Um sistema nebuloso-evolutivo para determinar o grau de portabilidade de benchmarks paralelos. *Tese de Doutorado em Ciências da Computação e Matemática Computacional. Universidade de São Paulo.*, 2004. Citado 2 vezes nas páginas 27 e 29.

CORTES, O. A. C. Aplicando algoritmos genéticos real-coded no refinamento de funções de pertinência fuzzy: Uma análise estatística. *X Congresso Brasileiro De Inteligência Computacional – Cbic*, Fortaleza, CE, Novembro 2011. Citado na página 21.

CORTES, O. A. C.; SILVA, J. C. A local search algorithm based on clonal selection and genetic mutation for global optimization. *Eleventh Brazilian Symposium on Neural Networks (SBRN)*, 2010. Citado 2 vezes nas páginas 22 e 58.

DEB, K.; BEYER, H.-g. Self-adaptive genetic algorithms with simulated binary crossover. *Evol. Comput.*, MIT Press, Cambridge, MA, USA, v. 9, n. 2, p. 197–221, jun. 2001. ISSN 1063-6560. Disponível em: <http://dx.doi.org/10.1162/106365601750190406>. Citado na página 22.

DEVOS, O.; DOWNEY, G.; DUPONCHEL, L. Simultaneous data pre-processing and svm classification model selection based on a parallel genetic algorithm applied to spectroscopic data of olive oils. *Food Chemistry*, v. 148, p. 124 – 130, 2014. ISSN 0308-8146. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0308814613014520>. Citado na página 24.

D.POWELL; SKOLNICK, M. Using genetic algorithms in engineering design optimization with non-linear constraints. *Proceedings of the Fifth International Conference on Genetic Algorithms*, 1993. Citado na página 34.

FLETCHER, R. *Practical Methods of Optimization*. [S.l.]: John Wiley and Sons, 2013. Citado na página 33.

FLYNN, M. J. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21, n. 9, p. 948–960, Sept 1972. ISSN 0018-9340. Citado na página 38.

FOX, G. C.; WILLIAMS, R. D.; MESSINA, G. C. *Parallel computing works!* [S.l.]: Morgan Kaufmann, 2014. Citado na página 37.

GEYER, C. F. et al. Sistemas distribuídos. *ERAD–ESCOLA REGIONAL DE ALTO DESEMPENHO*, v. 1, p. 195–204, 2001. Citado na página 37.

GOLUB, M. An implementation of binary and floating point chromosome representation in genetic algorithm. In: *Proceedings of the 18th International Conference ITI, Pula*. [S.l.: s.n.], 1996. p. 417–422. Citado na página 28.

HAJELA, P.; LEE, J. Constrained genetic search via schema adaptation, an immune network solution. *Structural Optimization*, 1997. Citado na página 34.

HERRERA, F.; M.LOZANO; VERDEGAY, J. Tackling real coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, n. 12, 1998. Citado 2 vezes nas páginas 21 e 28.

JANIKOW, C. Z.; MICHALEWICZ, Z. An experimental comparison of binary and floating point representations in genetic algorithms. In: *ICGA*. [S.l.: s.n.], 1991. p. 31–36. Citado na página 28.

KALEGARI, D. Algoritmo de evolução diferencial paralelo aplicado ao problema da predição da estrutura de proteínas utilizando o modelo ab em 2d e 3d. *Dissertação de Mestrado da Universidade Federal Tecnológica do Paraná. Orientador: H.S. Lopes*, 2015. Citado 2 vezes nas páginas 22 e 24.

KALOGERAKOS M. GOURMA, J. S. Use of openmp to parallelise a onedimensional multiphase flow code. *Proceedings of the International Conference on Numerical Analysis and Applied Mathematics*, China, 2014. Citado na página 45.

KIRK, D. B.; WEN-MEI, W. H. *Programming massively parallel processors: a hands-on approach*. [S.l.]: Morgan kaufmann, 2016. Citado na página 37.

KOREJO, I. A. et al. Genetic algorithm using an adaptive mutation operator for numerical optimization functions. *Sindth University Res. Journal (Sci.Ser)*, n. 1, 2013. Citado na página 22.

KOUPAEI, J. A.; HOSSEINI, S.; GHAINI, F. M. A new optimization algorithm based on chaotic maps and golden section search method. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 50, p. 201–214, 2016. Citado na página 22.

KUMAR, C.; ALWARSAMY, T. Dynamic economic dispatch – a review of solution methodologies. *European Journal of Scientific Research*, n. 4, Novembro 2011. Citado na página 60.

LEHRER, C.; BORGES, P. S. S. Algoritmos genÉticos com operaÇÃo de seleÇÃo hawk-dove com estratÉgia tit for tat, e com operação de recombinaÇÃo com taxas variáveis. *XXII Encontro Nacional de Engenharia de Produção*, Curitiba - PR, Outubro 2002. Citado na página 29.

LIN, D.-Y.; KU, Y.-H. Using genetic algorithms to optimize stopping patterns for passenger rail transportation. *Computer-Aided Civil and Infrastructure Engineering*, v. 29, n. 4, p. 264–278, 2014. ISSN 1467-8667. Citado na página 21.

LINDEN, R. Algoritmos genéticos na indústria do petróleo: Uma visão geral. *Revista da Engenharia de Instalações no Mar.*, n. 1, Jan/Jul 2008. Citado na página 21.

LINDEN, R. *Algoritmos Genéticos*. Terceira edição. Rio de Janeiro: Editora Ciência Moderna, 2012. Citado 2 vezes nas páginas 28 e 29.

LU, H. et al. A selfadaptive genetic algorithm to estimate ja model parameters considering minor loops. *Journal of Magnetism and Magnetic Materials*, 2015. Citado na página 22.

MEHTA, K.; GABRIEL, E. Multi-threaded parallel i/o for openmp applications. *Springer Science+Business Media*, New York, 2014. Citado na página 22.

MICHALEWICZ, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. [S.l.]: Springer, 1996. Citado na página 28.

MICHALEWICZ, Z.; JANIKOW, C. Z. Handling constraints in genetic algorithms. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1993. Citado na página 34.

MICHALEWICZ, Z.; NAZHIYATH, G. G. genocop iii: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, 1995. Citado na página 34.

MIRANDA, R. C.; MONTEVECHI, J. A. B.; PINHO, A. F. de. Development of na adaptive genetic algorithm for simulation optimization. *Acta Scientiarum: Technology*, n. 37, Jul/Set 2015. Citado na página 22.

MIRJALILI, S. Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. *Neural Computing and Applications*, Springer, v. 27, n. 4, p. 1053–1073, 2016. Citado na página 22.

MOLDOVAN, D. I. *Parallel processing from applications to systems*. [S.l.]: Elsevier, 2014. Citado na página 37.

MONTESINOS, F. G.; ARNOSO, J.; VIEIRA, R. Using a genetic algorithm for 3-d inversion of gravity data in fuerteventura (canary islands). *Int J Earth Sci (Geol Rundsch)*, n. 94, 2005. Citado na página 21.

MUDDUKRISHNA, A.; P.JONSSON; BRORSSON, M. Characterizing task-based openmp programs. *PLOS ONE Journal*, Sweden, 2015. Citado na página 42.

NAVAUX, P.; ROSE, C. D.; PILLA, L. Fundamentos das arquiteturas para processamento paralelo e distribuído. In: _____. *Programação Paralela Baseada em Tarefas em Arquiteturas com Memória Distribuída*. 2. ed. [S.l.]: ERAD, 2011. cap. 2. Citado na página 40.

NIEBERG, S. M.; BEYER, H. Self-adaptation in evolutionary algorithms. In: _____. *Parameter Setting in Evolutionary Algorithms*. [S.l.]: Springer, 2007. Citado na página 49.

NIKNAM, T.; GOLESTANEH, F. Enhanced adaptive particle swarm optimisation algorithm for dynamic economic dispatch of units considering valve-point effects and ramp rates. *IET Generation, Transmission Distribution*, v. 6, n. 5, p. 424–435, May 2012. ISSN 1751-8687. Citado 6 vezes nas páginas 23, 32, 57, 71, 74 e 77.

OLIVEIRA, L. W.; OLIVEIRA, A. R.; GOMES, F. Aplicação de algoritmos genéticos para o planejamento de geração distribuída em sistemas de distribuição. *Revista Científica Interdisciplinar*, n. 1, Janeiro/Março 2016. Citado na página 21.

OPENMP, A. *OpenMP application program interface*. 2013. Disponível em: <www.openmp.org>. Citado na página 41.

PATEL, C.; GULATI, R. Software performance analysis: A data mining approach. *International Journal of Information Technology and Management Information System (IJITMIS)*, v. 5, p. 28–31, Maio - Agosto 2014. ISSN 0976–6413. Citado na página 24.

PESSINI, E. C. Algoritmos genÉticos paralelos – uma implementaÇÃo distribuÍda baseada em javaspaces. *Dissertação de Mestrado da Universidade Federal de Santa Catarina. Orientador: J.Mazzucco Jr.*, 2003. Citado 2 vezes nas páginas 22 e 24.

SCHOENAUER, M.; MICHALEWICZ, Z. Evolutionary computation at the edge of feasibility. *Proceedings of the Fourth International Conference on Parallel Problem Solving from Nature*, 1996. Citado na página 34.

SERPELL, M.; SMITH, J. Self-adaptation of mutation operator and probability for permutation representations in genetic algorithms. *Evolutionary Computation*, n. 3, 2010. Citado na página 22.

SHANG, Y. W.; QIU, Y. H. A note on the extended rosenbrock function. *Evolutionary Computation*, v. 14, n. 1, p. 119–126, March 2006. ISSN 1063-6560. Citado na página 59.

SILVA, F. J. M. da; OLIVEIRA, A. C. de; VERAS, R. de M. S. Um algoritmo genético paralelo aplicado ao problema de cobertura de conjuntos. *SBSI UFMG*, 2013. Citado 2 vezes nas páginas 22 e 24.

SRIYANYONG, P.; LU, H. Implementation and comparison of pso-based algorithms for multi-modal optimization problems. *International Symposium on Computational Models for Life Sciences*, Novembro 2013. Citado na página 60.

TRIVEDI, I. N. et al. Adaptive krill herd algorithm for global numerical optimization. In: *Advances in Computer and Computational Sciences*. [S.l.]: Springer, 2017. p. 517–525. Citado na página 22.

TSOULOS, I. G.; TZALLAS, A.; TSALIKAKIS, D. Pdoublepop: An implementation of parallel genetic algorithm for function optimization. *Computer Physics Communications*, Elsevier, v. 209, p. 183–189, 2016. Citado na página 22.

XIAO, W. et al. Applying a new adaptive genetic algorithm to study the layout of drilling equipment in semisubmersible drilling platforms. *Mathematical Problems in Engineering.*, Jul/Set 2015. Citado na página 22.

XU, X. et al. A second-order smooth penalty function algorithm for constrained optimization problems. *Computational Optimization and Applications*, v. 55, Maio 2013. Citado na página 33.

YADAV, A.; YADAV, N.; KIM, J. H. A comparative study of exploration ability of harmony search algorithms. In: SPRINGER. *International Conference on Harmony Search Algorithm*. [S.l.], 2017. p. 22–31. Citado na página 22.

YENIAY, O. Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and Computational Applications*, Beytepe, Ankara, v. 10, 2005. Citado na página 33.