EDNALDO OLIVEIRA LIMA

ALGORITMO GENÉTICO HÍBRIDO APLICADO À OTIMIZAÇÃO DE FUNÇÕES.

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

LAVRAS
MINAS GERAIS - BRASIL
2008

EDNALDO OLIVEIRA LIMA

ALGORITMO GENÉTICO HÍBRIDO APLICADO À OTIMIZAÇÃO DE FUNÇÕES.

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Área de Concentração:
Otimização

Orientador:
Cláudio Fabiano Motta Toledo

LAVRAS
MINAS GERAIS - BRASIL
2008

Ficha Catalográfica Preparada pela Divisão de Processos Técnicos da Biblioteca Central da UFLA

Lima, Ednaldo Oliveira

Algoritmo Genético Híbrido aplicado à otimização de funções /Ednaldo Oliveira Lima. Lavras – Minas Gerais, 2008. p: 73.

Monografia de Graduação - Universidade Federal de Lavras. Departamento de Ciência da Computação.

1. Otimização Combinatória. 2.Computação Evolutiva. 3. Algoritmos Genéticos. I. LIMA, E.O. II. Universidade Federal de Lavras. III. Algoritmo Genético Híbrido aplicado à otimização de funções.

EDNALDO OLIVEIRA LIMA

ALGORITMO GENÉTICO HÍBRIDO APLICADO À OTIMIZAÇÃO DE FUNÇÕES.

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 20 de Julho de 2008

Marluce Rodrigues Pereira

Ahmed Ali Abdalla Esmin

Cláudio Fabiano Motta Toledo

LAVRAS
MINAS GERAIS – BRASIL
2008

Agradecimentos

Primeiramente agradeço à Deus e a minha família que sempre me apoiaram, meus amigos em especial o Lucas que também me ajudou nessa pesquisa ao lado do Cláudio que foi um excelente orientador.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AG – Algoritmo Genético

AGH – Algoritmo Genético Híbrido

CE – Computação Evolutiva

DNA – Ácido desoxinorribunucléico

EE – Estratégia Evolutiva

IA – Inteligência Artificial

MA – *Memetic Algoritms* Algoritmo Memético

PE - Programação Evolutiva

PPSN - Parallel Problem Solving from Nature

ALGORITMO GENÉTICO HÍBRIDO APLICADO À OTIMIZAÇÃO DE FUNÇÕES.

RESUMO

A presente monografia propõe um algoritmo genético híbrido (AGH) e avalia seu desempenho na otimização de funções. O AGH utiliza codificação binária para indivíduos, onde cadeias binárias de diferentes tamanhos são testadas. Os indivíduos em uma população estão estruturados de forma hierárquica em árvores binária ou ternária. Uma busca local é executada sobre o melhor indivíduo de cada população. O uso de multi-populações e diversos tipos de crossover também são avaliados. As abordagens propostas para o AGH são aplicadas na otimização de sete funções uni e multi-modais. Os melhores resultados obtidos pelo AGH são comparados aos resultados existentes na literatura.

HYBRID GENETIC ALGORITHM APPLIED TO FUNCTIONS OPTMIZATION.

ABSTRACT

The present work proposes a hybrid genetic algorithm (HGA) and evaluates its performance in functions optimization. The HGA uses a binary codification for individuals, where different strings lengths are tested. The individuals in each population are hierarchically structured in binary or ternary trees. A local search is executed over the best individual found in each population. The use of multi-population and several crossover types are also evaluated. The approaches proposed for the HGA are applied to optimize seven minimum and multiminima functions. The best results found by HGA are compared with results available in the literature.

1. INTRODUÇÃO

1.1. Contextualização e Motivação

A presente monografia pode ser contextualizada como um trabalho de pesquisa dentro da área de computação evolutiva, mais especificamente relacionada à proposta de novas abordagens para algoritmos genéticos. A motivação do trabalho está no uso de algoritmos genéticos multi-populacionais aplicados à otimização de funções.

Algoritmos Genéticos (AG) integram uma área jovem de pesquisa conhecida como computação evolutiva. Segundo Castro & Von Zuben (2008), a denominação computação evolutiva foi proposta em 1990 durante o encontro de pesquisadores em algoritmos evolutivos na conferência *Parallel Problem Solving from Nature* (PPSN), realizada em Dortmund. Todavia, seu contexto vem sendo utilizado de forma multidisciplinar, ou seja, desde ciências naturais e engenharia até biologia e ciência da computação. A idéia fundamental, despontada nos anos cinqüenta, é empregar o processo de evolução natural como um modelo de solução de problemas, a partir de sua implementação em computador.

Segundo Von Zuben (2007), a vantagem mais significativa na computação evolutiva está na possibilidade de resolver problemas pela simples descrição do que se quer ver presente na solução, não havendo necessidade de se indicar explicitamente os passos até o resultado, que certamente seriam específicos para cada caso. É lógico que algoritmos evolutivos correspondem a uma seqüência de passos até a solução, mas estes passos são os mesmos para uma ampla gama de problemas, fornecendo robustez e flexibilidade na aplicação da abordagem evolutiva. Sendo assim, a computação evolutiva deve ser entendida como um conjunto de técnicas e procedimentos genéricos e adaptáveis, aplicados à solução de problemas complexos para os quais outras técnicas conhecidas são ineficazes ou nem sequer são aplicáveis.

Em termos históricos, três algoritmos para a computação evolutiva foram desenvolvidos independentemente:

- Algoritmos Genéticos: Holland (1962), Bremermann (1962) e Fraser (1957);
- Programação evolutiva: Fogel (1962).
- Estratégias evolutivas: Rechenberg (1965) e Schwefel (1965).

Os Algoritmos Genéticos (AGs), apresentados inicialmente com o nome de Planos Evolutivos, são uma classe de algoritmos que utilizam técnicas inspiradas na biologia

evolutiva como hereditariedade, mutação, seleção e recombinação. AGs se diferenciam de métodos tradicionais de otimização por se basearem em uma população de possíveis soluções e não apenas na otimização de parâmetros do problema. Além disso, AGs podem ser implementados sem a necessidade de conhecimento prévio do problema. Necessita-se apenas de uma forma adequada de representação e avaliação das possíveis soluções do problema a ser tratado.

Programação Evolutiva (PE), apresentada por Fogel et. al (1966), foi proposta como uma técnica de simulação de evolução que enfatiza a mutação para desenvolver uma forma diferenciada de Inteligência Artificial (IA). Inicialmente foi proposta para evoluir máquinas de estado finito, porém, o problema evolutivo foi definido como sendo a evolução de um algoritmo que pudesse operar na seqüência de símbolos. Segundo Koza (1992), na Programação evolutiva os indivíduos representam pequenos programas de computador que serão avaliados de acordo com o resultado de sua execução. Estes programas podem ser expressões simples, como fórmulas aritméticas ou programas complexos, com operações de laço e condicionais, típicas de uma linguagem de programação comum.

Estratégias Evolutivas (EE), desenvolvidas por Rechenberg (1973) e Schweffel (1975, 1977), são definidas da seguinte forma por Castro & Von Zuben (2008):

> *"Estratégias Evolutivas: utilizam mutações com distribuição normal para modificar vetores reais e enfatizam a mutação e recombinação como operadores essenciais ao processo de busca no espaço de busca e no espaço de parâmetros. O operador de seleção é determinístico e o tamanho da população de pais e de filhos pode ser distinto."*

Considerando o contexto apresentado, o presente trabalho estará focado em avaliar estratégias para Algoritmos Genéticos (AGs) voltadas à otimização de funções. Uma abordagem multi-populacional com uma estrutura hierarquizada para indivíduos em cada população será proposta.

1.2. Objetivos e estrutura do trabalho

A presente monografia propõe o uso de algoritmos genéticos com uma abordagem multi-populacional para avaliar diversos tipos de funções. A abordagem multi-

populacional foi adotada já que, segundo Mendes (2003), ocorre uma falta de consistência ao executarmos o AG uni-populacional por várias vezes, causada em parte pelo seu forte embasamento aleatório. Na abordagem multi-populacional há populações evoluindo em paralelo e seguindo caminhos evolutivos diferentes. Assim, um espaço de soluções maior acaba sendo explorado.

As funções avaliadas neste trabalho são comumente utilizadas na validação de outras abordagens envolvendo algoritmos genéticos. Dessa forma, os resultados obtidos por este trabalho poderão ser comparados a resultados já existentes e disponíveis na literatura.

Desta forma, os principais objetivos deste trabalho são:

- Propor um Algoritmo Genético multi-populacional aplicável a um conjunto de funções;
- Avaliar sua aplicação na otimização dessas funções;
- Comparar os resultados obtidos de acordo com a estrutura populacional utilizada;
- Comparar os resultados obtidos com outros existentes na literatura.

O trabalho se estrutura da seguinte maneira: o capítulo dois engloba o referencial teórico, ou seja, uma apresentação dos principais conceitos que envolvem o trabalho e estão relacionados ao assunto. O capítulo três contém a metodologia, apresentando os conceitos que levaram ao método desenvolvido bem como os atributos do método proposto nessa pesquisa. O capítulo quatro apresenta os resultados computacionais e o capítulo cinco a conclusão do trabalho.

2. Referencial Teórico

O presente capítulo apresenta os conceitos e a respectiva literatura relacionados ao tema desta monografia. Inicialmente são apresentadas as relações com a biologia que levaram ao estabelecimento de conceitos biológicos utilizados pelos algoritmos genéticos. Em seguida, os conceitos e as literaturas relacionados ao algoritmo genético são estabelecidos.

2.1 Relação com a Biologia

A Computação Evolutiva (CE) introduz uma importante ferramenta de resolução de problemas, os algoritmos genéticos, que são parte de uma área emergente da computação inteligente denominada Computação Bio-Insprirada (LINDEN, 2006). Nesta área, métodos para soluções de problemas computacionais difíceis são desenvolvidos inspirados por processos evolutivos encontrados na natureza. Assim, esta seção vai abordar o contexto histórico e biológico da teoria da evolução.

2.1.1 Hereditariedade

Segundo Von Zuben (2007), até o século XIX os cientistas acreditavam em teorias como a do criacionismo, onde Deus criou o universo assim como ele é, ou da geração espontânea em que a vida surge de essências presentes no ar. Durante dois mil anos acreditou-se que o sexo do filho era determinado pela procedência do líquido seminal do homem, onde se o sêmen tivesse origem no testículo direito o filho seria do sexo masculino e caso se originasse do testículo esquerdo seria do sexo feminino. Essa crença teve fim em 1672 com a descoberta do óvulo feita pelo holandês Graaf. Essa descoberta revelou a importância do papel das mulheres na reprodução. Em 1675 o também holandês Von Leeuwenhoeck descobriu o espermatozóide e com isso desenvolveram uma teoria absurda chamada de homúnculo. O espermatozóide já seria o feto em proporções menores que na barriga da mãe, mais especificamente no óvulo, ganharia somente o tamanho ideal para nascer. Em 1775 Spallanzani provou que era necessário um espermatozóide e um óvulo

para haver reprodução humana na qual o esperma era o fator fecundante, deitando por terra as teorias do homúnculo.

As primeiras idéias que surgiram efetivamente acerca de hereditariedade foram propostas por Gregor Mendel em 1986, um monge agostiniano. Ele atacou o problema de modo simplificado e lógico: concentrou-se em características constantes, desenvolveu um programa de cruzamentos controlados, tratou os resultados de forma eficiente e percebeu a existência dos hoje conhecidos genes. Ficou faltando somente esclarecer os mecanismos celulares envolvidos. Curiosamente um dos fatores que dificultaram a compreensão dos resultados de Mendel por 30 anos foi o debate científico causado em 1859 pelo livro "A Origem das espécies" de Charles Darwin (naturalista inglês, 1809-1882) baseado em observações realizadas durante uma longa viagem por vários lugares a bordo do navio HMS Beagle. Darwin pôde perceber as diferenças entre animais de mesma espécie em ecossistemas diversificados e suas adaptações referentes ao ambiente, culminando na teoria da seleção natural.

2.1.2 Seleção Natural

O princípio da seleção natural indica que os indivíduos cujas variações se adaptam melhor ao ambiente terão maior probabilidade de sobreviver e se reproduzir. A evolução darwiniana é nada mais que a conseqüência inevitável da competição entre sistemas de reprodução de informação, operando no interior de uma arena finita em um universo com diferencial de entropia positivo (ATAMAR, 1992).

Este processo implica nos descendentes dos indivíduos serem variações de seus pais. Assim, um descendente é ligeiramente diferente de seu pai, podendo esta diferença ser positiva ou negativa mostrando que não se trata de um processo dirigido, com intuito de maximizar algumas características da espécie. Na verdade, não existe nenhuma garantia de que os descendentes de pais muito bem adaptados também o sejam (LINDEN, 2006).

Segundo Von Zuben (2007), desde Darwin, a teoria da evolução unificou diversas áreas da biologia, pois a seleção natural foi a força propulsora que distingue os sistemas biológicos dos demais sistemas químicos e físicos. A seleção natural aponta os tipos de variação, que devem conduzir os organismos a uma melhor adaptação ao meio. Até que em 1893 surge a grande contribuição de Weismann para a teoria das leis de seleção natural, onde foi provado que uma outra teoria, a teoria de Jean Baptiste de Lamarck, (naturalista

francês, 1744-1829) aceita até mesmo por Darwin, estava errada. A contribuição de Weismann é de que seria impossível transmitir as informações adquiridas pelas células somáticas a seus descendentes, ou seja, a informação pode passar de DNA para DNA ou de DNA para proteína, mas não pode transmitir de proteína para DNA.

A existência de mudanças acidentais e não adaptativas e os resultados do trabalho de Weismann fundamentaram a base do Neodarwinismo. Para o Neodarwinismo, é necessário acrescentar algumas hipóteses ligadas à genética. Essas hipóteses são:

- Algum processo de variação continuada deve ser responsável pela introdução de novas informações junto à carga genética dos organismos;
- Não há limite para a sucessão de variações que podem ocorrer;
- A seleção natural é o mecanismo para preservação das novas informações que correspondam a uma maior adaptação.

Logo, a seleção natural é probabilística, e seu alvo primário é o indivíduo, embora seu efeito resultante vá se manifestar na espécie como um todo. A espécie é o beneficiário final do processo evolutivo (MAYR, 1988).

2.1.3. Genótipo e Fenótipo

Um conjunto específico de genes é chamado de genótipo. O genótipo é base do fenótipo, que é a expressão das características físicas e mentais codificadas pelos genes e modificadas pelo ambiente, tais como cor dos olhos, inteligência, etc. Nosso DNA codifica toda a informação necessária para nos descrever, mas essa informação está sob o controle de uma grande rede de regulação gênica que, associada às condições ambientais, gera as proteínas na quantidade certa, que farão de nós tudo aquilo que efetivamente somos (Linden, 2006).

Segundo Fogel (1994) existe uma dualidade entre seu código genético (genótipo) e suas características comportamentais, fisiológicas e morfológicas (fenótipo) envolvendo indivíduos e espécies. De acordo com Von Zuben (2007), não existe uma relação biunívoca entre um gene (elemento do genótipo) e uma característica (elemento do fenótipo) em sistemas evoluídos naturalmente. Um único gene pode afetar diversos traços fenotípicos simultaneamente (pleiotropia) e uma única característica fenotípica pode ser determinada pela interação de vários genes (poligenia). Os efeitos de pleiotropia e

poligenia geralmente tornam os resultados de variações genéticas imprevisíveis. Para Hartl & Clark (1989), sistemas naturais em evolução são fortemente pleiotrópicos e altamente poligênicos. Não ocorrendo o mesmo em sistemas artificiais por elevar consideravelmente o custo computacional do sistema e fazendo com que haja uma relação de um-pra-um entre genótipo e fenótipo.

2.1.4. Outros Conceitos Biológicos Relevantes

Alguns outros conceitos genéticos também são utilizados por diversos algoritmos evolutivos:

- Cromossomo é uma estrutura formada por uma cadeia de DNA sendo a base física dos genes.
- *Crossover* é uma recombinação que consiste numa troca de material genético ocasionada por um evento aleatório entre dois cromossomos.
- Genes são blocos funcionais de DNA e cada um deles localiza-se em uma posição particular do cromossomo.
- Mutação é a troca, também aleatória, de posição entre genes de um cromossomo.
- Genoma é um conjunto de material genético que compõe um organismo.

2.2. Algoritmos Genéticos

Segundo Diagalakis *et tal.* (2000), Algoritmos Genéticos (AG) são uma classe de algoritmos probabilísticos que se baseiam na evolução biológica. Conforme Toledo (2005), AG simula processos biológicos de recombinação genética, mutação e seleção gerando soluções para um dado problema.

O primeiro trabalho sobre AG foi apresentado por Holland (1975) e um aprofundamento sobre conceitos e técnicas envolvendo AG podem ser encontrados nos trabalhos de Goldberg (1989), Davis (1990), Michalewicz (1996), Bäck *et al.* (2000a) e Bäck *et al.* (2000b). Ainda hoje a maior parte da teoria existente sobre algoritmos genéticos aplica-se ao modelo introduzido por Holland, conhecido como algoritmo

genético canônico. Avanços recentes na teoria da modelagem do algoritmo genético também se aplicam primariamente ao algoritmo genético canônico (VOSE, 1993).

Apesar de Holland não ser o primeiro a investigar na área, ele é considerado o pai dos algoritmos genéticos. Ao contrário do que a maioria pensa, em seu livro “Adaptation in Natural and Artificial Systems”, o AG é apresentado como uma metáfora para os processos evolutivos, de forma que ele pudesse simular em computadores a adaptação e a evolução no mundo real. Entretanto, os AG transcenderam a esse papel e se transformam em uma ferramenta para os cientistas (LINDEN, 2006).

A forma usual dos algoritmos genéticos foi descrita por Goldberg (1989) como técnicas de busca estocástica baseadas no mecanismo de seleção natural e genética natural. Segundo Gen e Cheng (1997), os AGs se diferem das técnicas convencionais de pesquisa, ao utilizarem um conjunto inicial randômico de soluções chamado *população*. Cada indivíduo na população é chamado de *cromossomo*, uma possível representação da solução para o problema em mãos. Um cromossomo é uma seqüência de símbolos e normalmente, mas não necessariamente, uma cadeia binária. Os cromossomos evoluem através de sucessivas iterações, chamada geração. Durante cada geração, os cromossomos são mensurados através de algumas medidas de avaliação também chamadas de *fitness*. Para criar a próxima geração, o novo cromossomo, chamado de filho, é formado utilizando (a) um operador de *crossover* que mescla dois cromossomos da geração atual, (b) modificando o cromossomo utilizando o operador de *mutação*. Uma nova geração é então formada selecionando, de acordo com a avaliação dos cromossomos, alguns dos pais e descendentes gerados; (b) rejeitando outros cromossomos, de modo a manter as populações com tamanhos constantes. Depois de várias gerações, o algoritmo converge para o melhor cromossomo, o qual se espera que represente uma solução ótima ou próxima ao valor ótimo para o problema. Sejam P(t) e C(t) pais e filhos respectivamente, na atual geração *t*. A estrutura geral dos algoritmos genéticos descrita até o momento é sumarizada no pseudocódigo abaixo:

```
Procedimento: Algoritmo Genético
Inicio
        t = 0;
        inicialize P(t);
        avaliar P(t);
        enquanto (não terminar a condição) faça
                recombinar P(t) para rendimento C(t);
                avaliar C(t);
                selecionar P(t+1) de P(t) e C(t);
                t = t+1;
        fim_enquanto
fim
```

Figura 2.1 Pseudocódigo AG

2.2.1. Representação de Indivíduos

Um indivíduo da população é representado por um único cromossomo, o qual contém a codificação (genótipo) de uma possível solução do problema (fenótipo), ou seja, um indivíduo é um cromossomo e sua aptidão (VON ZUBEN, 2007).

Cromossomos são usualmente codificados na forma de listas de atributos ou vetores, onde cada atributo é conhecido como gene. Os possíveis valores que um determinado gene pode assumir são denominados alelos. Cada indivíduo de uma população representa um candidato em potencial à solução do problema em questão.

A representação do cromossomo consiste em traduzir a informação do problema estudado de uma maneira viável a ser tratada pelo computador. Ela deve ser o mais simples possível, onde restrições impostas pelo problema devam estar implícitas dentro da representação adotada. A representação pode ser feita a partir de três abordagens distintas que são utilizadas para realizar o processo de codificação dos AG. A codificação clássica (ou binária) que trabalha com cadeias (*strings*) de bits, a codificação real que trabalha diretamente com valores reais e a codificação inteira que trabalha com valores inteiros.

A motivação para o uso de codificação binária vem da teoria dos esquemas (*schemata theory*), utilizada com relativo sucesso para explicar por que os algoritmos genéticos funcionam. Holland (1992) argumenta que seria benéfico para o desempenho do algoritmo maximizar o paralelismo implícito inerente ao algoritmo genético. Ele prova que

um alfabeto binário maximiza o paralelismo implícito. Entretanto, em diversas aplicações práticas a utilização de codificação binária leva a um desempenho insatisfatório. Em problemas de otimização numérica com parâmetros reais, algoritmos genéticos com representação em ponto flutuante freqüentemente apresentam desempenho superior à codificação binária. Isto ocorre porque, em alguns casos, seriam necessários cromossomos binários muito grandes para transcrever as informações e a precisão desejada para o problema.

Fogel (1994), entretanto, aponta que a maximização do paralelismo implícito nem sempre produz um desempenho ótimo. Segundo Von Zuben (2007) a codificação é uma das etapas mais críticas na definição de um algoritmo genético. A definição inadequada da codificação pode levar a problemas de convergência prematura do AG.

2.2.1.1. Representação Binária

Para representarmos números reais como números binários, precisamos saber a faixa de operação de cada uma das variáveis e a precisão desejada para que possamos definir a quantidade de bits necessária. Se usarmos *k* bits para uma variável x_i, que trabalha numa faixa $[inf_i, sup_i]$ então estamos definindo que a precisão máxima dessa variável é de:

$$\frac{sup_i - inf_i}{2^K - 1} \qquad (1)$$

Para converter o número binário para o número real dentro da faixa, temos que obter o número inteiro r_i correspondente ao número binário e depois fazer a seguinte operação:

$$real = inf_i + \left(\frac{sup_i - inf_i}{2^k - 1} \right) * r_i \qquad (2)$$

Se quiser representar mais que um número real em nosso cromossomo binário, basta posicionar os mesmos lado a lado, como uma concatenação de string onde os k_1 primeiros bits representam x_1, os k_2 primeiros x_2 e assim sucessivamente. Os tamanhos de k_1, $k_{2,...}$ k_n não são necessariamente os mesmos pois podemos ter variáveis com precisão diferentes, assim como ilustra a figura .2 - a.

Por exemplo, supondo a faixa de x_1 de [-2,2] e x_2 de [0,1] temos que o cromossomo acima codificaria:

$$r_1 = 000011 = 3$$

$$r_2 = 110011 = 51$$

Portanto:

$$x_1 = -2 + 3 * \left(\frac{2 - (-2)}{2^6 - 1} \right) = -1{,}809$$

$$x_2 = 0 + 51 * \left(\frac{1 - 0}{2^6 - 1} \right) = 0{,}809$$

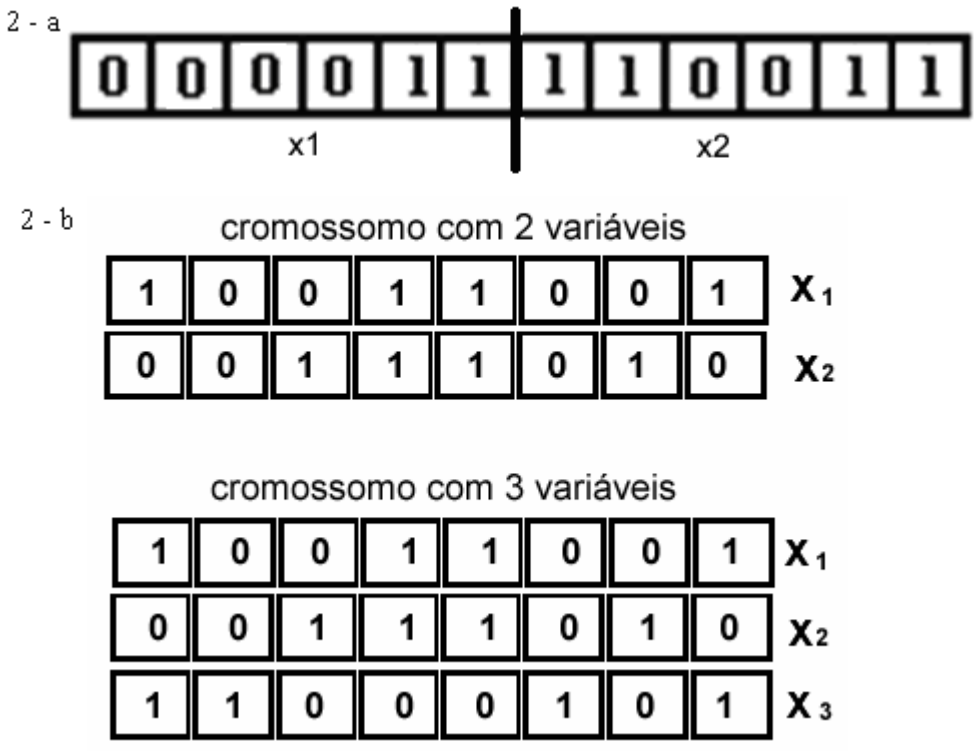

Figura 2.2 – Cromossomo Binário

Como mostra a figura.2 – b, podemos representar mais variáveis de uma outra forma. Ao invés de utilizarmos um vetor, utilizamos uma matriz *n* x *m*, onde *n* representa o número de linhas, ou seja, o número de variáveis e *m* representa o número de colunas, o

tamanho da cadeia do cromossomo. Sendo essa maneira de representação a adotada em nosso trabalho.

2.2.1.2 Representação Real

A representação real utiliza cromossomos mais compactos e o tempo de processamento tende a ser menor. A codificação binária pode gerar cromossomos muito grandes, onde a aplicação dos operadores genéticos pode demandar um elevado tempo computacional. Na codificação real não há necessidade de conversões para avaliar a função objetivo, pois cada gene corresponde a uma variável.

A utilização da codificação real permite um maior controle em relação à ação dos operadores genéticos, pois cada gene representa uma variável. No caso da codificação binária, a aplicação dos operadores genéticos produz modificações nos fenótipos que são difíceis de serem previstas. O limite de precisão da solução obtida em codificação real é o da precisão da máquina. Em codificação binária este limite é baseado no número de genes (bits) utilizados na representação das variáveis.

O cromossomo seria representado por um vetor de pontos flutuantes.

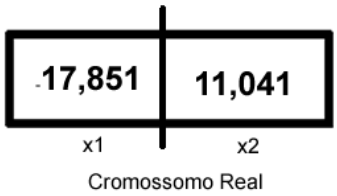

Figura 2.3 – Cromossomo Real

2.2.2. Função de Avaliação

O *fitness* é uma função de avaliação, também conhecida como função de custo, que determina a qualidade do indivíduo (cromossomo) enquanto solução para o problema. Essa função é uma parte não genérica dos AG que deve ser capaz de identificar todas as restrições e objetivos, ou seja, ela é especifica para cada problema.

A função de avaliação e a função de aptidão devem ser bem distinguidas. A função de avaliação ou função objetivo fornece uma medida de desempenho com respeito a um

conjunto particular de parâmetros. É a função de avaliação (*fitness*) que transforma esta medida em alocação de oportunidades reprodutivas. Assim, a avaliação de uma cadeia binária, representando um conjunto particular de parâmetros, é independente da avaliação de qualquer outra cadeia.

Já a aptidão é definida de acordo com outros membros da atual população em um algoritmo genético. Assim, a aptidão é definida como *fi*/*f* onde *fi* é a avaliação associada ao indivíduo *i* e *f* é a avaliação média de todos os indivíduos na população. A aptidão pode também ser associada à classificação de um indivíduo ou outras medidas como seleção por torneio (GOLDBERG, 1990). A função aptidão pode ser igual à função objetivo, ou resultado do escalonamento da função objetivo, ou baseada no *ranking* do indivíduo da população, ou ainda através de normalização linear (GOLDBERG, 1989).

2.2.3. Operadores Genéticos

Operadores genéticos são mecanismos que buscam perturbar o sistema, ao exemplo do que ocorre na natureza, e possuem a função de busca nos AGs por regiões desconhecidas do espaço de soluções. Esses operadores são o *crossover* (recombinação) e a mutação. Tais operadores ocorrem no algoritmo genético segundo taxas de probabilidade percentual de ocorrência, onde a taxa de *crossover* é normalmente mais alta que a de mutação.

O *crossover* busca, através de recombinação das informações codificadas no indivíduo, encontrar novas e, se possíveis, melhores soluções. De acordo com Bessaou e Siarry (2001), o *crossover* nos permite criar novos indivíduos (filhos) a partir daqueles já existentes (pais), onde filhos podem herdar de seus pais os melhores atributos. Todavia ele faz com que o algoritmo tenda à convergência devido à homogeneização da população. Existem diversos tipos de *crossover* que costumam ser definidos de acordo com a codificação do cromossomo.

Na Figura 2.4, é apresentado o *crossover* de um ponto para indivíduo com codificação binária. Trata-se de uma recombinação entre dois cromossomos pais que trocam partes de sua cadeia binária a partir de um ponto aleatório de corte.

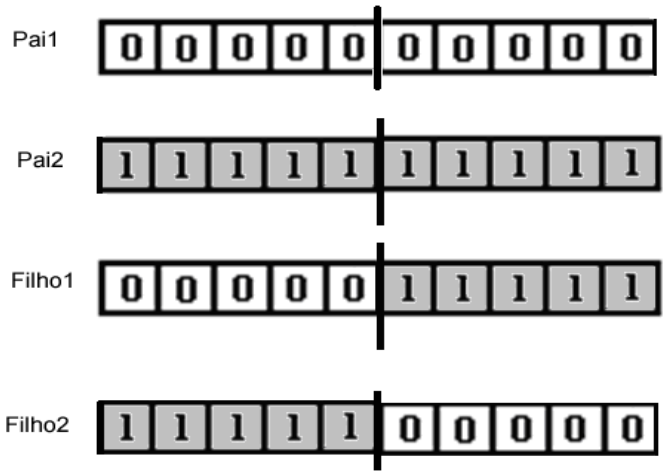

Figura – 2.4 *Crossover* de um ponto

Neste caso, o ponto de corte é cinco, então os cinco bits iniciais do Pai 1 foram mantidos no Filho 1 e sua cadeia é completada com os cinco bits finais de Pai 2. A cadeia binária do Filho 2 é composta pelos cinco primeiros bits de Pai 2 mais os cinco últimos de Pai 1.

O *crossover* de dois pontos tem como característica preservar mais esquemas. Por exemplo, na Figura 2.5, o *crossover* de dois pontos mantém o esquema 00***000 do Pai1 e 11***111 do Pai2.

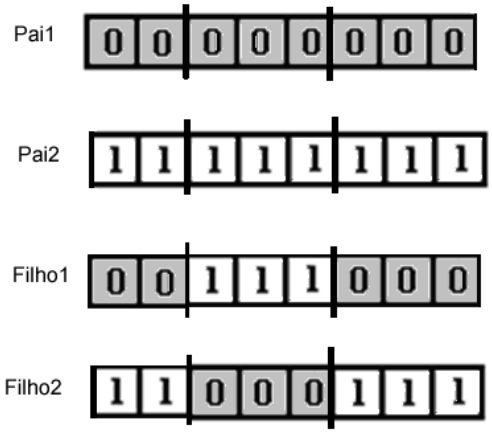

Figura – 2.5 *Crossover* de dois pontos

Este *crossover* seleciona dois pontos aleatórios de corte, nesse caso o segundo e o quinto bit. O Filho1 recebe o conteúdo entre os dois cortes do Pai2 e o Filho2 recebe o mesmo conteúdo referente ao Pai1, ou seja, ocorre troca somente na parte intermediária da

cadeia. Esse processo de corte e troca entre as partes das cadeias pode ser repetido com *N* pontos de cortes, sendo o de dois pontos o mais eficiente na maioria dos casos.

O *crossover* uniforme é capaz de combinar todo e qualquer esquema existente. (LINDEN, 2006). Este *crossover* possui uma grande possibilidade de alterar esquemas e seu desempenho costuma ser superior aos anteriores. Seu funcionamento é ilustrado pela 2.7.

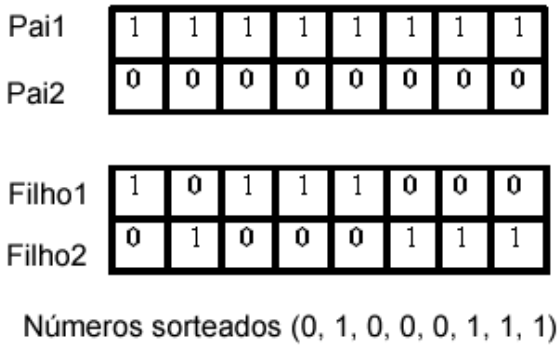

Figura – 2.6 *Crossover* Uniforme

O *crossover* uniforme sorteia para cada posição da cadeia, um número 1 ou 0. Se o número sorteado for 0, o Filho1 recebe o mesmo bit do Pai1 e o Filho2 recebe do Pai2. Senão, ocorrerá a atribuição contrária.

O *crossover* de maioria, Figura 2.7, analisa a maioria dos bits até o momento em determinada posição da cadeia e gera um filho com este bit.

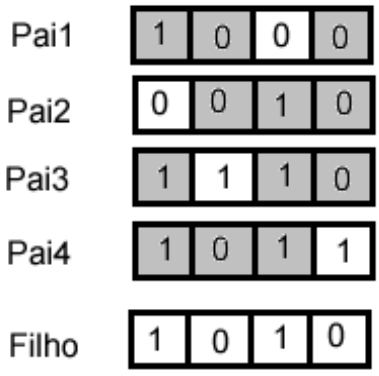

Figura 2.7 - *Crossover* maioria

O crossover de maioria é alternativo, porém costuma não ser tão eficiente quanto os demais já que tem uma forte tendência a convergência.

Os métodos de recombinação utilizados em codificação real são mais diversificados e ocorrem através de operações matemáticas entre os pais. Apenas para exemplificar, temos métodos como o de média aritmética e da média geométrica:

$$\text{Filho1} = \frac{\text{Pai1} + \text{Pai2}}{2} \qquad (3)$$

$$\text{Filho1} = \sqrt{Pai1 \quad * Pai2} \qquad (4)$$

Neste caso, cada posição na cadeia real de um filho é fornecida como resultado da operação matemática executada sobre as respectivas posições dos pais.

Após a recombinação, podemos aplicar um operador genético de mutação que altera algumas informações codificadas no indivíduo, restaurando assim uma saudável diversidade na população. Isso faz com que a população não fique presa em determinados esquemas que podem ser pontos de mínimos ou máximos locais do problema.

Tipicamente a mutação é aplicada com probabilidade menor que 1%. Bremermann, em trabalhos antigos citados por Linden (2006), afirmou que a taxa de mutação ótima para problemas de otimização de cromossomos binários é igual a 1/*L*, onde *L* é igual ao número de variáveis binárias. Já Deb (1998) diz que a taxa de mutação ótima varia de acordo com a representação.

A figura 2.8 apresenta um exemplo de mutação bastante simples, onde o terceiro bit sofre mutação, ou seja, seu valor foi invertido de 1 para 0.

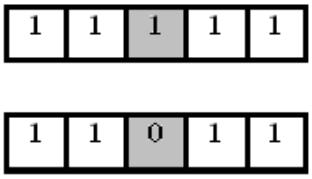

Figura 2.8 – Mutação

Uma mutação dirigida visa causar alterações no esquema dominante, causando modificações nos genes que nos interessam. Uma maneira de descobrir o esquema dominante seria aplicarmos um operador lógico XNOR entre os n elementos dois a dois. O Resultado desta operação é 1 nas posições onde todos os n indivíduos são iguais e 0 onde há elementos diferentes. Assim obtemos uma máscara que pode ser aplicada a qualquer

elemento para descobrirmos se esse esquema é comum. Pode ser criada uma versão menos restritiva, onde 2/3 ou 3/4 dos elementos sejam iguais ao invés de todos. Depois de descoberto o esquema que as melhores soluções têm em comum, o operador realiza as mutações. Isto vai fazer com que apareçam novas soluções com bagagem genética radicalmente diferente daquela que domina a população naquele instante.

Quatro indivíduos foram selecionados e foi verificado com o operador XNOR quais posições eram iguais em todos os bits. Isto permitiu criar uma máscara, contendo um, para os bits que são parte do esquema e zero para os que não são comuns a todos os indivíduos selecionados.

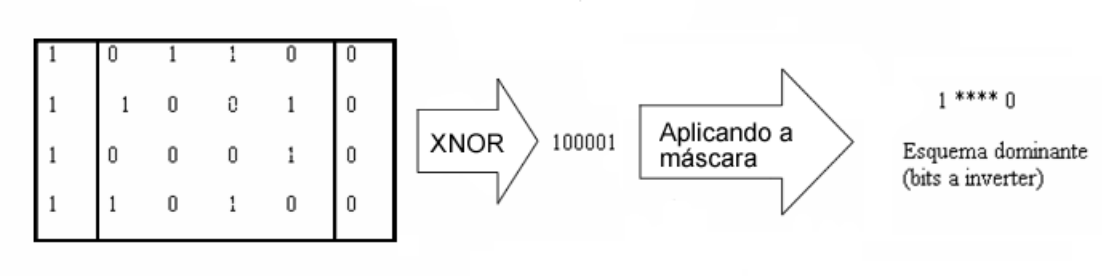

Figura 2.9 - Exemplo de escolha do esquema dominante.
(Fonte: LINDEN, 2006)

As mutações para cromossomos com codificação real são feitas através de operações matemáticas. Segundo Santa Catarina (2004), existem diversas formas de se realizar a mutação quando a codificação real está sendo utilizada, tais como mutação uniforme, mutação gaussiana, mutação não uniforme e não-uniforme múltipla. HERRERA *et al.* (1996) propuseram a mutação randômica que é bastante simples. Os genes do novo cromossomo são obtidos aleatoriamente de uma distribuição uniforme obtida em um intervalo determinado. Segundo o autor, a mutação uniforme é efetuada através da substituição do gene selecionado do cromossomo por outro gene gerado aleatoriamente entre os limites mínimos e máximos permitidos.

Segundo Linden (2006) o método de seleção de pais deve simular o mecanismo de seleção natural que atua sobre as espécies biológicas, em que os pais mais capazes geram mais filhos. Ao mesmo tempo, os pais menos aptos também devem poder gerar descendentes para que a população não tenha somente indivíduos semelhantes, sem determinada característica que no futuro possa vir a ser importante.

Após calcular o *fitness* para todas as cadeias da população atual, a seleção é feita. Segundo Whitley (1994), no algoritmo genético a probabilidade que o cromossomo seja duplicado e posicionado na população intermediária é proporcional à aptidão.

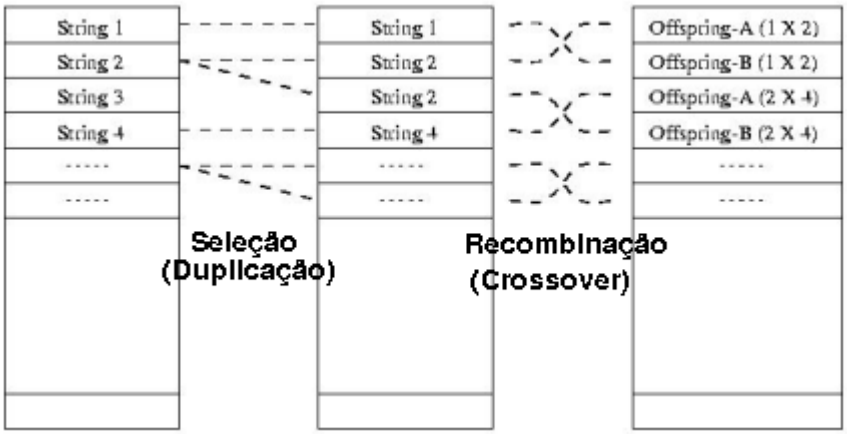

Figura 2.10 - Fases da geração, duplicação na seleção.

Existem diversas maneiras para efetuar o processo de seleção dentre elas podemos citar o método da roleta, torneio, ranking, seleção local, truncada, por amostragem estatística, etc. No método da seleção roleta é feita uma distribuição dos indivíduos numa roleta de maneira onde cada elemento ocupa um espaço proporcional à sua aptidão.

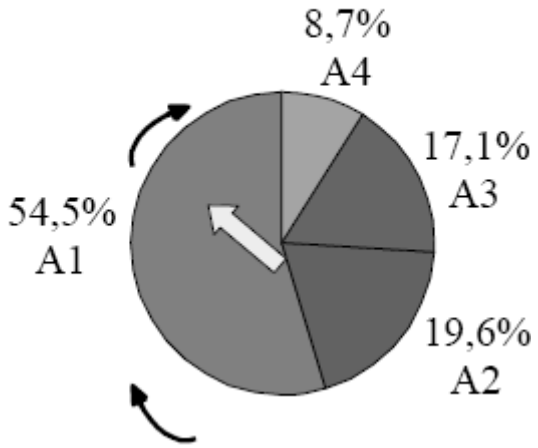

Figura 2.11 – Método de seleção da roleta

Como mostra a Figura 2.11, o indivíduo A1 é o mais apto e ocupa 54,5 % da roleta. Porém, a seleção é feita através de sorteio, assim mesmo o individuo A4 com 8,7% tem chance de ser escolhido.

O método do torneio consiste em escolher *n* indivíduos aleatórios e proporcionar entre eles um torneio onde suas armas são suas aptidões. O indivíduo com melhor avaliação dentre eles ganha essa competição. Segundo Linden (2006), evidências empíricas sugerem que o método do torneio com dois participantes seja melhor que o método da roleta.

Na seleção local, cada indivíduo existe dentro de um ambiente limitado que contém uma vizinhança que pode ser definida arbitrariamente. A primeira metade dos pais da nova geração é escolhida através de um outro método qualquer e, os outros indivíduos escolhidos farão parte da vizinhança dos primeiros.

A seleção por *ranking*, descrita por Mitchell (1996), é um método que evita a convergência prematura e a dominância do super indivíduo. Ele ordena os indivíduos de acordo com sua aptidão e utiliza para seleção sua posição ordenada, ou seja, seu *ranking* ao invés de seu valor de aptidão. Maiores detalhes podem ser encontrados em Linden (2006).

2.3. Algoritmos Genéticos multi-populacionais

Segundo Linden (2006), os biólogos perceberam que ambientes isolados como, por exemplo, ilhas, freqüentemente geram espécies animais melhor adaptadas às peculiaridades de seus ambientes do que com áreas de maior extensão. De acordo com Gould (1977) populações centrais grandes e estáveis exercem uma forte influência homogeneizante, pois novas mutações favoráveis são diluídas na grande massa populacional em que devem se espalhar. Assim modificações filéticas, processo evolutivo no qual uma unidade taxonômica diverge gradualmente de sua forma ancestral, mas sem ramificar ou dar origem a novas linhas evolutivas dentro do complexo, são raras nesse tipo de população. Enquanto em pequenas massas populacionais a pressão seletiva se torna mais intensa e variações favoráveis se espalham rapidamente.

Tratando-se de algoritmos genéticos aplicados a problema de otimização, a abordagem uni-populacional intensifica a busca num espaço de solução mais limitado e isso diminui a possibilidade de se encontrar soluções de melhor qualidade ou mesmo soluções factíveis. A abordagem multi-populacional prestigia o aspecto exploratório na busca de soluções (Toledo, 2005). As populações evoluem em paralelo, possibilitando

compartilhar os bons resultados entre as múltiplas populações, através de migrações de indivíduos, criando uma sinergia e tornando assim as abordagens multi-populacionais mais eficientes.

Conseqüentemente, escolhemos utilizar nesse trabalho um algoritmo genético multi-populacional, onde várias populações evoluem no decorrer das gerações. A abordagem multi-populacional no AG permite maior exploração do espaço de busca já que, segundo o conceito de *genetic drift*, indivíduos em populações diferentes seguem em geral caminhos evolutivos distintos.

2.4. *Exploitation e Exploration*

Vários problemas podem ser descritos como uma tentativa de alcançar um determinado objetivo, isto é, de chegar a um determinado estado onde uma condição é satisfeita. O AG é um esquema de busca que apresenta algumas boas características que potencializam essa busca.

A pesquisa é um dos métodos de resolução de problemas onde não se pode determinar a priori a seqüência de passos que conduzem a uma solução (GEN e CHENG, 1997). A busca pode ser realizada com as estratégias cegas ou estratégias heurísticas. Estratégias de busca cega não utilizam informações sobre o domínio do problema. Estratégias de busca heurística utilizam uma informação adicional para orientar a pesquisa junto com a melhor direção a ser pesquisada.

Há duas questões importantes na busca por estratégias: *exploitation e exploration*. Michalewicz (1994) fez uma comparação da busca *Hill-climbing*, pesquisa randômica e busca genética. *Hill-climbing* é um exemplo de uma estratégia que explora a melhor solução possível localmente ignorando uma exploração abrangente do espaço de busca (*exploitation*). Pesquisa randômica é um exemplo de estratégia que explora o espaço de busca, ignorando a exploração local das regiões promissoras (*exploration*). Algoritmos genéticos são uma classe de propósito geral que combina elementos de pesquisa que podem apresentar um notável equilíbrio entre *exploration* e *exploitation*. No início da pesquisa genética, há uma ampla e diversificada população randômica e o operador de *crossover* tende a intensificar a busca na vizinhança de cada uma das soluções.

2.5. Algoritmo Genético Híbrido

Existem muitas situações em que os simples algoritmos genéticos não são particularmente bons, e vários métodos de hibridização vem sendo propostos. Goldberg (1989) usou a idéia do G-bit para melhoria das cadeias binárias. Posteriormente incluiram melhorias nos operadores locais dos algoritmos genéticos puros para otimização de problemas NP-hard.

Gen e Cheng (1997) mencionam uma das formas mais comuns de algoritmos genéticos híbridos, a de incorporar uma otimização local como um complemento extra no simples laço de recombinação e seleção. A otimização local é aplicada a cada novo filho gerado na população. Algoritmos genéticos realizam a global *exploration* entre as populações, enquanto métodos heurísticos executam a local *exploitation* em torno do cromossomo. Assim, a abordagem híbrida muitas vezes melhora o desempenho destes métodos.

2.6 Considerações

Algoritmos Genéticos têm se mostrado uma versátil e efetiva ferramenta para ser utilizada na otimização de funções. Todavia, existem muitas situações em que os simples algoritmos genéticos não são particularmente bons, e vários métodos de hibridização vem sendo propostos.

Nosso trabalho utiliza o AG com uma codificação binária para os indivíduos e soma a essa excelente ferramenta de otimização algumas técnicas, como por exemplo, multi-populações hierarquicamente estruturadas, migração, reinicialização e ainda hibridiza o método com a busca local. O objetivo final será demonstrar os resultados desse método que utiliza de técnicas combinadas de forma viável para encontrarmos o ótimo global das funções de uma maneira mais eficiente do que outros métodos existentes.

3. METODOLOGIA

Neste capítulo é apresentado o método proposto por este trabalho. O trabalho apresenta um algoritmo genético multi-populacional e híbrido que será aplicado na otimização de funções existentes na literatura. Assim, detalhes envolvendo a abordagem

multi-populacional serão apresentados. Da mesma forma, as características que definem o método como híbrido serão detalhadas. Por último, as funções a serem otimizadas serão descritas.

3.1. Algoritmo Genético Híbrido Proposto

O Algoritmo Genético proposto por este trabalho é multi-populacional, híbrido, com indivíduos estruturados hierarquicamente e utiliza representação binária para os cromossomos. Nesta seção iremos descrever o algoritmo proposto e suas características.

3.1.1 Indivíduos e operadores genéticos

De acordo com Goldberg (1989), a representação binária dos indivíduos oferece o máximo número de esquemas por cada bit de informação de qualquer codificação. Isto torna mais fácil a análise teórica e permite a utilização de operadores genéticos mais elegantes. Por estas razões, os indivíduos que foram representados aqui estão usando o alfabeto binário. Contudo, a representação binária tem inconvenientes para lidar com muitas dimensões e alta precisão dos problemas (MICHALEWICZ, 1996).

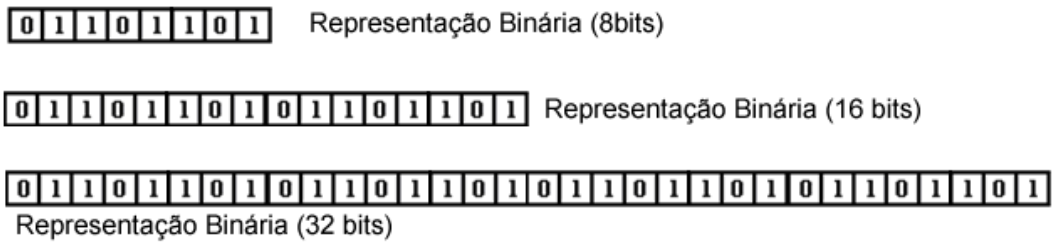

Figura – 3.1 Representação Binária

Nos testes computacionais, foram avaliados indivíduos com representação binária e tamanho de cadeia 8, 16 e 32 bits, assim como representado na figura 3.1. Essa faixa de trabalho foi definida a mérito de comparação com outros métodos de otimização presentes na literatura que serão definidos posteriormente.

O algoritmo foi avaliado utilizando o *crossover* de um ponto, dois pontos e uniforme. Esses *crossovers* foram implementados de acordo com a descrição apresentada para os mesmos no capítulo anterior.

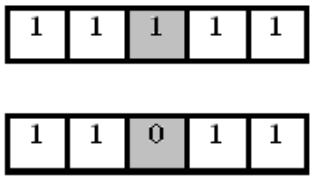

Figura 3.2 - Mutação

A figura 3.2 mostra o operador de mutação, cujo valor binário em cada posição do indivíduo selecionado pode ser alterado se a taxa de mutação for satisfeita assim como ocorre no terceiro bit do cromossomo.

3.1.2 Busca Local

O Algoritmo Genético proposto é híbrido, pois executa uma busca local sobre o melhor indivíduo de cada população. O procedimento encontra os vizinhos dos melhores indivíduos, através da alteração do valor binário de cada gene do indivíduo (figura 3.3). Cada um dos vizinhos é avaliado individualmente e substitui o indivíduo, se tiver um valor melhor de *fitness*.

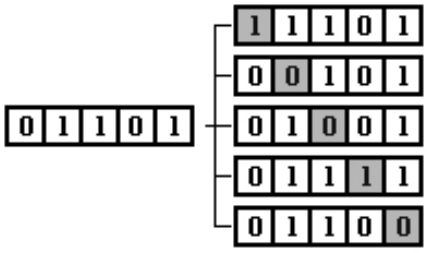

Figura 3.3 - O indivíduo e seus vizinhos

3.1.3. Estruturas da População e Migração

O AG proposto se baseia nas idéias vistas em Mendes (2003) e Toledo (2005) que já utilizam árvores binárias e ternárias como estruturas de populações, onde os indivíduos são posicionados de acordo com os valores de sua aptidão. A hierarquia é definida posicionando o melhor indivíduo como nó líder do *cluster* (subpopulação). Os indivíduos com valores inferiores de *fitness*, em relação ao líder, são posicionados no cluster como nós seguidores. Figura 3.4 ilustra esta representação usando uma estrutura binária para a população com 15 indivíduos.

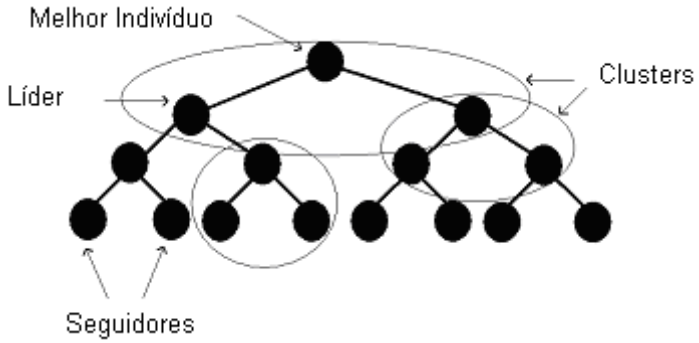

Figura 3.4. - População estruturada em árvore binária.

Na Figura 3.4, população estruturada em árvore binária apresenta 7 *clusters* com 3 indivíduos cada, arranjados em 3 níveis: 1 *cluster* no nível um, dois *clusters* no nível 2 e quatro no nível 3. Nos testes computacionais, foram avaliadas estruturas binárias com 7, 15 e 31 indivíduos.

Outra abordagem avaliada neste trabalho considera indivíduos hierarquicamente estruturados em árvores ternárias. Figura 3.5 ilustra esta situação utilizando uma estrutura para as populações ternárias com 13 indivíduos.

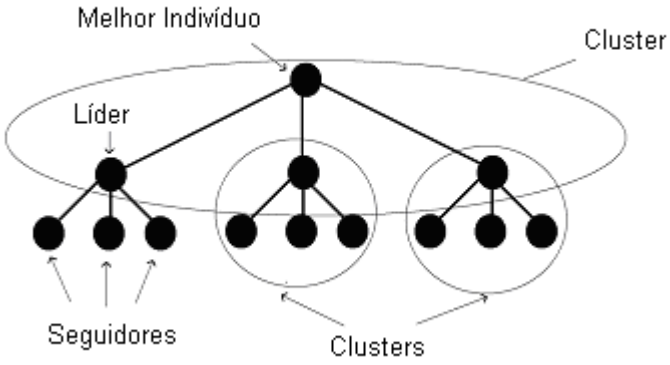

Figura 3.5 - População estruturada em árvores ternárias.

Na Figura 3.5 temos a população estruturada em árvore ternária onde cada população possui quatro clusters de quatro indivíduos cada, arranjados em dois níveis: um cluster no nível um e três no nível dois. Nos testes computacionais, foram avaliadas estruturas ternárias com 4, 13 e 40 indivíduos.

O processo de recombinação será detalhado num contexto de estrutura populacional em árvore binária. Inicialmente, um cluster é selecionado de forma aleatória. O melhor indivíduo do cluster é escolhido para realizar a recombinação com um dos seus seguidores

aleatoriamente selecionado. O indivíduo filho será inserido no lugar de um dos pais, caso seu valor de *fitness* seja melhor. Um exemplo é dado na figura 3.6

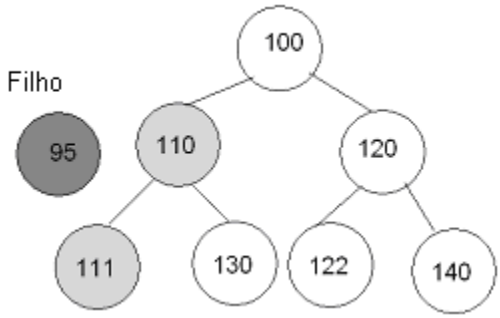

Figura 3.6 - População antes da inserção de novo indivíduo.

Levando em consideração que seja um problema de minimização, onde o pior *fitness* é aquele com maior valor numérico, um melhor indivíduo com valor de *fitness* 110 é aleatoriamente selecionado bem como seu seguidor com valor 111. O filho resultante dessa combinação apresenta um *fitness* de 95. O novo indivíduo será inserido no lugar do pai com pior valor de *fitness.* A Figura 3.7 apresenta um *cluster* após a inserção do novo indivíduo.

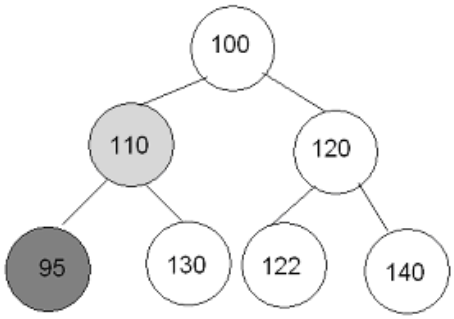

Figura 3.7 - População após inserção de novo indivíduo.

A fase de reestruturação da população ocorre após as recombinações. Todos os *clusters* são percorridos e, se algum dos seguidores apresenta melhor *fitness* que um líder, uma troca de posições irá ocorrer. As Figuras 3.8 e 3.9 exemplificam a reestruturação da população.

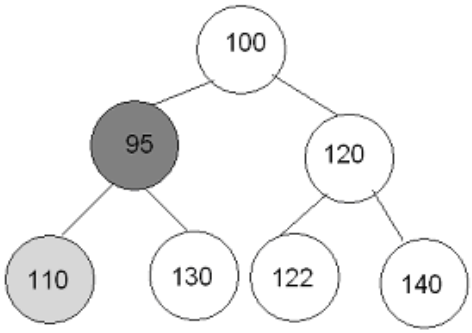

Figura 3.8 - Restauração das populações – Passo 1

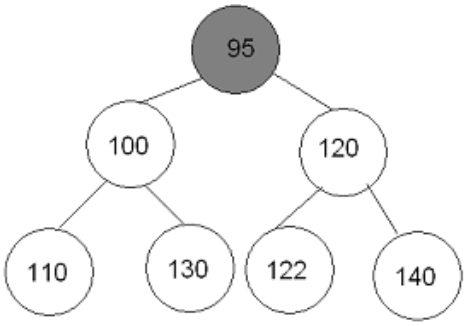

Figura 3.9 - Restauração das populações – Passo 2

Se após uma série de recombinações não é possível criar indivíduos com valor de *fitness* melhor que um dos pais, a população terá convergido e deverá ser reinicializada. Uma migração acontece antes de a população ser reinicializada. (Figura 3.10)

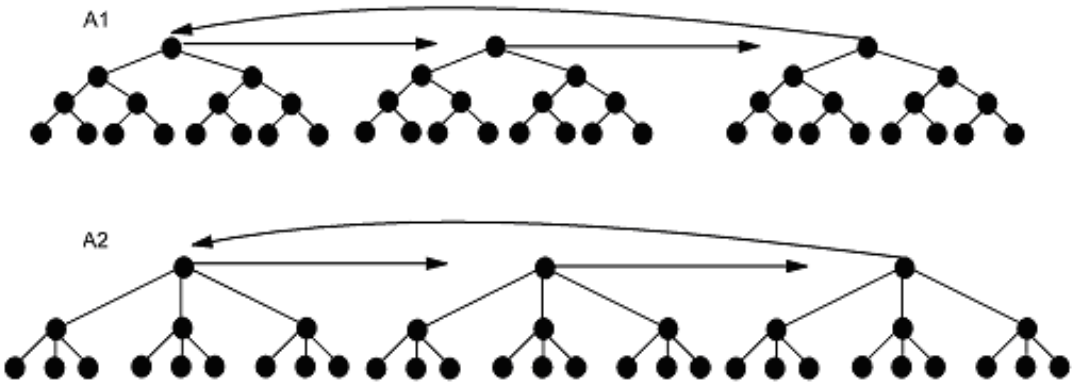

Figura 3.10. O melhor indivíduo migra para a próxima população

Na figura 3.10 as setas indicam a migração de uma cópia do melhor indivíduo para a população seguinte. Esse indivíduo é inserido no lugar de um indivíduo aleatoriamente selecionado, exceto o melhor da população. A reinicialização da população poupa os melhores indivíduos e aqueles que acabaram de migrar.

3.1.4 Algoritmo Genético Multi-Populacional Híbrido

O pseudocódigo apresentado na figura 3.11 resume o algoritmo genético multi-populacional e híbrido proposto por este trabalho.

```
Method algGeneticoMultiPopulacional;
begin
        repeat
                for i = 1 to numberOfPopulations do
                        inicializarPopulação(pop(i));
                        avaliaFitnessPopulação(pop(i));
                        estruturaPopulação(pop(i));
                        repeat
                                for j = 1 to numberOfRecombinations do
                                        selecionaPais(individuoA, individuoB) C pop(i);
                                        novoInd = recombina(individuoA, individuoB);
                                        if(efetuaMutação novoInd) then novoInd =
mutação(novoInd);
                                        avaliaFitnessIndviduo(novoInd);
                                        inserePopulação(novoInd, pop(i));
                                end
                                estruturaPopulação(pop(i));
                        until(populaçãoConvergiu pop(i))
                end
                for i = 1 to numberOfPopulations do
                        efetuaMigração pop(i);
                end
        until (condição de parada)
end
```

Figura 3.11 - Pseudocódigo de um AG estruturado.

O método **estruturaPopulação**(*pop(i)*) organiza os indivíduos dentro de cada população. Os operadores genéticos de seleção (**selecionaPais**(*individuoA, individuoB*)), recombinação (**recombina** (*individuoA, individuoB*)), e mutação (**mutação**(*novoInd*)) são

executados enquanto a população *i* avaliada não convergir. Uma população converge se, após certo número de iterações, nenhum novo indivíduo é inserido. Se todas as populações avaliadas convergiram, antes do critério de parada ser satisfeito, ocorrerá uma migração entre as populações (**efetuaMigração** *pop(i)*) e uma nova inicialização das populações. Todavia, essa nova inicialização poupa os melhores indivíduos e aqueles indivíduos que acabaram de migrar. Trata-se de uma reinicialização parcial da população.

O AGH (Algoritmo Genético Híbrido) proposto por este trabalho foi desenvolvido utilizando a linguagem C++ baseado no framework NpOpt desenvolvido em Java por Mendes (2003), com intuito de apresentar uma estruturação bem definida de classes com um forte conceito de orientação a objetos, permitindo um bom reaproveitamento de código.

Todos os parâmetros do AGH são definidos através de um arquivo, o *batch*.txt, que é lido pelo método no início da execução. Ao final do processo é gerado outro arquivo com a tabela de resultados.

3.2. Funções

O foco principal deste trabalho é analisar o comportamento dos AG proposto na otimização de funções, cujo objetivo é vasculhar o espaço de busca por soluções de determinadas funções. Na sua maioria são funções do tipo unimodal e algumas multímodais. Funções unimodais são funções que apresentam um único ótimo. Funções multimodais são aquelas que possuem apenas um ótimo global e vários ótimos locais.

De acordo com Diagalakis & Margaritis (2000), a análise teórica do uso de algoritmos genéticos não é uma ferramenta que poderia determinar completamente a escolha mais adequada dos melhores operadores e parâmetros a serem adotados. Assim, a avaliação de uma metodologia para AG através da otimização de funções ajuda a mensurar a eficiência e o desempenho de vários aspectos do AG. Por exemplo, teremos uma melhor idéia a respeito da qualidade dos indivíduos inicialmente gerados, as estruturas populacionais e os operadores genéticos propostos. Essa abordagem também pode auxiliar na definição ou ajuste de parâmetros do método.

A seguir, são apresentadas as funções não lineares de duas variáveis: Goldstein and Price, B2, Easom, Rosenbrock, Shubert. Por último, apresentaremos De Joung e Hartmann que são funções com três variáveis.

A fórmula (1) defini a função que chamaremos de GP. O seu domínio está dentro do intervalo de [-2,2], possui quatro mínimos locais e um mínimo Global GP(0,-1) = 3.

$$GP\ (x_1, x_2) = [1 + (x_1 + x_2 + 1)^2 \times (19 - 14\ x_1 + 3\,x_1^2 - 14\ x_2 + 6\,x_1 x_2 + 3\,x_2^2)] \times [30 + (2\,x_1 - 3\,x_2)^2 \times (18 - 32\ x_1 + 12\ x_1^2 + 48\ x_2 - 36\ x_1 x_2 + 27\ x_2^2)] \quad (1)$$

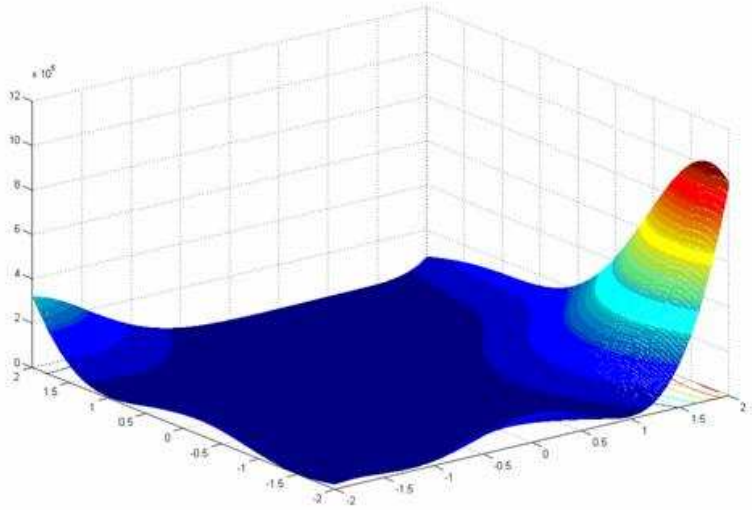

Figura 3.12 - Função GP

Fonte: Hadel (2008)

A fórmula (2) define a função que chamaremos de B2. O seu domínio está dentro do intervalo de [-100,100] e seu ótimo global é B2(0,0) = 0.

$$B2(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7 \quad (2)$$

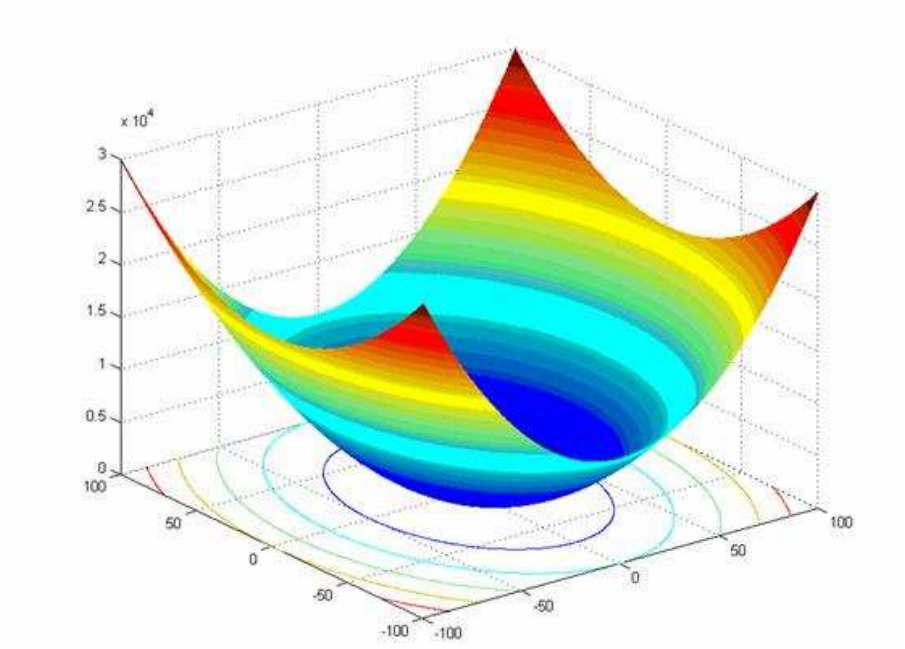

Figura 3.13 - Função B2

Fonte: Hadel (2008)

A fórmula 3 defini a função Easom que é uma função unimodal, onde o mínimo global tem uma pequena área em relação ao seu espaço de busca e seu domínio está dentro do intervalo de [-100,100] e seu ótimo global é ES (π ,π) = -1

$$ES(x_1, x_2) = -\cos(x_1)\cos(x_2)\exp(-((x_1 - \pi)^2 + (x_2 - \pi)^2)) \quad (3)$$

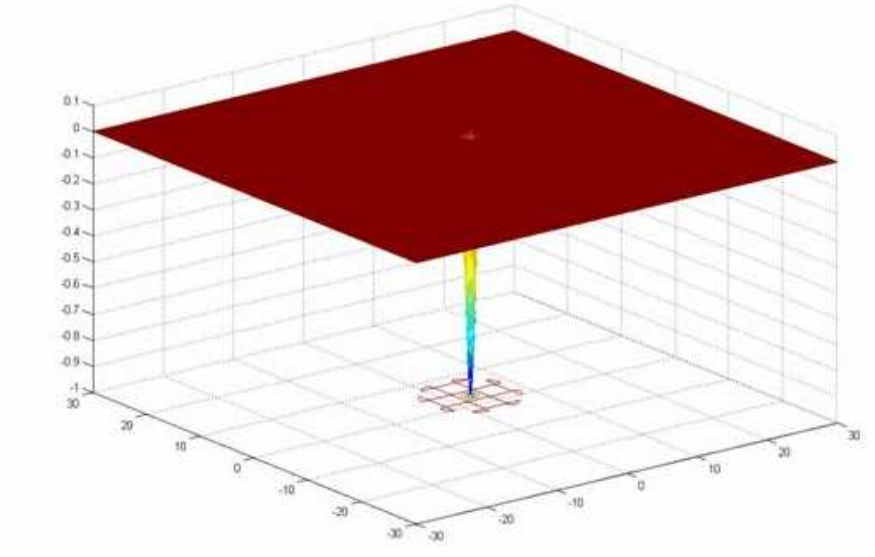

Figura 3.14 - Função Eason

Fonte: Hadel (2008)

A fórmula 4 define Rosenbrock que é um problema clássico de otimização que aqui chamaremos de R2, mas também conhecido como função banana. O ótimo global está dentro de uma longa e estreita parábola, aos moldes de um vale. Para encontrar o vale é trivial, porém a convergência para o ótimo global é difícil e, portanto, este problema tem sido repetidamente utilizado para avaliar o desempenho de algoritmos de otimização.O seu domínio está dentro do intervalo de [-5,10]. A R2 na verdade ela pode ser considerada Rn, pois o número de variáveis desta função e determinada pelo fator *n*, porém estamos utilizando-a com duas varáveis. Seu ótimo global é definido por R2(1,1) = 0 e ela é formalizada pela fórmula:

$$R_n(x) = \sum_{j=1}^{n-1}[(1-x_j)^2 + 100(x_{j+1} - x_j^2)^2] \qquad (4)$$

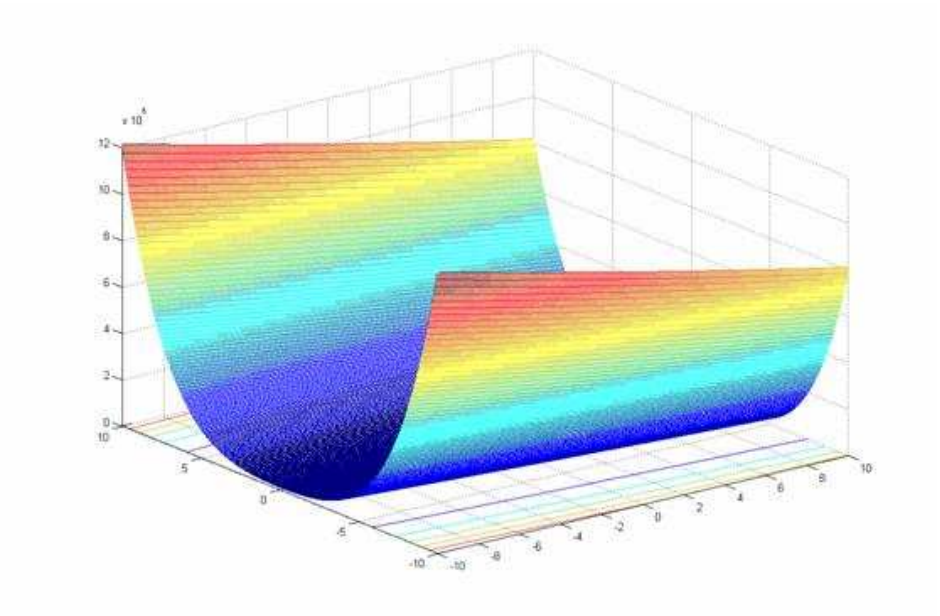

Figura 3.15 - Função R

Fonte: Hadel (2008)

Esta é a função que chamaremos de SH é definida pela fórmula (5). O seu domínio está dentro do intervalo de [-10,10]. Esta função possui 18 ótimos locais e um ótimo global SH(x1,x2) = - 186.7309.

$$SH(x_1, x_2) = \{\sum_{j=1}^{5} j\cos[(j+1)x_1 + j]\} \times \{\sum_{j=1}^{5} j\cos[(j+1)x_2 + j]\} \qquad (5)$$

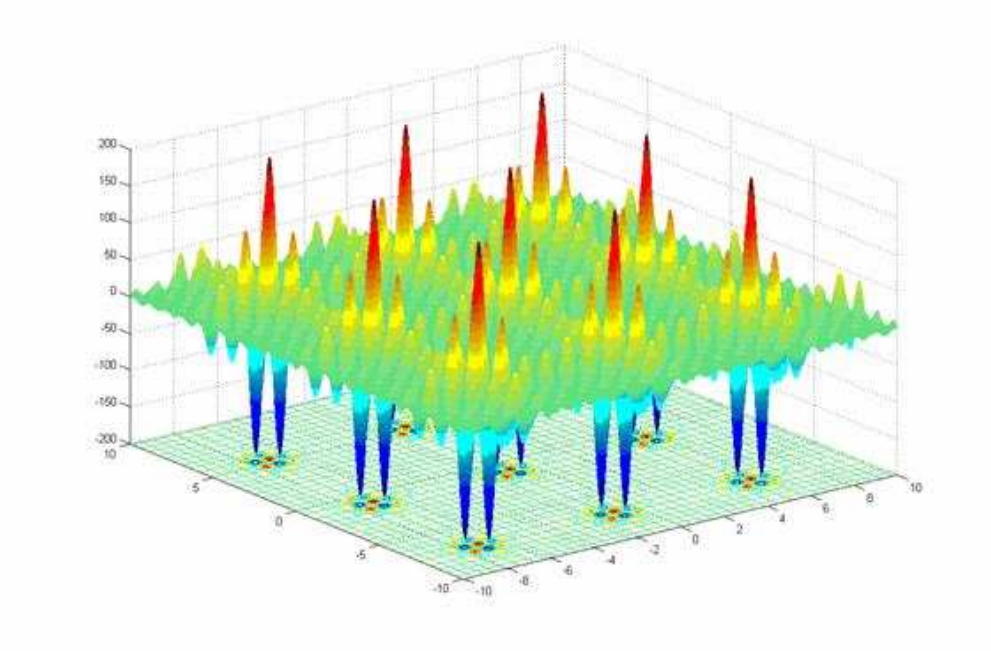

Figura 3.16 - Função SH

Fonte: Hadel (2008)

Chamaremos a função da fórmula (6) de DJ, porém ela é muito conhecida como Sphere. O seu domínio está dentro do intervalo de [-5.12,5.12] e possui três variáveis, com isto possui um mínimo Global DJ(0,0,0) = 0.

$$DJ(x_1, \quad x_2, \quad x_3) = x_1^2 \quad + \quad x_2^2 + x_3^2 \tag{6}$$

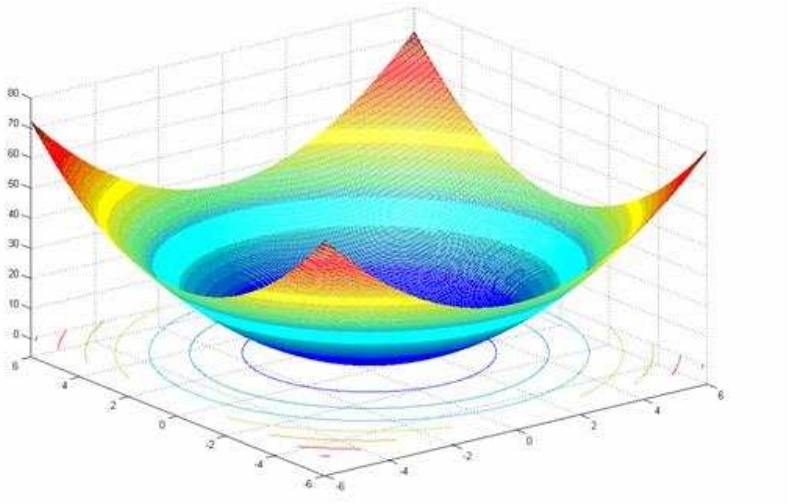

Figura 3.17 - Função Sphere com duas variáveis

Fonte: Hadel (2008)

A fórmula (7) define a função que chamaremos de H3,4. O seu domínio está dentro do intervalo de [0,1] e possui três variáveis, com isto possui um mínimo Global H3,4(0.11; 0.555; 0.855) = - 3.86278. A H3,4 pode ser definida pela fórmula:

$$H_{3,4}(x) = -\sum_{i=1}^{4} c_i \exp[-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2] \quad (7)$$

As funções apresentadas são *benchmarkings* utilizadas a fim de demonstrar a eficiência de algoritmos genéticos e já foram aplicadas a diversas técnicas de otimização onde algumas serão comparadas com o AGH. Se o método proposto conseguir bons resultados para estas funções significa que este método possui todas as características necessárias para obter sucesso em qualquer outro problema no qual um algoritmo Genético possa ser aplicado.

O próximo capítulo apresenta os resultados computacionais e as considerações feitas após a comparação do AGH com outras técnicas de otimização que utilizaram o mesmo conjunto de funções já apresentadas.

4. Resultados Computacionais

As diversas abordagens para o Algoritmo Genético Híbrido (AGH) proposto serão avaliadas em cada uma das funções apresentadas na seção 3.2 do capítulo 3. Essas abordagens se distinguem de acordo com a notação descrita a seguir:

- AGH representa Algoritmo Genético Híbrido.
- AGHb indica o AGH com estruturação dos indivíduos em árvores binárias, onde:
  - AGHb7 para 7 indivíduos em cada população.
  - AGHb15 para 15 indivíduos em cada população.
  - AGHb31 para 31 indivíduos em cada população.
- AGHt indica o AGH com estruturação dos indivíduos em árvores ternárias, onde:
  - AGHt4 para 4 indivíduos em cada população.
  - AGHt13 para 13 indivíduos em cada população.
  - AGHt40 para 40 indivíduos em cada população.
- AGH(t)/(b) s indica o AGH com determinado tamanho de cromossomo, onde
  - AGH s8 para cromossomos com cadeia binária com 8 bits.
  - AGH s16 para cromossomos com cadeia binária com 16 bits.
  - AGH s 31 para cromossomos com cadeia binária com 32 bits.

Os testes computacionais serão realizados considerando diversas combinações desses parâmetros de acordo com as nomenclaturas mostradas na Tabela 1.

Binária	Ternária
AGHb7s8	AGHt4s8
AGHb7s16	AGHt4s16
AGHb7s32	AGHt4s32
AGHb15s8	AGHt13s8
AGHb15s16	AGHt13s16
AGHb15s32	AGHt13s32
AGHb31s8	AGHt40s8
AGHb31s16	AGHt40s16
AGHb31s32	AGHt40s32

Tabela 1 – Descrição das abordagens avaliadas

As cadeias binárias foram fixadas com tamanho 8, 16 e 32 para permitir a comparação com resultados relatados em Berthiau e Siarry (1997) que utilizaram os mesmos tamanhos de cadeia binária. Procurou-se também avaliar o comportamento do

AGH quanto ao tipo de estrutura populacional (binária e ternária), quanto ao tamanho dessas estruturas e quanto ao número de populações.

4.1. Avaliação das Abordagens para AGH

Os resultados reportados nesta seção procuram avaliar o desempenho das variações do AGH quanto ao tipo de *crossover* utilizado à medida que o número de populações aumenta. Assim, avalia-se o AGH ajustado com uma, duas e três populações. Para cada população, o método é executado com *crossover* de um ponto, dois pontos e uniforme.

As tabelas nesta seção apresentam o número médio de avaliações de funções executadas. Essa média é calculada considerando as 100 execuções do AGH realizadas em cada função. O número de avaliações das funções consiste no número de vezes que o *fitness* (função considerada) é calculado no processo evolutivo até se determinar o *fitness* de um indivíduo cujo valor seja o ótimo global da função. O valor é considerado ótimo quando a diferença entre o *fitness* obtido e o valor ótimo está dentro da precisão proposta por Chelouah e Siarry (2003). Essa precisão varia de acordo com a função utilizada. As tabelas 2, 3 e 4 apresentam os resultados do AGH com uma única população.

AGH	B2	DJ	ES	GP	H34	R2	SH
AGHb7s8	-	-	-	-	570	10865	-
AGHb7s16	-	85	5686	**170**	852	13398	23612
AGHb7s32	121	133	9330	239	1331	21551	32939
AGHb15s8	-	-	-	-	441	7348	-
AGHb15s16	-	274	2850	360	504	6558	10818
AGHb15s32	567	332	3531	406	507	7305	10437
AGHb31s8	-	-	-	-	1162	7489	-
AGHb31s16	-	1095	1705	1362	1295	5393	7847
AGHb31s32	1742	1132	2319	1414	1244	5112	6936
AGHt4s8	-	-	-	-	1333	30219	-
AGHt4s16	-	57	12138	243	3095	71887	29016
AGHt4s32	76	104	19679	464	5971	118316	58829
AGHt13s8	-	-	-	-	**433**	5304	-
AGHt13s16	-	231	2194	288	479	5981	6752
AGHt13s32	441	286	3804	335	516	6528	6933
AGHt40s8	-	-	-	-	1538	6009	-
AGHt40s16	-	1737	2033	1982	1802	4597	4722
AGHt40s32	2174	1756	2106	2039	1848	5014	4798

Tabela 2 – AGH com uma população e *crossover* uniforme

AGH	B2	DJ	ES	GP	H34	R2	SH
AGHb7s8	-	-	-	-	37641	112646	-
AGHb7s16	-	57	12502	296	58991	147010	149422
AGHb7s32	439	101	29423	471	146084	345285	371436
AGHb15s8	-	-	-	-	13211	56932	-
AGHb15s16	-	98	6485	336	51636	89013	94148
AGHb15s32	123	129	15745	391	71207	191140	262315
AGHb31s8	-	-	-	-	19409	40858	-
AGHb31s16	-	318	5043	802	49227	69361	105282
AGHb31s32	429	235	8062	539	64425	122135	304180
AGHt4s8	-	-	-	-	42884	215746	-
AGHt4s16	-	**51**	26862	486	158374	318155	183479
AGHt4s32	**61**	96	57378	822	297363	722709	466548
AGHt13s8	-	-	-	-	18359	68640	-
AGHt13s16	-	92	6580	271	66540	85610	75230
AGHt13s32	212	122	19121	353	70081	183677	153143
AGHt40s8	-	-	-	-	21230	38505	-
AGHt40s16	-	552	4897	1162	56768	56514	77180
AGHt40s32	654	407	7969	757	88253	91576	190450

Tabela 3 – AGH com uma população e *crossover* de um ponto

AGH	B2	DJ	ES	GP	H34	R2	SH
AGHb7s8	-	-	-	-	881	13432	-
AGHb7s16	-	88	1448	221	1687	13037	5789
AGHb7s32	146	150	4898	342	2598	26728	7668
AGHb15s8	-	-	-	-	542	4085	-
AGHb15s16	-	278	694	323	929	5246	3236
AGHb15s32	423	351	1278	396	878	7829	2435
AGHb31s8	-	-	-	-	1062	4125	-
AGHb31s16	-	825	1222	848	1398	4587	2388
AGHb31s32	848	901	1723	925	1632	6986	2498
AGHt4s8	-	-	-	-	2837	44164	-
AGHt4s16	-	55	7382	365	6803	75531	27424
AGHt4s32	169	106	14911	714	14970	138970	31529
AGHt13s8	-	-	-	-	562	5658	-
AGHt13s16	-	222	**650**	263	1044	8599	2956
AGHt13s32	362	271	1852	324	1416	14453	3696
AGHt40s8	-	-	-	-	1461	**3694**	-
AGHt40s16	-	1051	1590	1146	1833	6340	**2424**
AGHt40s32	942	1172	1962	1089	2229	7599	2781

Tabela 4 – AGH com uma população e *crossover* de dois pontos

As entradas em branco nas tabelas indicam que o tamanho da cadeia binária utilizada não foi suficiente para se atingir a precisão desejada em determinadas funções. Dentre os AGHs com uma população, o *crossover* uniforme retorna os melhores resultados

em GP e H34. O melhor resultado para GP é encontrado quando a população está estruturada em árvore binária com 7 indivíduos e comprimento da cadeia de 16 bits (AGHb7s16). O melhor resultado para H34 é encontrado quando a população está estruturada em árvore ternária com 13 indivíduos e comprimento da cadeia de 8 bits (AGHt13s8). Utilizando *crossover* de um ponto temos os melhores resultados em B2 e DJ. Os melhores resultados para B2 e DJ são encontrados quando a população está estruturada em árvore ternária com 4 indivíduos. O melhor resultado para B2 ocorre com cadeia de 32 bits (AGHt4s32) e para DJ com cadeia de 16 bits (AGHt4s16). O *crossover* de dois pontos apresenta melhores resultados para ES, R2 e SH. Os melhores resultados para ES, R2 e SH são encontrados quando a população está estruturada em árvore ternária. O melhor resultado para ES foi obtido com população de 13 indivíduos e cadeia de 16 bits (AGHt13s16). O melhor resultado para R2 e SH ocorre com 40 indivíduos na população e cadeia de 8 bits (AGHt40s8) e 16 bits (AGHt40s8), respectivamente.

Neste ponto, observamos que o *crossover* de dois pontos para o AGH estruturado em árvore com uma população apresentou os melhores resultados em 3 das 7 funções (ES, R2 e SH) considerando o melhor resultado de cada função dentre as 3 tabelas com resultados de uma população. O *crossover* uniforme e o *crossover* de um ponto apresentaram bons resultados para duas funções cada. As tabelas 5, 6 e 7 apresentam os resultados obtidos com duas populações.

AGH	B2	DJ	ES	GP	H34	R2	SH
AGHb7s8	-	-	-	-	659	17341	-
AGHb7s16	-	91	6582	**180**	838	26490	14779
AGHb7s32	133	141	11483	274	1647	43012	21627
AGHb15s8	-	-	-	-	486	6820	-
AGHb15s16	-	299	2242	411	560	6512	6180
AGHb15s32	554	345	4060	417	552	6488	7733
AGHb31s8	-	-	-	-	1129	6998	-
AGHb31s16	-	1140	1793	1396	1306	5424	6006
AGHb31s32	1780	1150	1947	1443	1339	5369	5867
AGHt4s8	-	-	-	-	2472	59652	-
AGHt4s16	-	61	14610	255	4740	99207	38409
AGHt4s32	80	108	30509	504	9772	220071	71356
AGHt13s8	-	-	-	-	**395**	7019	-
AGHt13s16	-	242	2673	313	490	7192	4706
AGHt13s32	430	297	4351	370	513	7448	5956
AGHt40s8	-	-	-	-	1533	5288	-
AGHt40s16	-	1739	1871	2049	1806	5106	4927
AGHt40s32	2234	1774	2053	2128	1781	5049	4131

Tabela 5 – AGH com duas populações e *crossover* uniforme.

AGH	B2	DJ	ES	GP	H34	R2	SH
AGHb7s8	-	-	-	-	15867	99842	-
AGHb7s16	-	66	13449	247	54813	134469	80471
AGHb7s32	83	112	36013	552	117662	251768	184539
AGHb15s8	-	-	-	-	9878	49479	
AGHb15s16	-	123	6198	340	36803	62928	47554
AGHb15s32	137	153	14918	411	41482	126959	113609
AGHb31s8	-	-	-	-	68455	31769	-
AGHb31s16	-	389	4962	918	13306	40598	68928
AGHb31s32	370	285	5991	671	84242	91260	150181
AGHt4s8	-	-	-	-	29366	292102	-
AGHt4s16	-	**55**	26753	493	93453	356603	110124
AGHt4s32	**74**	103	67712	1302	208908	554195	337314
AGHt13s8	-	-	-	-	13352	51284	-
AGHt13s16	-	105	7828	297	26869	72252	41627
AGHt13s32	140	143	14141	384	57986	160932	79306
AGHt40s8	-	-	-	-	13744	31443	-
AGHt40s16	-	616	4281	1296	37413	39748	55356
AGHt40s32	597	398	8366	830	56743	84775	108932

Tabela 6 – AGH com duas populações e *crossover* um ponto.

AGH	B2	DJ	ES	GP	H34	R2	SH
AGHb7s8	-	-	-	-	1070	17442	-
AGHb7s16	-	96	1635	208	2394	25228	10754
AGHb7s32	188	145	5476	257	2183	42990	14122
AGHb15s8	-	-	-	-	618	5550	-
AGHb15s16	-	279	**762**	316	840	7563	2919
AGHb15s32	401	336	1157	376	966	12155	2530
AGHb31s8	-	-	-	-	1089	**4515**	-
AGHb31s16	-	845	1147	862	1391	5135	2527
AGHb31s32	843	932	1629	898	1560	7325	**2277**
AGHt4s8	-	-	-	-	3932	56721	-
AGHt4s16	-	60	11161	365	9417	76916	30608
AGHt4s32	85	109	36187	678	15191	272503	55317
AGHt13s8	-	-	-	-	496	8828	-
AGHt13s16	-	238	795	298	752	11236	2918
AGHt13s32	405	286	1586	316	1042	18675	4398
AGHt40s8	-	-	-	-	1436	4737	-
AGHt40s16	-	1108	1523	1108	1848	6110	2467
AGHt40s32	1024	1196	2051	1222	1892	9758	2710

Tabela 7 – AGH com duas populações e *crossover* de dois pontos.

Dentre os AGHs com duas populações, o que utiliza *crossover* uniforme obteve o melhor resultado na função GP e H34. Para GP, utilizando estrutura binária com 7 indivíduos em cada população (14 indivíduos no total) e comprimento da cadeia de 16 bits (AGHb7s16). Para H34, utilizando estrutura ternária com 13 indivíduos em cada

população (26 indivíduos no total) e cadeia de 8 bits (AGHt13s8), o AGH retornou o melhor resultado para a função H34. O *crossover* de um ponto, obteve os melhores resultados para B2 e DJ, utilizando a estrutura ternária com 4 indivíduos em cada população (8 indivíduos no total) e cadeia com 32 bits (AGHt4s32) para B2 e 16 bits para DJ (AGHt4s16).

O *crossover* de dois pontos obteve os melhores resultados para ES, R2 e SH. O método apresentou melhor resultado para ES utilizando estrutura binária com 15 indivíduos em cada população (30 indivíduos no total) e cadeia de 16 bits (AGHb15s16). Utilizando árvore binária com 31 indivíduos em cada população (62 indivíduos no total), o AGH apresentou os melhores resultados pra R2 com cadeia de 8 bits (AGHb31s8) e para SH com cadeia de 32 bits (AGHb31s32).

Até este momento podemos observar que o *crossover* de dois pontos continua se destacando. Obteve o melhor resultado em 3 das 7 funções enquanto o *crossover* de um ponto se saiu melhor em duas funções e o uniforme em uma única função. As tabelas 8-10 apresentam os resultados utilizando três populações.

AGH	**B2**	**DJ**	**ES**	**GP**	**H34**	**R2**	**SH**
AGHb7s8	-	-	-	-	468	15547	-
AGHb7s16	-	100	4881	204	797	26009	15225
AGHb7s32	131	147	9663	269	1162	39693	25748
AGHb15s8	-	-	-	-	516	7118	-
AGHb15s16	-	310	2328	408	512	5924	6374
AGHb15s32	593	360	3225	458	577	5614	6696
AGHb31s8	-	-	-	-	1207	7253	-
AGHb31s16	-	1168	1394	1385	1370	5439	6440
AGHb31s32	1789	1207	1917	1471	1357	6051	6020
AGHt4s8	-	-	-	-	1626	56669	-
AGHt4s16	-	65	15415	262	3742	102836	33843
AGHt4s32	84	113	26791	484	6403	212776	58514
AGHt13s8	-	-	-	-	442	7006	-
AGH**t13s16**	-	252	2154	346	**429**	6499	4823
AGHt13s32	486	310	3069	386	461	6737	5755
AGHt40s8	-	-	-	-	1615	6388	-
AGHt40s16	-	1852	1685	2090	1941	5535	4690
AGHt40s32	2311	1843	2097	2223	1966	5782	4665

Tabela 8 – AGH com três populações e *crossover* uniforme.

AGH	B2	DJ	ES	GP	H34	R2	SH
AGHb7s8	-	-	-	-	14170	72184	-
AGHb7s16	-	73	12794	288	43508	149477	57777
AGHb7s32	87	119	30994	439	111296	234029	134222
AGHb15s8	-	-	-	-	9269	45147	-
AGHb15s16	-	132	6757	367	28666	64117	46857
AGHb15s32	160	163	12156	363	58962	167276	90660
AGHb31s8	-	-	-	-	12766	34224	-
AGHb31s16	-	394	4902	862	52773	60064	62682
AGHb31s32	397	334	6531	717	38675	98613	101108
AGHt4s8	-	-	-	-	27483	239947	-
AGHt4s16	-	**59**	26573	577	67965	320765	135459
AGHt4s32	**75**	106	62593	823	155727	819360	331664
AGHt13s8	-	-	-	-	13014	52081	-
AGHt13s16	-	116	7750	343	31016	64764	41026
AGHt13s32	137	153	17146	426	68544	148037	86176
AGHt40s8	-	-	-	-	11695	30244	-
AGHt40s16	-	637	4403	1364	46037	41711	53995
AGHt40s32	640	478	5676	895	56481	93826	68254

Tabela 9 – AGH com três populações e *crossover* de um ponto.

AGH	B2	DJ	ES	GP	H34	R2	SH
AGHb7s8	-	-	-	-	979	17033	-
AGHb7s16	-	106	1946	**177**	1676	22517	9635
AGHb7s32	163	153	3786	308	3179	46219	12762
AGHb15s8	-	-	-	-	588	**4785**	
AGHb15s16	-	307	**679**	345	842	6201	3275
AGHb15s32	454	380	1090	421	947	9986	2662
AGHb31s8	-	-	-	-	1125	4825	
AGHb31s16	-	822	1189	920	1595	4807	2737
AGHb31s32	890	963	1701	977	1458	8175	**2168**
AGHt4s8	-	-	-	-	3358	52432	
AGHt4s16	-	66	5983	541	9837	101499	33919
AGHt4s32	90	114	35254	730	17849	230392	55913
AGHt13s8	-	-	-	-	552	6344	
AGHt13s16	-	257	742	268	937	9209	3184
AGHt13s32	361	309	1332	373	1065	19602	3388
AGHt40s8	-	-	-	-	1413	4994	
AGHt40s16	-	1145	1619	1154	1921	5514	2309
AGHt40s32	1121	1324	2119	1128	2026	8735	2537

Tabela 10 – AGH com três populações e *crossover* de dois pontos.

O AGH com 3 populações e *crossover* uniforme encontrou o melhor resultado para H34 quando foi estruturada em árvore ternária com 13 indivíduos em cada população e 39

indivíduos no total e cadeia de 16 bits (AGHb13s16). Utilizando o *crossover* de um ponto, o AGH encontrou os melhores resultados para B2 e para DJ usando árvore ternária com 4 indivíduos em cada população (12 indivíduos no total) ajustado, respectivamente, com cadeia de 16 bits (AGHt4s16) e 32 bits (AGHt4s32).

O *crossover* de dois pontos encontrou os melhores resultados para as funções ES, GP, R2 e SH. Os melhores resultados foram obtidos através da estruturação em árvore binária, sendo que para GP e ES foi utilizada codificação de 16 bits, SH codificação de 32 bits e R2 8 bits. O método obteve melhor resultado para ES com 7 indivíduos em cada população e 21 indivíduos no total (AGHb7s16), para GP com 15 indivíduos em cada população e 45 indivíduos no total (AGHb15s16), para R2 com 15 indivíduos em cada população e 45 indivíduos no total (AGHb15s8), e para SH com 31 indivíduos em cada população e 93 indivíduos no total (AGHb31s32). Considerando três populações, o *crossover* de dois pontos foi melhor em quatro funções enquanto o uniforme foi superior em uma e o de um ponto melhor em duas funções.

Ao passo que o número de populações aumentava, os três tipos de *crossovers* continuaram obtendo melhores resultado para determinadas funções. Isso nos leva a crer que o uso de certo tipo de *crossover* não pode ser inferido somente através de características do método, mas também pela característica do problema.

Levando em consideração que cada tabela possui 7 funções e foram feitos testes para 3 populações temos então 21 testes. O *crossover* uniforme obteve melhor resultado em 5 dos 21 testes e manteve certa coerência de resultados. Para todas as funções, levando em consideração todos os resultados obtidos pelo *crossover* uniforme, podemos observar que o AGH se aproximou da melhor resposta na maioria dos casos. O *crossover* de um ponto, por exemplo, com determinadas configurações extrapola em muito o resultado esperado.

O *crossover* de um ponto obteve melhores resultados em 6 dos 21 testes. Sempre se saiu melhor que os outros nas funções B2 e DJ, onde o método gastou poucas soluções para encontrar o ótimo. O *crossover* de dois pontos, o melhor em 10 dos 21 testes, se apresentou melhor em funções que o método encontrou maior dificuldade para chegar ao ótimo.

Os AGH com estruturas de árvores ternárias apresentam um desempenho superior às estruturas binárias em 12 dos 21 melhores resultados encontrados para as funções. Árvores com menor número de níveis com, por exemplo, 2 e 3 níveis somaram 13 dos

melhores resultados, enquanto que utilizando 4 e 5 níveis somaram 8 dos melhores resultados.

4.2. Comparação com Resultados da Literatura

Nesta seção iremos comparar as configurações do AGH, que retornaram os melhores resultados, com resultados existentes na literatura para o mesmo conjunto de funções. Inicialmente, a tabela 11 apresenta as configurações do AGH com melhores resultados. Mas para isto foi necessário acrescentarmos algumas informações na nomenclatura já definida anteriormente.

- Número de Populações:
    - AGHt4s32_**p1** acrescentando p1 para uma população.
    - AGHt4s32_**p2** acrescentando p2 para duas populações.
    - AGHt4s32_**p3** acrescentando p3 para três populações.
- Tipo de *crossover:*
    - AGHt4s32_p1**c1** acrescentando c1 para *crossover* de um ponto.
    - AGHt4s32_p1**c2** acrescentando c2 para *crossover* de dois pontos.
    - AGHt4s32_p1**Un** acrescentando Un para *crossover* uniforme.

AGH	Função	Resultado
AGHt4s32_p1c1	B2	61
AGHt4s16_p1c1	DJ	51
AGHt13s16_p1c2	ES	650
AGHb7s16_p1Un	GP	170
AGHt13s8_p2Un	H34	395
AGHt40s8_p1c2	R2	3694
AGHb31s32_p3c2	SH	2168

Tabela 11 – Melhores resultado obtidos pelo AGH

A Tabela 12 compara o AGH com os resultados dos métodos apresentados em Chelouah e Siarry (2003):

- *Continuous Hybrid Algorithm* (CHA), Chelouah e Siarry (2003).
- *GA for globally minimizing function* (GAGMF), Berthiau e Siarry (1997).

- Continuous genetic algorithm (CGA), Chelouah e Siarry (2000b).
- *Continuous reactive tabu search* com valor mínimo (CRT min) e médio (CRT med.), Battiti e Tecchiolli (1996).
- *Enhanced continuous tabu* (ECTS), Chelouah e Siarry (2000a).

O GAGMF utiliza um código binário com comprimento 8 (GAGMF8b), 16 (GAGMF16b) e 32 (GAGMF32b). O GAGMF também obteve resultados usando uma codificação Gray com as mesmas cadeias binárias (GAGMF8g, GAGMF16g e GAGMF32g). As minimizações com taxas de sucesso inferior a 100% nas 100 execuções realizadas sobre cada função estão entre parênteses na Tabela 12.

	B2	**DJ**	**ES**	**GP**	**H34**	**R2**	**SH**
GAGMF8b	-	-	-	891	1968	1554(0.42)	-
GAGMF8g	-	-	-	1085(0.98)	508(0.98)	1764(0.53)	-
GAGMF16b	-	-	-	1997	1742	3675(0.39)	-
GAGMF16g	-	-	-	2748(0.96)	2277(0.99)	5596(0.54)	-
GAGMF32b	-	-	-	2873(0.99)	2860	4834(0.35)	-
GAGMF32g	-	-	-	4008(0.96)	2735(0.99)	5829(0.52)	-
CGA	320	-	-	410	582	960	575
CHA	132	371	952	259	492	**459**	**345**
ECTS	210	-	-	231	548	480	370
CRTS min.	-	-	-	171	609	-	-
CRTS ave.	-	-	-	248	513	-	-
AGHt4s32_p1c1	**61**	96	57378	822	297363	722709	466548
AGHt4s16_p1c1	-	**51**	26862	486	158374	318155	183479
AGHt13s16_p1c2	-	222	**650**	263	1044	8599	2956
AGHb7s16_p1Un	-	85	5686	**170**	852	13398	23612
AGHt13s8_p2Un	-	-	-	-	**395**	7019	-
AGHt40s8_p1c2	-	-	-	-	1461	3694	-
AGHb31s32p3c2	890	963	1701	977	1458	8175	2168

Tabela 12 – Comparação com Resultados da Literatura

As abordagens do AGH foram melhores nas funções B2, DJ, ES, GP e H34. Estes resultados superaram CGA e CHA que utilizam codificação real. Os AGH com árvores ternárias e uma única população encontraram os melhores resultados em B2, DJ e ES. Sendo que B2 e DJ utilizaram *crossover* de um ponto e ES *crossover* de dois pontos. O AGH utilizou duas populações e *crossover* uniforme em H34. O melhor resultado para o GP é encontrado usando uma estrutura binária com uma única população e *crossover* uniforme.

Na função R2, AGHt40s8_p1c2 supera o GAGMF com codificação binária e codificação Gray de comprimento 32. Ele também supera o GAGMF com codificação

Gray de comprimento 16. No entanto, todas as abordagens de GAGMF têm menos de 60% de taxa de sucesso nas minimizações, enquanto que o AGH apresenta taxa de 100% nas mesmas funções. De forma geral, o AGH não conseguiu obter bons resultados para R2 e SH. O método foi superado por CGA e CHA que utilizam codificação real para os indivíduos.

5. Conclusão

Este trabalho apresentou e avaliou o uso de um Algoritmo Genético Híbrido (AGH) na otimização de sete funções com resultados disponíveis na literatura. O AGH proposto utiliza codificação binária de tamanhos 8, 16 e 32 para representar os indivíduos. Os indivíduos pertencem a populações hierarquicamente estruturadas. As populações estão estruturadas em árvores binárias ou ternárias e cada indivíduo é posicionado na árvore de acordo com seu valor de *fitness*. O AGH também executa busca em vizinhança sobre os melhores indivíduos encontrados em cada população.

O método foi avaliado quanto ao número de populações, tipo e tamanho da estrutura populacional, e uso de diferentes tipos de *crossovers*. Os testes computacionais revelaram que, quando o número de populações aumentava, as mesmas variações do AGH com determinados tipos de *crossovers* continuaram obtendo melhores resultados em um mesmo subconjunto de funções. Isso nos leva a acreditar que o uso de certo tipo de *crossover* não pode ser definido somente através de características do método, mas também pela característica do problema. Neste trabalho, os *crossovers* apresentaram um desempenho semelhante, onde em 3 das 7 funções o *crossover* de dois pontos se saiu melhor que o *crossover* de um ponto e o *crossover* uniforme. Tanto o *crossover* de um ponto quanto o uniforme obtiveram sucesso em 2 das 7 funções.

Os melhores resultados obtidos pelo AGH foram comparados aos resultados existentes na literatura. O AGH proposto superou abordagens como Busca Tabu e Algoritmos Genéticos com codificação binária e real. Os AGH com estruturas ternárias, utilizando uma e duas populações, retornaram os melhores resultados em 4 das 7 funções (B2, DJ, ES e H34). A estrutura binária retornou o melhor resultado com uma única população para GP. Assim, o AGH proposto superou os métodos existentes em 5 das 7 funções avaliadas. A maioria dos melhores resultados foi obtida com uma abordagem uni-populacional para o AGH. O uso de população hierarquicamente estruturada e a execução de busca local sobre os melhores indivíduos podem explicar o desempenho do método proposto.

Os AGHs não obtiveram um bom desempenho na função Rosenbrock (R2), mas apresentaram uma taxa de sucesso superior a de outras abordagens para AG utilizando codificação binária (GAGMF). Na função Shubert (SH), o AGH não se saiu bem como em outros métodos (CGA e CHA) que utilizam codificação real.

Algumas configurações do AGH proposto não conseguiram atingir a precisão necessária para alcançar o ponto ótimo em algumas funções, ao utilizar cadeias binárias de tamanho 8 e 16. Considerando também os melhores resultados obtidos por CGA e CHA na função SH, o uso de codificação real para indivíduos no AGH é uma proposta para trabalho futuro.

6. REFERÊNCIAS BIBLIOGRÁFICAS

ATMAR, W. **On the Rules and Nature of Simulated Evolutionary Programming**, Proc. of the First Ann. Conf. On Evolutionary Programming, pp. 17-26, 1992.

BÃCK, T.D.B.FOGEL., and MICHALEWICZ, T., editors. **Evolutionary Computation1, Basic Algorithms and Operators.** Institute of Physics Publishing, 2000a

BÃCK, T.D.B.FOGEL., and MICHALEWICZ, T., editors. **Evolutionary Computation2, Advanced Algorithms and Operators.** Institute of Physics Publishing, 2000b

BATTITI, R., TECCHIOLLI, G.: **The continuous reactive tabu search: Blending combinatorial optimization and stochastic search for global optimization**. Annals of Operations Research 63, 53--188 (1996).

BERTHIAU, G., SIARRY, P.: **A genetic algorithm for globally minimizing functions of several continuous variables**. In: Second International Conference on Meta-heuristics, Sophia-Antipolis, France (1997).

BESSAOU, M; SIARRY, P, **A genetic algorithm with real-value coding to optimize multimodal continuous functions**, Struct Multidisc Optim 23, 63–74, Springer-Verlag 2001

BREMERMANN, H.J. **Optimization through evolution and recombination**, *in* M.C. Yovits, G.T. Jacobi & G.D. Goldstine (eds.) *Self-Organizing Systems*, pp. 93-106, Spartan Books,1962.

CASTRO L. & VON ZUBEN. F., **“Introdução à Computação Evolutiva”**, disponível no endereço ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/ia707_02/topico8_02.pdf , visto no dia dezessete de janeiro de 2008.

CHELOUAH.R , SIARRY.P, **Tabu search applied to global optimization**, European Journal of Operational Research 123/2 (2000a)

CHELOUAH.R, SIARRY.P, **A continuous genetic algorithm designed for the global optimization of multimodal functions**, Journal of Heuristics 6 (2000b)

CHELOUAH, R., SIARRY, P.: **Genetic and Nelder–Mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions**. European Journal of Operational Research 148, 335--348 (2003).

DAVIS, S.G., Shedulling. **Economic lost size production runs**. Management Science., 36(8), 1990.

DEB. K. A. **Understanding Interactions Among Genetic Algorithms Parameters.** In Banzhaf, São Francisco, EUA, 1998.

DIAGALAKIS, JG; MARGARITIS,K.G. **On Benchmarking Functions for Genetic Algorithms**, University of Macedonia, 54046, Thssalonikki, Greece, (Received 9 March 2000; in final form 8 September 2000).

FOGEL, L.J. **Autonomous automata**, *Industrial Research*, vol. 4, pp. 14-19, 1962.

FOGEL, L.J., OWENS, A.J. & WALSH, M.J. **Artificial Intelligence through Simulated Evolution**, Wiley, 1966.

FOGEL, D.B. **An Introduction to Simulated Evolutionary Computation**, *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 3-14, 1994.

FRASER, A.S. **Simulation of genetic systems by automatic digital computers**: I. Introduction, *Austral. J. Biol.Sci.*, vol. 10, pp. 484-491, 1957.

GEN, M., CHENG, R.: **Genetic algorithms & engineering design.** John Wiley & Sons NewYork (1997).

GOLDBERG, D.E. **Genetic Algorithms in search, optimization, and machine learning.** Addison Wesley, 1989.

GOLDBERG, D.E. **Information Models, Views, and Controllers**. Dr. Dobb's Journal, July 1990

GOULD S.J., **Evolutions errate pace,** Natural History, Nova Iórque, EUA, 1997.

HADEL, A, **Global Optimization Test Problems,** http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm, 2008.

HARTL, D.L. & CLARK, A.G. **Principles of Population Genetics**, Sinauer, 1989.

HERRERA F, LOZANO M.,VERDEGA Y. **Trackling Real Coded Genetic Algorithms – Operator and Tools for Behavioral Analysis.** Artificial Inteligence Review 12, Klewer Academic Publichers, Amsterdã, Holanda, 1996.

HOLLAND, J. H. **"Outline for a logical theory of adaptative systems**", J. Assoc. Comput. Mach.,vol.3, pp 297-314. (1962)

HOLLAND, J.H. **Adaptation in natural and artificial systems,** The University of Michigan Press, 1975.

HOLLAND, J.H. **"Adaptation in Natural and Artificial Systems**", 2nd edition, The MIT Press, 1992.

KOZA, J.R.. **Genetic Programming: On the Programming of Computers by Means of Natural Selection**. MIT Press, 1992.

LINDEN, Ricardo. **Algoritmos genéticos: uma importante ferramenta da inteligência computacional**. Rio de Janeiro: Brasport, 2006

MAYR, E. **Toward a New Philosophy of Biology: Observations of an Evolutionist**, Belknap Press, 1988.

MENDES, A.S., **O Framework NP-Opt e suas Aplicações a Problemas de Otimização.** Phd Thesis, Universidade Estadual de Campinas, 2003.

MICHALEWICZ, Z. "**Genetic algorithms + Data Structures = Evolution Programs**", 2rd edition, Springer-Verlag, 1994.

MICHALEWICZ, Z. "**Genetic algorithms + Data Structures = Evolution Programs**", 3rd edition, Springer-Verlag, 1996.

MITCHELL, M. "**An Introduction to Genetic Algorithms**", The MIT Press, 1996.

RECHENBERG, I. **Cybernetic solution path of an experimental problem**, *Royal Aircraft Establishment*, Library Translation no. 1122, 1965.

RECHENBERG, I. **Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution**, Frommann-Holzboog, 1973.

SANTA CATARINA, **A. Aplicações De Algoritmos Genéticos em Sistemas de Informações Geográficas**. Monografia final do Curso de Introdução ao Geoprocessamento. INPE, São José dos Campos, 2004.

SCHWEFEL, H.-P. **Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik**, Diplomarbeit, Hermann Föttinger Institut für Strömungstechnik, Technische Universität, Berlin, 1965.

SCHWEFEL, H.-P. **Evolutionsstrategie und numerische Optimierung**, Dissertation, Technische Universität, Berlin, 1975.

SCHWEFEL, H.-P**. Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie**. Basel:Birkhäuser, 1977.

TOLEDO, C.F.M. **Problema Conjunto de Dimensionamento de Lotes e Programação da Produção,** Campinas, SP, UNICAMP, 2005.

VON ZUBEN, F.J. **Computação Evolutiva: Uma Abordagem Pragmática**, DCA/FEEC/UNICAMP disponível em dia oito do mês de janeiro no ano de dois mil e sete no endereço ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/tutorial/tutorialEC.pdf

VOSE. M.. **Modeling Genetic Algorithms.** Fundation of genetic Algorithms, -2-, D.Whitley, ed., Morgan Kaufmann. pp: 63-73. (1993)

WHITLEY, D., **A Genetic Algorithm Tutorial,** Computer Science Departament, Colorado State University Fort Collins, 1994

Optimization of Multiminima Functions using a Memetic Algorithm with Population Hierarchically Structured

Abstract. This paper presents preliminary results using a memetic algorithm as an approach for global optimization of multiminima functions. The memetic algorithm proposed has individuals hierarchically structured in trees, where binary and ternary tree structures are evaluated. The method can also handle several populations which evolve separately. The binary strings are initially used to encode solutions. The computational results for seven benchmark functions are reported. These results are compared to other approaches found in the literature.

Keywords: Memetic Algorithm, Multi-population, Hierarchical structures, Global optimization, Multiminima functions.

1 Introduction

This paper studies the use of a memetic algorithm with population hierarchically structured as a method for global optimization. The global optimization is the computation and characterization of global optimum [1], which can be a minimum or maximum value of nonconvex functions in a specific domain. The problem at hand can be unconstrained or constrained, where optimization techniques must find the global values without being trapped into local points [2].

Hybrid genetic algorithms have been used to solve many problems [3], with applications in global optimization [4]-[5]. The Memetic Algorithm (MA) is a hybrid population-based approach which combines the strength of population methods with the intensification capability of a local search [6]. A memetic algorithm with structured population has been proposed to solve the asymmetric traveling salesman problem [7] and the single machine scheduling problem [8].

The MA proposed by this paper can handle several populations, where their individuals are hierarchically structured in trees. This hierarchy among individuals means that the best ones will be placed in the highest level of the tree structure. The use of binary and ternary trees will be evaluated in this paper as well as the number of populations. A total of seven multiminima functions are initially used as benchmarks to evaluate the MA approaches, where the results found are compared with those existing in the literature [9]-[13]. The next sections describe the MA (section 2) and

the multiminima functions (section 3). The computational results are shown on section 4 and the conclusions are presented on section 5.

2. The Multi-Population Memetic Algorithm

This section describes the Memetic Algorithm (MA) approaches.

Pseudo code for the Memetic algorithm

```
MultiPopulationMemeticAlg()
begin
 repeat
  for i = 1 to numberOfPopulations do
    initializePopulation(pop( i ));
    evaluatePopulationFitness(pop( i ));
    structurePopulation(pop( i ));
    repeat
     for j = 1 to numberOfCrossover do
       selectParents(individualA,individualB);
       newInd=crossover(individualA, individualB);
       if (executeMutation newInd) then
             newInd=mutation(newInd);
       evaluateFitnessIndividual(newInd);
       insertPopulation(newInd, pop(i ));
     end
     structurePopulation(pop( i ));

    until(populationConvergence pop(i ));
    localSearch(pop(i), bestInd);
  end
  for i = 1 to numberOfPopulations do
    executeMigration pop(i );
  end
 until(stop criterion)
end
```

In the multi-population pseudo code, the first task is to generate the initial populations. Next, genetic operators (crossover and mutation) are executed for each population. The *numberOfCrossover* is defined by *crossoverRate*sizeOfPopulation*. The population converges when new individuals are not inserted. Thus, a local search for better results is executed in the neighborhoods of the best individual. A migration and a new initialization take place when all populations have converged. The aim is to bring back diversity to the populations. However, this new initialization keeps the best individual and the individual that has just migrated to the population.

2.1 Individual, Genetic Operators and Local Search

According to [14], the binary representation of individuals offers the maximum number of schemata per bit of information of any coding. This becomes easier theoretical analysis and allows the use of more elegant genetic operators. For these reasons, the individuals were represented here using the binary alphabet. However, the binary representation has drawbacks to deal with multidimensional and high-precision problems [15]. The uniform crossover was chosen in this paper based on previous computational experiments. For each gene in the same position, a random value $\lambda \in [0,1]$ is generated. If $\lambda < 0.5$, the new individual inherits the gene of one parent; otherwise, it inherits the gene of the other parent. In the mutation operator, each binary position in the selected individual can be changed, if $\lambda <$ *mutationRate* for $\lambda \in [0,1]$.

MA executes a local search for better results when it was not possible to insert a new individual in the population. This procedure looks for neighbors of the best individual of this population. A neighbor is another individual slightly changed by neighborhood operations (moves). In this paper, the local search built-in changes the binary value in each gene of the best individual (fig. 1). Each neighbor of the best individual is evaluated and replaces the best one, if it has a better function value.

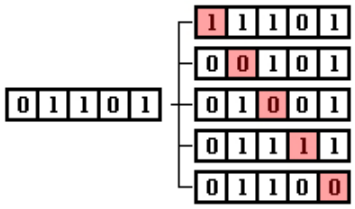

Fig. 1 The individual and its neighbors obtained by a bit change in each position.

2.2 Population structures

The population in a so-called canonical genetic algorithm has individuals ranked by their fitness value with no special structure connecting them. In this paper, it is proposed the use of binary and ternary tree structures for populations, where the individuals are positioned according to their fitness values. The hierarchy is defined by a leader node (better cluster individual) followed by its supporters. Fig. 2 illustrates this using a binary structure for population with 15 individuals.

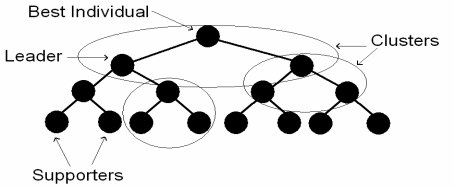

Fig. 2. Population structured in binary trees. Each population has 7 clusters of 3 individuals each, arranged in three levels: one cluster in level 1, two clusters in level 2 and four clusters in level 3.

In the computational tests, it is evaluated binary structures with 7, 15 and 31 individuals. Another MA approach considers individuals hierarchically structured in ternary trees. Fig. 3 illustrates this using a ternary structure for population with 13 individuals. Ternary structures with 7, 15 and 31 individuals are also evaluated in the computational tests.

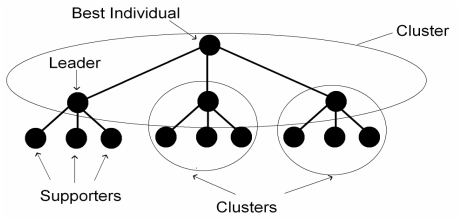

Fig. 3. Population structured in ternary trees. Each population has 4 clusters of 4 individuals each arranged in two levels: one cluster in level 1 and three clusters in level 2.

The selection of parents for recombination (crossover) begins selecting a cluster at random. The cluster leader is always selected and one of its supporters is randomly chosen. The new individual (child) will be inserted in this cluster if it is better than one of their parents. The population structure must be updated after any child insertion (Fig. 4).

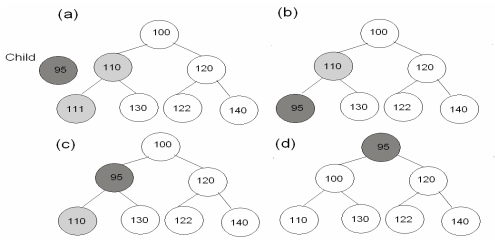

Fig. 4. Updating the population structure after the child insertion.

A migration occurs when new individuals are not inserted in all populations. A copy of the best individual of each population moves to the next population replacing any individual randomly chosen, except the best one (Fig. 5).

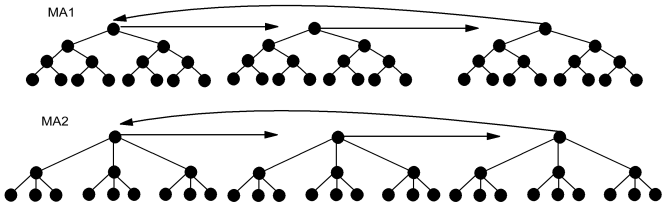

Fig. 5. Migration. The best individual migrates to the next population.

3. Multiminima Functions

This section present the seven multiminima functions used as benchmarks to evaluate the memetic algorithms (MAs). There are computational results for the same functions established by other approaches [10]-[14], which will allow evaluating better the performance of the MAs. First, it is presented the nonlinear functions: Goldstein and Price (1), B2 (2), Easom (3), Rosenbrock (4) and Shubert (5).

$$GP(x_1, x_2) = [1 + (x_1 + x_2 + 1)^2 \times (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)] \quad (1)$$

$$B2(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7 \quad (2)$$

$$ES(x_1, x_2) = -\cos(x_1)\cos(x_2)\exp(-((x_1 - \pi)^2 + (x_2 - \pi)^2)) \quad (3)$$

$$R_n(x) = \sum_{j=1}^{n-1}[(1 - x_j)^2 + 100(x_{j+1} - x_j^2)^2] \quad (4)$$

$$SH(x_1, x_2) = \{\sum_{j=1}^{5} j\cos[(j+1)x_1 + j]\} \times \{\sum_{j=1}^{5} j\cos[(j+1)x_2 + j]\} \quad (5)$$

These functions have 2 variables. The Goldestein and Price has domains $2 < x_j < 2$ with j = 1, 2 and global minimum $GP(0,-1) = 3$.The B2 and Easom have the same domains $-100 < x_j < 100$, with j = 1, 2 and global minimum $B2(0,0) = 0$ and $ES(\pi,\pi) = -1$. The Rosenbrock with $n = 2$ has domains $-5 < x_j < 10$ with j = 1, 2 and global minimum $R_2(1,1) = 0$. The Shubert has domains $-10 < x_j < 10$ with j = 1, 2, eighteen global minima $SH(x_1,x_2) = -186.7309$. The last functions are De Joung (6) and Hartmann (7) with three variables.

$$DJ(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2 \quad (6)$$

$$H_{3,4}(x) = -\sum_{i=1}^{4} c_i \exp[-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2] \quad (7)$$

The De Joung has domains $-5.12 < x_j < 5.12$ with j = 1, 2, 3 and global minimum $DJ(0,0,0) = 0$. The Hartmann has domains $0 < x_j < 1$ with j=1, 2, 3 and global minimum $H_{3,4}(0.11; 0.555; 0.855) = -3.86278$.

4. Computational Results

The minimization values obtained by the Memetic Algorithms (MAs) approaches are evaluated using the same criterion proposed in [10], where several methods were compared taking in account the average of the objective function evaluation numbers after 100 minimizations. Thus, the MAs are also executed 100 times for each benchmark function. However, each function minimization is considered a success here, if the gap between the best objective function value returned by the MAs and the global optimum is less than the average gaps determined in [10]. The time limit of each MA run is set to 10 seconds. The MAs are executed with one, two and three populations using binary and ternary structures. The binary structures are evaluated with 7 individuals (3 levels), 15 individuals (4 levels) and 31 individuals (5 levels). The ternary structures are evaluated with 4 individuals (2 levels), 13 individuals (3 levels) and 40 individuals (4 levels). The main idea is to compare similar population sizes for each tree structure. The MAs were set with a crossover rate of 0.7 and a mutation rate of 0.07.

The length of the binary string is adjusted to 8, 16 and 32 bits. These values were chosen to allow a fair comparison with the results reported in [10] for methods which also uses binary coding. However, the binary string with length 8 did not reach the necessary gap precision in functions B2, DJ, ES and GP. The same happened with the binary string with length 16 in function B2. Tables 1-3 compare the MAs approaches listing the average of the objective function evaluation numbers. The approaches were named according to their tree structures and the length of the binary string. For example, MAb7s8 means the Memetic Algorithm executed using binary tree structure with 7 individuals (b7) and binary string with length 8 (s8).

Table 1. MA results with 1 populations.

MA	B2	DJ	ES	GP	H34	R2	SH
MAb7s8	-	-	-	-	570	10865	-
MAb7s16[d]	-	85	5686	**170**	852	13398	23612
MAb7s32	121	133	9330	239	1331	21551	32939
MAb15s8	-	-	-	-	441	7348	-
MAb15s16	-	274	2850	360	504	6558	10818
MAb15s32	567	332	3531	406	507	7305	10437
MAb31s8	-	-	-	-	1162	7489	-
MAb31s16	-	1095	1705	1362	1295	5393	7847
MAb31s32	1742	1132	2319	1414	1244	5112	6936
MAt4s8	-	-	-	-	1333	30219	-
MAt4s16[b]	-	**57**	12138	243	3095	71887	29016
MAt4s32[a]	**76**	104	19679	464	5971	118316	58829
MAt13s8	-	-	-	-	433	5304	-
MAt13s16	-	231	2194	288	479	5981	6752
MAt13s32	441	286	3804	335	516	6528	6933
MAt40s8	-	-	-	-	1538	6009	-

MAt40s16[f]	-	1737	2033	1982	1802	**4597**	4722
MAt40s32	2174	1756	2106	2039	1848	5014	4798

The MA with 1 population (Table 1) has the best results in B2, DJ, GP and R2. The best results for B2 and DJ is found using a ternary tree with only 4 individuals and binary coding with lenght 32 (MAt4s32) and 16 (MAt4s16), respectively. The best result for R2 is found using a ternary tree structure with 40 individuals and binary coding with length 16 (MAt40s16). The best result for GP is reached by a binary structure using 7 individuals and binary coding with length 16 (MAb7s16).

Table 2. MA results with 2 poppulations.

	B2	DJ	ES	GP	H34	R2	SH
MAb7s8	-	-	-	-	659	17341	-
MAb7s16	-	91	6582	180	838	26490	14779
MAb7s32	133	141	11483	274	1647	43012	21627
MAb15s8	-	-	-	-	486	6820	-
MAb15s16	-	299	2242	411	560	6512	6180
MAb15s32	554	345	4060	417	552	6488	7733
MAb31s8	-	-	-	-	1129	6998	-
MAb31s16	-	1140	1793	1396	1306	5424	6006
MAb31s32	1780	1150	1947	1443	1339	5369	5867
Mat4s8	-	-	-	-	2472	59652	-
Mat4s16	-	61	14610	255	4740	99207	38409
Mat4s32	80	108	30509	504	9772	220071	71356
Mat13s8[e]	-	-	-	-	**395**	7019	-
MAt13s16	-	242	2673	313	490	7192	4706
MAt13s32	430	297	4351	370	513	7448	5956
Mat40s8	-	-	-	-	1533	5288	-
MAt40s16	-	1739	1871	2049	1806	5106	4927
MAt40s32[g]	2234	1774	2053	2128	1781	5049	**4131**

The MA with 2 populations (Table 2) using ternary structure had the best results for H34 and SH. In this case, the ternary structure with 13 individual in each population (26 individuals in all) and binary coding with length 8 (MAt13s8) returns the best result for H34. The ternary structure with 40 individuals in each population (80 individuals in all) and binary coding with length 32 (MAt40s32) returns the best results for SH.

Table 3. MA results with 3 poppulations.

MA	B2	DJ	ES	GP	H34	R2	SH
MAb7s8	-	-	-	-	468	15547	-
MAb7s16	-	100	4881	204	797	26009	15225

MAb7s32	131	147	9663	269	1162	39693	25748
MAb15s8	-	-	-	-	516	7118	-
MAb15s16	-	310	2328	408	512	5924	6374
MAb15s32	593	360	3225	458	577	5614	6696
MAb31s8	-	-	-	-	1207	7253	-
MAb31s16[c]	-	1168	**1394**	1385	1370	5439	6440
MAb31s32	1789	1207	1917	1471	1357	6051	6020
Mat4s8	-	-	-	-	1626	56669	-
Mat4s16	-	65	15415	262	3742	102836	33843
Mat4s32	84	113	26791	484	6403	212776	58514
Mat13s8	-	-	-	-	442	7006	-
Mat13s16	-	252	2154	346	429	6499	4823
Mat13s32	486	310	3069	386	461	6737	5755
Mat40s8	-	-	-	-	1615	6388	-
Mat40s16	-	1852	1685	2090	1941	5535	4690
Mat40s32	2311	1843	2097	2223	1966	5782	4665

The MA with 3 populations find the best result only for ES using a binary structure with 31 individuals in each population (93 individuals in all) and a binary coding with length 16 (MAt31s16). At this point, the MAs with ternary tree structures outperforms the binary structures in 5 out of 7 best results found for the benchmark functions.

Table 4 compares the best results found by the MAs with other methods. These methods are the Continuous Hybrid Algorithm (CHA) [9], Genetic Algorithm for Globally Minimization Function (GAGMF) [10], Continuous Genetic Algorithm (CGA) [11], Continuos Reactive Tabu Search Minimum (CRTS min) and Average (CRTS ave.) [12], and Enhanced Continuous Tabu Search (ECTS) [13]. The GAGMF uses a binary coding with length 8 (GAGMF8b), 16 (GAGMF16b) and 32 (GAGMF32b). The GAGMF has also results using a gray coding with the same binary lengths (GAGMF8g, GAGMF16g and GAGMF32g). The minimization success rates less than 100% are shown in parentheses in Table 4.

Table 4. MA best results compared with other methods.

	B2	DJ	ES	GP	H34	R2	SH
GAGMF8b	-	-	-	891	1968	1554(0.42)	-
GAGMF8g	-	-	-	1085(0.98)	508(0.98)	1764(0.53)	-
GAGMF16b	-	-	-	1997	1742	3675(0.39)	-
GAGMF16g	-	-	-	2748(0.96)	2277(0.99)	5596(0.54)	-
GAGMF32b	-	-	-	2873(0.99)	2860	4834(0.35)	-
GAGMF32g	-	-	-	4008(0.96)	2735(0.99)	5829(0.52)	-
CGA	320	-	-	410	582	960	575
CHA	132	371	**952**	259	492	**459**	**345**
ECTS	210	-	-	231	548	480	370
CRTS min.	-	-	-	171	609	-	-

CRTS ave.	-	-	-	248	513	-	-
MAt4s32[a]	**76**	104	19679	464	5971	118316	58829
MAt4s16[b]	-	**57**	12138	243	3095	71887	29016
MAb31s16[c]	-	1168	1394	1385	1370	5439	6440
MAb7s16[d]	-	85	5686	**170**	852	13398	23612
Mat13s8[e]	-	-	-	-	**395**	7019	-
MAt40s16[f]	-	1737	2033	1982	1802	4597	4722
MAt40s32[g]	2234	1774	2053	2128	1781	5049	4131

The MA approaches have found the best values in functions B2, DJ, GP and H34. These results outperformed the CGA and CHA which use a real coding for individual representation. The MAs with ternary tree structure find the best results in B2, DJ and H34, where B2 and DJ are solved with 1 population and H34 with 2 populations. The best result for GP is found using a binary structure with 1 population. In function R2, MAt40s16[f] outperforms the GAGMF with binary and gray coding with length 32. It also outperforms the GAGMF with gray coding with length 16. MAt40s16[f] does not outperform the others GAGMF results. However, all the GAGMF approaches have less than 60% of minimization sucess rate, while the MAs have 100% of minimization sucess rate. The CGA, CHA and ECTS have a better performance than MAs for R2 and SH. The CGA outperforms the MAb31s16[c] in ES.

5. Conclusion

This paper presented preliminary results using a memetic algorithm (MA) with a population hierarchically structured for global optimization of multiminima functions. A binary coding with lengths 8, 16 and 32 was used as representation of individual. These individuals were hierarchically structured in trees, where binary and ternary structures were evaluated as well as the use of different number of populations.

The computational results showed that populations hierarchically structured were able to outperform other approaches, including Tabu Searches and Genetic Algorithms with binary and real coding. The MAs with ternary structures, using one and two population, returned the best results in 3 out of 7 functions (B2, DJ and H34). The binary structure returned the best result with one population in GP. The MAs did not perform well in Rosenbrock (R2), but had a better minimization success rate than the other GAs binary coding approaches. In Shubert (SH) and Easom (ES) functions, the MAs did not performed well as the other available methods. Based on these insights, an approach using real coding for solutions is been developed and it will be evaluated using the same population structures presented here. Also, the MAs are being executed over other benchmark functions shown in [10] and these results will be reported in future works.

References

1. Floudas, C.A., Akrotirianakisa, I.G., Caratzoulasa, S., Meyera, C.A., Kallrathb, J.: Global optimization in the 21st century: Advances and challenges. Computer & Chemical Engineering 29, 1185--1202 (2005).
2. Horst, R., Pardalos, P.M., Thoai, N.V.: Introduction to Global Optimization, Second Edition. Kluwer Academic Publishers (2000).
3. Gen, M., Cheng, R.: Genetic algorithms & engineering design. John Wiley & Sons New York (1997).
4. Yuan, Q. He, Z., Leng, H.: A hybrid genetic algorithm for a class of global optimization problems with box constraints. Applied Mathematics and Computation 197, 924--929 (2008).
5. Xing, L., Chena, Y., Caia, H.: An intelligent genetic algorithm designed for global optimization of multi-minima functions. Applied Mathematics and Computation 15, 355—371 (2006).
6. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Technical Report Caltech Concurrent Computation Program, Report 826, Caltech, Pasadena, California (1989).
7. Buriol, L.; França, P.M.; Moscato, P.: A new memetic algorithm for the asymmetric traveling salesman problem. Journal of Heuristics 10, 483--506 (2004).
8. França, P.M.; Mendes, A.S.; Moscato, P.: A memetic algorithm for the total tardiness single machine scheduling problem. European Journal of Operational Research 132 (1), 224--242 (2001).
9. Chelouah, R., Siarry, P.: Genetic and Nelder–Mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions. European Journal of Operational Research 148, 335--348 (2003).
10. Berthiau, G., Siarry, P.: A genetic algorithm for globally minimizing functions of several continuous variables. In: Second International Conference on Meta-heuristics, Sophia-Antipolis, France (1997).
11 Chelouah, R., Siarry, P.: A continuous genetic algorithm designed for the global optimization of multimodal functions. Journal of Heuristics 6, 191--213 (2000).
12 Battiti, R., Tecchiolli, G.: The continuous reactive tabu search: Blending combinatorial optimization and stochastic search for global optimization. Annals of Operations Research 63, 53--188 (1996).
13 Chelouah, R., Siarry, P.: Tabu search applied to global optimization. Special issue on combinatorial optimization. European Journal of Operational Research 123(2), 30--44 (2000).
14 Goldberg, D.E.: Genetic Algorithms in search, optimization, and machine learning. Reading, MA: Addison-Wesley (1989).
15 Michalewicz, Z.: Genetic Algorithms + data structure = evolution programs. Springer-Verlag (1996).