Gustavo Pinho Kretzer de Souza

Otimização de funções reais multidimensionais utilizando algoritmo genético contínuo

Florianópolis

2014

Gustavo Pinho Kretzer de Souza

Otimização de funções reais multidimensionais utilizando algoritmo genético contínuo

Monografia submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciências da Computação.

Universidade Federal de Santa Catarina

Departamento de Informática e Estatística

Curso de Ciências da computação

Orientador: Prof. Dr. Mauro Roisenberg

Florianópolis

2014

Gustavo Pinho Kretzer de Souza

Otimização de funções reais multidimensionais utilizando algoritmo genético contínuo

Monografia submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciências da Computação.

Prof. Dr. Mauro Roisenberg
Orientador

Professor
Elder Rizzon Santos

Professor
Ricardo Azambuja Silveira

Florianópolis
2014

RESUMO

Cada vez mais o homem busca o melhor aproveitamento possível dos recursos existentes. Otimização é uma grande área de interesse pois em uma grande gama de aplicações matemáticas é comum desejar encontrar pontos ótimos de funções. O algoritmo genético é uma abordagem estocástica de busca baseado nos conceitos de reprodução e evolução natural de Charles Darwin. Este trabalho objetiva a implementação de um algoritmo genético contínuo e mostrar a sua viabilidade de utilizar o algoritmo genético para a otimização de funções reais com múltiplas variáveis. Inicialmente o algoritmo gera uma população inicial. Nos indivíduos da população aplica-se os operadores genéticos de seleção, cruzamento e mutação adaptados para variáveis reais, de forma a aumentar a eficácia do algoritmo em convergir para soluções aceitáveis. Posteriormente o desempenho do algoritmo é avaliado para diferentes alternativas e parâmetros do algoritmo.

Palavras-chaves: Algoritmos Genéticos. Otimização. Inteligência Artificial. Otimizações de Funções.

ABSTRACT

Increasingly man seeking the best possible use of resources existents. Optimization is a large area of interest as in a large range of mathematical applications is common to want to find great function points. The genetic algorithm is a stochastic search approach based on the concepts of reproduction and natural evolution of Charles Darwin. This work aims the implementation of a continuous genetic algorithm and show the viability of using genetic algorithm to the optimization of real functions with multiple variables. Initially the algorithm generates an initial population. Individuals in the population applies genetic operators of selection, crossover and mutation adapted to real variables, in order to increase the effectiveness of the algorithm to converge to acceptable solutions. Subsequently the performance of the algorithm is evaluated for different alternatives and algorithm parameters.

Key-words: Genetic Algorithms. Optimization. Artificial Intelligence. Optimizations Functions.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AG	Algoritmo Genético
AGC	Algoritmo Genético contínuo
COCO	COmparing continuous Optimisers

SUMÁRIO

1 INTRODUÇÃO

Cada vez mais o homem busca o melhor aproveitamento possível dos recursos existentes. A otimização consiste na busca do melhor entre todos os valores possível para dadas variáveis, para uma função de um determinado objetivo e com as limitações existentes.

Em uma grande gama de aplicações matemáticas, é comum se desejar encontrar o valor máximo ou mínimo de funções, ou seja, encontrar um valor para todas as variáveis da mesma, na qual o valor da função seja o ótimo. Existem várias técnicas para otimizar uma função, como por exemplo o cálculo diferencial Bortolossi (2002), entretanto, para muitas funções pode não ser possível a utilização dessas técnicas.

O foco deste trabalho está no métodos de otimização probabilísticos, chamado algoritmo genético. Entre os métodos de otimização probabilísticos destacam-se os algoritmos evolucionários, os algorítimos genéticos, têmpera Simulada (Simulated Anneling) e o algorítimo de otimização por colônia de formigas (Ant Colony Optimization) entre outros.

Técnicas também comumente utilizadas para otimização de funções é a programação linear, sendo que a principal diferença entre a programação linear e os métodos probabilísticos está no fato que, o último procura encontrar os valores ótimos globais procurando evitar cair em ótimos locais.

Os algoritmos genéticos são métodos de busca baseado no processo biológico de evolução e seleção natural de Charles Darwin Linden (2012), onde as características favoráveis tornam-se mais comuns nas gerações sucessivas e as características desfavoráveis tendem a desaparecer. No algoritmo genético é mantida uma população de possíveis soluções do problema, sendo essas avaliadas por uma função de aptidão. Ao longo de várias gerações são geradas novas populações a partir da população anterior, sendo esses novos indivíduos fruto do cruzamento e mutações das soluções anteriores, este processo iterativo vai-se evoluindo as soluções, convergindo com uma certa probabilidade para o máximo global Norvig (2012).

O AG, originalmente proposto por John Holland na década de 70 era adaptado para a solução de problemas combinatoriais discretos, devido a maneira como os parâmetros do problema eram codificados nos "cromossomos"dos indivíduos que compunham a população e pela maneira como os operadores de cruzamento e mutação eram definidos. Entretanto, muito problemas práticos do mundo real tem natureza contínua, isto é, os parâmetros que regem o problema são números reais e desta forma é necessário que o algoritmo genético seja adaptado para ter um bom desempenho nesta classe de problemas.

O propósito deste trabalho é desenvolver um software que implemente os mecanismo do algoritmo genético contínuo. Sendo que a importância desta implementação servirá para testar o conceito de que se é possível resolver problemas de otimização contínuos utilizando técnicas baseadas na evolução natural de forma que o mesmo seja reconfigurável e também reutilizável.

É neste contexto, que este trabalho propõe uma investigação dos algoritmos genéticos contínuos, voltados para a solução de problemas combinatoriais cujo espaço de busca são os números reais, uma implementação e testes com diferentes parâmetros no algoritmo para um medida de desempenho do mesmo para diferentes funções.

1.1 Objetivos

O trabalho a ser realizado, diante do exposto, apresenta os seguintes objetivos:

1.1.1 Objetivo Geral

Desenvolver um algoritmo genético contínuo e realizar testes para avaliar o desempenho do mesmo com diferentes funções e parâmetros.

1.1.2 Objetivos Especificos

- Analisar os conceitos teóricos de algorítimos de otimização de meta-heurísticas.
- Implementar o algoritmo genético em uma linguagem de programação.
- Analisar as estratégias efetuadas pelo algoritmo ao longo da execução.
- Comparar o desempenho para as estratégias e funções utilizadas.

1.2 Metodologia

Este trabalho iniciou-se com a abordagem da importância dos problemas de otimização em várias áreas de pesquisa e algumas das principais ferramentas para resolver problemas de otimização.

Após realizado estudo de como a teoria da evolução e seleção natural biológica foi adaptado para resolver problemas de otimização, criando assim a área dos algoritmos evolutivos e algoritmos genéticos(AG). Foi construído um referencial teórico demonstrando o funcionamento de um AG, com a representação cromossômica e seus operadores para algoritmos genéticos contínuos (AGC).

Posteriormente houve a implementação do AGC utilizando a linguagem de programação Octave juntamente com a ferramenta COCO (COmparing contínuous Optimisers)

com a finalidade de comparar e testar as suas funções. Com o código desenvolvido, o mesmo foi testado e avaliado o desempenho conforme os parâmetros estabelecidos.

1.3 Organização do trabalho

Os capítulos seguintes são organizados da seguinte forma. O capitulo 2 apresenta o que são problemas de otimização, a sua importância em resolvê-los, a seção 2.2 expões a classe de algoritmos de meta-heurísticas de otimização. O capítulo 3 apresenta o algoritmo genético genérico, e vários conceitos importantes para o seu entendimento. O capítulo 4 exibe as particularidades dos algoritmos genéticos contínuos. O capítulo 5 apresenta o projeto de implementação de um AGC. O capítulo 6 apresenta os resultados dos testes utilizando o AGC. E por fim, o capítulo 7 são as considerações finais de todo o trabalho.

2 PROBLEMAS DE OTIMIZAÇÃO

> Otimizar é melhorar o que já existe, projetar o novo com mais eficiência e menor custo. A otimização visa determinar a melhor configuração de projeto sem ter que testar todas as possibilidades envolvidas. Saramago (2012)

Problemas de otimização são encontrados praticamente em todas as áreas que se possa construir modelos matemáticos que representam um sistema dinâmico, onde possua alguma variável que se possa maximizar, assim resultando no melhor desempenho do sistema. Um dos princípios das técnicas de otimização é encontrar a melhor configuração de uma função, ou uma solução próxima da melhor, sem precisar testar todas as configurações possíveis.

Os métodos para a solução de problemas de otimização são divididos em dois grupos, as técnicas clássicas e os métodos aleatórios. Os métodos aleatórios dividem-se ainda de acordo com suas variáveis, podendo o mesmo ser contínuos ou discreto Saramago (2012).

Técnicas clássicas de otimização foram desenvolvidas com o cálculo diferencial e integral, a partir da álgebra e geometria. Essas técnicas são confiáveis e possuem uma vasta área de aplicação como nas engenharias e outras ciências, entretanto para determinadas funções existem alguns impedimentos em utilizá-las, como por exemplo:

> falta de continuidade das funções a serem otimizadas ou de suas restrições, funções não convexas, multimodalidade, existência de ruídos nas funções, necessidade de se trabalhar com valores discretos para as variáveis, existência de mínimos ou máximos locais (PRADO, 2005)

Por esse motivo que métodos aleatórios são ferramentas importantes para a resolução de muitos problemas práticos de otimização.

Os Problemas de otimização discretos, que também são conhecidos como otimização combinatória, ou problemas de conjunto finito são problemas onde é possível enumerar todas as soluções. Porém para a maioria dos problemas não é praticável a realização de uma busca exaustiva, pois são problemas de complexidade intratáveis. Para contornar isso, são utilizados vários métodos não exatos, que mesmo não existindo uma garantia de encontrar a melhor solução, normalmente apresentam bons resultados Linden (2012).

2.1 Problemas de otimização continuo

Problemas de otimização contínuos, são aqueles em que as variáveis são continuas, ou seja, onde se desejar minimizar ou maximizar uma determinada função numérica

normalmente com várias variáveis, em algum contexto que possa existir restrições. O máximo global é definido como (STEWART, 2008)

> Uma função f tem máximo absoluto (ou máximo global) em c se $f(c) \geq f(x)$ para todo $x \in D$, onde D é o domínio de f. Analogamente, f tem um mínimo absoluto em c se$f(c) \leq f(x)$ para todo $x \in D$, e o número $f(c)$ é denominado valor mínimo de $f \in D$.

Ou seja, o objetivo é encontrar esse valor c. A figura 1 mostra os máximos locais e o máximo global de uma função.

Figura 1 – Máximos locais e global

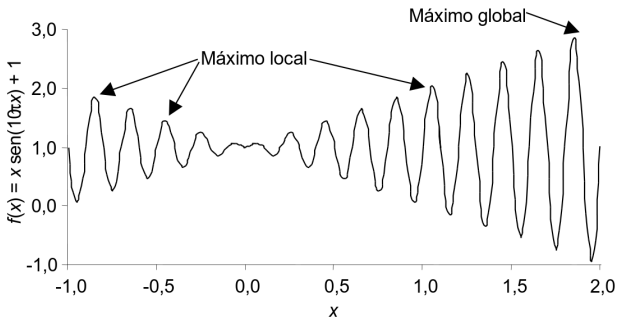


Gráfico da função $f(x) = x(sen(10\pi x)) + 1$, Fonte: (CARVALHO, 1999)

2.2 Algoritmos de meta-heurísticas de otimização

A palavra heurísticas que tem origem no grego heuriskein que significa descobrir, são procedimentos fundamentado em experiência para resolver problemas, que indica uma solução que não tem garantia que a mesma seja ótima ou seja é um conhecimento momentâneo, que não são verificáveis, são aplicados onde a busca exaustiva é impraticável. São métodos que devem ter baixo custo computacional e que encontram uma solução satisfatória(não exata) do problema.

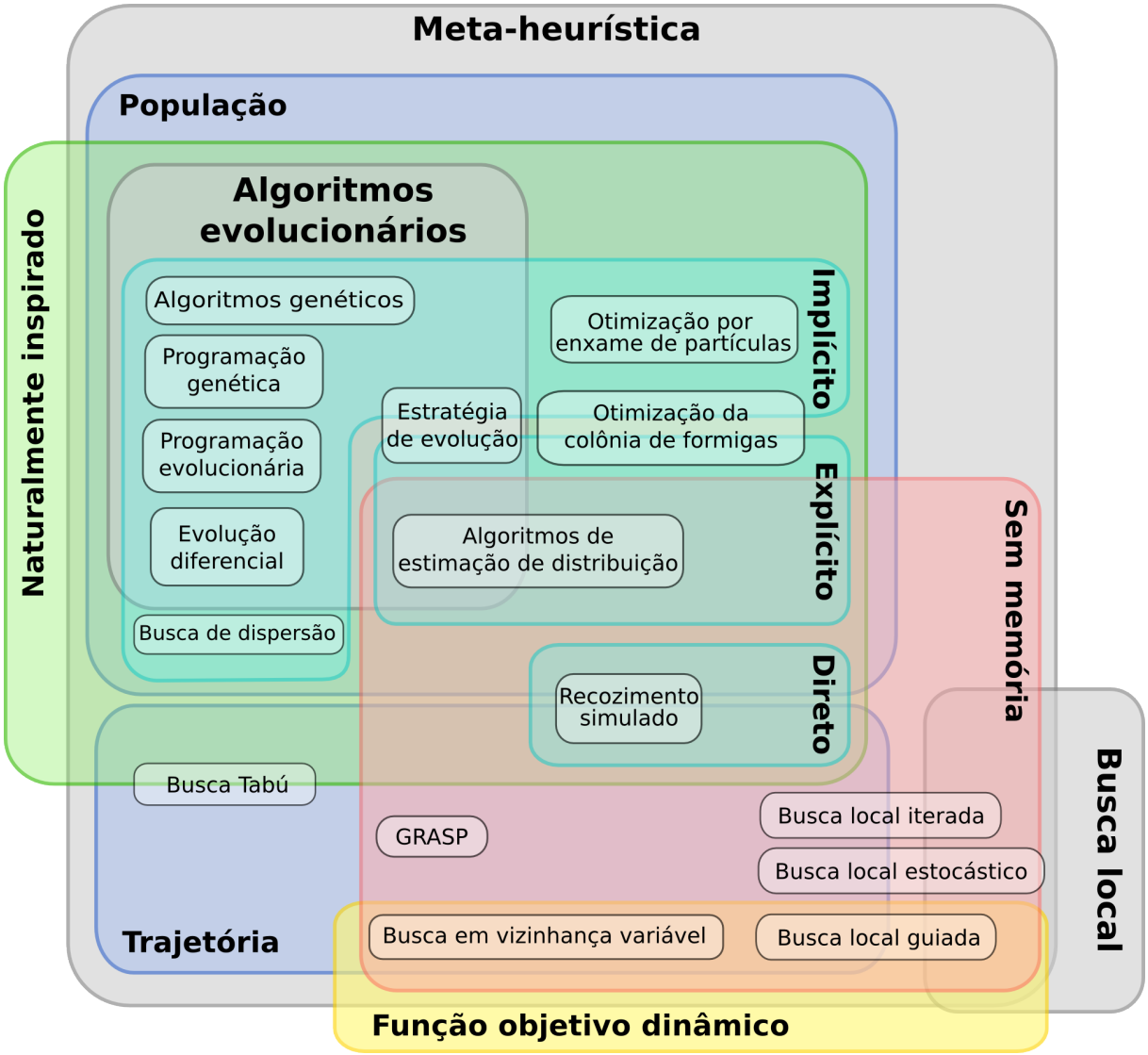
Meta-heurísticas são métodos heurísticos para resolver problemas de otimização de forma genérica, geralmente são utilizados quando não se conhece algum algoritmo eficiente para resolver tais problemas. Esses algoritmos utilizam a aleatoriedade e conhecimento histórico dos resultados anteriores para se guiarem pelo espaço de busca. De acordo com a definição meta-heurística é

> Um processo de geração iterativo que guia uma heurística subordinada por combinação de forma inteligente de diferentes conceitos para explorar e explotar o espaço de busca, aprendendo estratégias que são usadas para estrutura da informação na ordem de encontrar eficientemente uma solução próxima da ótima. (SHAMSUDDIN, 2013)

Existem várias classificações para os algorítimos de meta-heurísticas, conforme a figura 2. Entre eles os inspirados na natureza como o colônia de formiga, os algorítimos

genéticos e tempera simulada e os não inspirados na natureza como a busca tabu. São classificados também como baseado em população contra os de ponto único de busca, ou seja, classificamos as algoritmos de meta-heurísticas de acordo com o número de soluções utilizadas ao mesmo tempo, se o mesmo funciona utilizando mais de uma solução, esse algoritmo é tido como baseado em população, caso trabalha com apenas uma solução são algorítimos de ponto único (SHAMSUDDIN, 2013).

Figura 2 – Classificação de algoritmos de meta-heurística.



Fonte: (SHAMSUDDIN, 2013, Pag.4).

Os algoritmos baseado em população são propostos de modo que cada solução interaja localmente um com outros e com o ambiente. As soluções seguem regras, mas sem uma estrutura centralizada para controlar o seu comportamento, com a iteração entre as partículas e o ambiente são conduzidos a um comportamento inteligente global. Esses algoritmos possuem duas propostas, a exploração global e explotação local(SHAMSUDDIN, 2013).

A exploração ou busca global é a força para expandir o espaço de busca em regiões ainda desconhecidas. Por exemplo, uma pesquisa randômica é uma estratégia que explora o espaço ignorando se a mesma é uma região promissora ou não. Enquanto que a explotação ou busca local utiliza informações presentes no espaço de busca já explorado para encontrar soluções ótimas próximas de boas soluções. Um exemplo de estratégia de busca local é o método de subida de encosta (Hill climbing).

Para um bom desempenho do algoritmo deve haver uma equilíbrio entre a exploração e a explotação, pois caso a exploração for muito forte o algoritmo se comportara de forma semelhante uma busca totalmente aleatória(random walk), e se tiver muita explotação, o algoritmo terá um resultado semelhante aos métodos de subida de encosta (SHAMSUDDIN, 2013).

3 ALGORITMOS GENÉTICOS

Algoritmo genético é um método que simula o processo de evolução natural (biológica), destinando-se a solucionar problemas de otimização. O AG pode ser entendido como uma interpretação matemática e algorítmica das teorias de Darwin, no qual as hipóteses podem ser resumida: A evolução é maneira que age sobre os cromossomos do organismo, e não sobre o organismo que os transporta. Logo o que ocorrer durante a vida do organismo não altera os seus cromossomos, no entanto, os cromossomos refletem diretamente nas características do organismo. A seleção natural faz com que os cromossomos que criam organismos mais bem adaptado ao ambiente, sobrevivam e reproduzam mais que outros organismos com cromossomos menos adaptados. A reprodução é o ponto no qual a evolução se define. O processo de recombinação dos cromossomos (entre dois pais) podem gerar cromossomos dos filhos bem diferente dos pais.

Algoritmos genéticos é uma técnica heurística de otimização global utilizada para encontrar soluções aproximadas em problemas de otimização. Este algoritmo utiliza técnicas que foram inspiradas pela biologia evolutiva como seleção natural, hereditariedade, mutação e recombinação. AG não é um algoritmo de busca de solução ótima, mas sim uma heurística que encontra boas soluções, já que o mesmo utiliza da aleatoriedade, pode encontra soluções diferentes a cada execução, podendo esses serem ótimo locais ou globais Linden (2012).

O AG utiliza população de indivíduos, sendo que um individuo é uma representação abstrata de uma solução para o problema. Realizando uma analogia com a biologia, o gene é a parte indivisível da representação, e todos os genes compõem um cromossomo. A representação cromossômica deve ser adequada para cada problema, deve poder representar o problema de modo que tenha uma precisão desejada e outras particularidades da arquitetura da maquina ou da linguagem de programação utilizada. A representação deve ser o mais simples possível e que preferencialmente não represente soluções ilegais ao problema.

Em analogia com a natureza, a informação é codificada em cromossomos e a reprodução (sexuada) e a mutação se encarregará de fazer a população evoluir. A reprodução (crossover) é o processo que irá criar filhos baseado na mistura entre o cromossomos de dois ou mais pais. Ha vários modos de efetuar a reprodução, e o mesmo é dependente de como foi codificado o cromossomo.

Mutações são alterações nos genes, que podem ser causadas por erros de copia do material genético durante a divisão celular. Na natureza ocorrem com uma baixa probabilidade. Essa mesma pode ser benéfica, gerando um individuo com uma melhor

aptidão, ou maléfica reduzindo a aptidão do individuo. No AG as mutações são importantes para aumentar a diversidade de soluções na população, e evitar uma convergência genética muito rápida, ela é importante para evitar máximos locais.

Para que a população possa evoluir para uma boa solução, deve-se selecionar os melhores indivíduos da população para gerarem novos indivíduos. Sendo que os pais mais aptos tende a gerar mais filhos, e pais menos aptos gerarem menos descendentes. Como funciona na seleção natural onde as características favoráveis que são hereditárias se tornaram mais comum nas gerações sucessivas, e as características desfavoráveis que são hereditárias se tornam mais raras, ou até mesmo extintas. Essas características são preservadas em consequência da vantagem seletiva que concedem a seus portadores, permitindo que um indivíduo deixe mais descendentes que outros sem essas características.

Nos AG, a adaptabilidade do individuo é calculado pela função de avaliação, essa função tem como entrada o cromossomo e gera a avaliação do individuo. Na seleção, os indivíduos que tiverem uma avaliação alta devem ser privilegiado em detrimento aos que tiverem uma avaliação ruim. É importante que todos os indivíduos possam ser selecionado, pois indivíduos com uma baixa avaliação pode ter alguma característica que seja propícia a melhor solução do problema. Caso apenas os melhores indivíduos evoluírem, em cada geração a população tenderá a ser constituída por indivíduos cada vez mais semelhantes, consequentemente reduzindo a diversidade da população, gerando o efeito da convergência genética Linden (2012).

A convergência genética prematura não é desejável, pois o GA pode ter convergido para um ótimo local, sem antes ter realizado uma busca em uma região maior do espaço de busca. A convergência prematura pode ser causada por algum super individuo, que gere um número excessivo de filhos, ou pela alta pressão de seleção.

O primeiro AG proposto por Holland é conhecido como Standart Genetic Algorithm ou Simple Genetic Algorithm é descrito em 6 passos.

1 - Inicie uma população com os cromossomos gerados aleatoriamente, de tamanho N.

2 - Avalie cada cromossomo da população utilizando a função de avaliação.

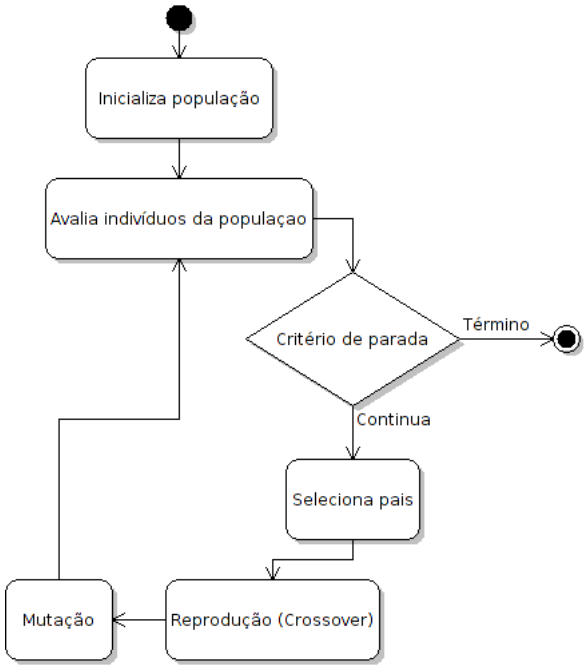
3 - Gere uma nova população de tamanho N a partir do cruzamento de cromossomos selecionados da população anterior. Aplique mutação nestes cromossomos.

4 - Remova a população anterior, trocando pela nova população criada.

5 - Avalie cada cromossomo da população utilizando a função de avaliação.

6 - Caso a solução ideal seja encontrado, ou o tempo do algoritmo se esgote (número máximo de gerações, ou avaliações, ou algum outro critério de parada), retorna o cromossomo com a melhor avaliação. Caso contrario retorne para o passo 3.

Figura 3 – Diagrama de sequencia AG



Fonte: Autor (2014)

Se tudo ocorrer bem, esta simulação do processo evolutivo produzirá, conforme a população for evoluindo, os cromossomos melhor adaptados (com uma melhor avaliação), irão obter uma boa solução para o problema tratado.

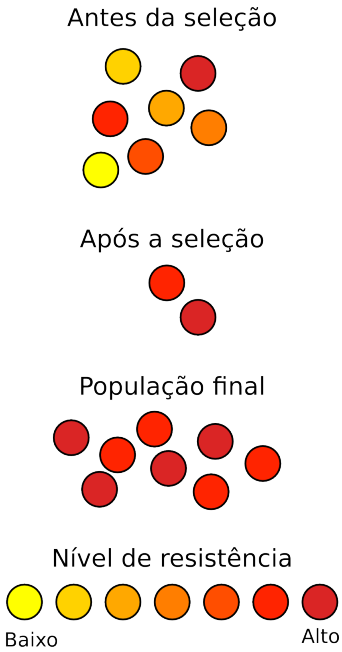
3.1 Teoria da Evolução e seleção natural

A evolução biológica é a mudança das características hereditárias de uma população de uma geração para uma geração seguinte. Neste método acarreta a diferenciação dos organismo com o tempo. A evolução é alguma mudança em um ou algum conjunto de genes, em uma determinada população, apos inúmeras gerações. Neste processo podem ocorrer mutações nos genes, onde são capazes de acarretar características novas ou modificar características já existentes. Estas novas características também podem aparecer através da troca de genes entre populações por consequência da migração.

A seleção natural é a maneira na qual as características hereditárias que auxiliam para a sobrevivência e reprodução se tornem mais comuns em uma população, a medida que as características desfavoráveis se tornam mais raras. Isto acontece pois indivíduos com características favoráveis tem uma maior probabilidade de sucesso na reprodução, como resultado os indivíduos na próxima geração herdarão essas características. Ao longo de várias gerações, adaptações acontecem ao longo de combinações de pequenas mudanças frequentes, pequenas e aleatórias nas características, mas expressivas em conjunto.

Um exemplo de como a seleção natural funciona é o caso do desenvolvimento de resistência a antibióticos em bactérias, como podemos ver na figura 4. Nas populações naturais de bactérias, existe uma grande variedade de material genéticos, e quando expostos a antibióticos, a maioria das bactérias morrem, mas algumas possuem genes que fazem a mesmas serem resistentes aos antibióticos. As bactérias sobreviventes irão se reproduzir e gerar uma nova geração resistente aos antibióticos, devido a eliminação dos indivíduos menos resistentes.

Figura 4 – Um exemplo de seleção natural, resistencia de bactérias a antibíoticos.



Uma representação esquemática de como a resistência antibiótica é desenvolvida pela seleção natural. Na primeira seção é representado uma população de bactérias antes de serem expostos a antibióticos. Na segunda seção exibe a população após a exposição dos antibióticos. Na 3 seção mostra a resistência da nova geração de bactérias. E a legenda o nível de resistência dos indivíduos ao antibióticos. Fonte da imagem: http://pt.wikipedia.org/wiki/Ficheiro:Antibiotic$_r esistance - pt.svg$

3.2 AG binários

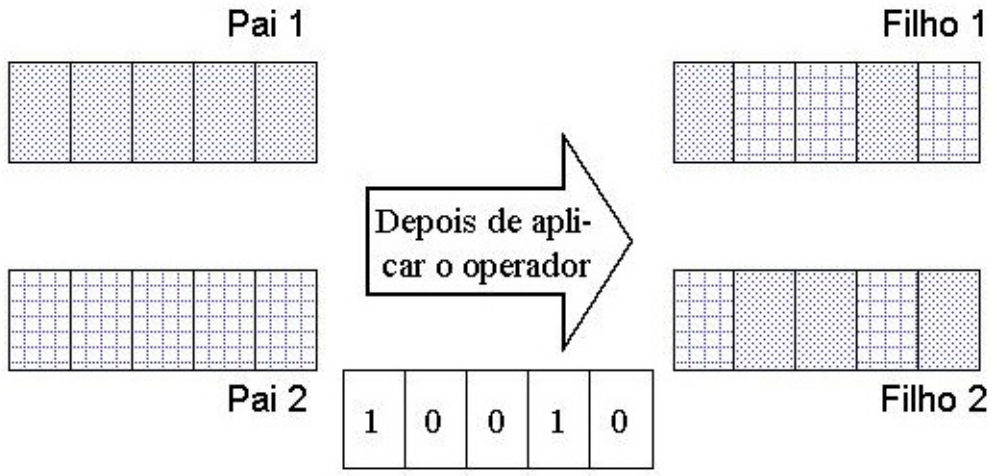
O primeiro AG desenvolvido por John Holland na década de 1970 foi proposto para a resolução de problemas combinatoriais discretos. A representação do cromossomo é binária, ou seja, um cromossomo é uma sequência de bits e cada gene é um bit. O que cada bit ou um conjunto de bits representam é inerente ao problema.

Um operador de cruzamento usualmente utilizado para cromossomos binários é o *cruzamento de 1 ponto*. O ponto do corte constitui uma posição entre dois genes de um cromossomo. Após sorteado o ponto de corte, separamos os pais em duas partes, uma

à esquerda do ponto de corte e outro a direita. Então o primeiro filho é formado pela concatenação da parte esquerda do primeiro pai e pela parte à direita do segundo pai. E o segundo filho é formado pela concatenação da parte esquerda do segundo pai com a parte à direita do primeiro pai.

Outro operador de crossover para AG discreto é o *cruzamento uniforme*. Considerando que na natureza a formação cromossomial de espécies que utiliza a reprodução sexuada, não se limita a apenas um ponto de corte, podendo assim termos o mesmo número de pontos de corte quanto ao número de genes. Para realizar o cruzamento uniforme geramos aleatoriamente uma string binária de combinação. Conforme a figura 5 o primeiro filho recebe os genes do primeiro pai em todas as posições onde foi sorteado 1 e o gene do segundo pai em todas as posições em que foi sorteado 0. o segundo filho é criado com os genes que não foram utilizados na criação do primeiro filho.

Figura 5 – Funcionamento do cruzamento uniforme no AG binário.

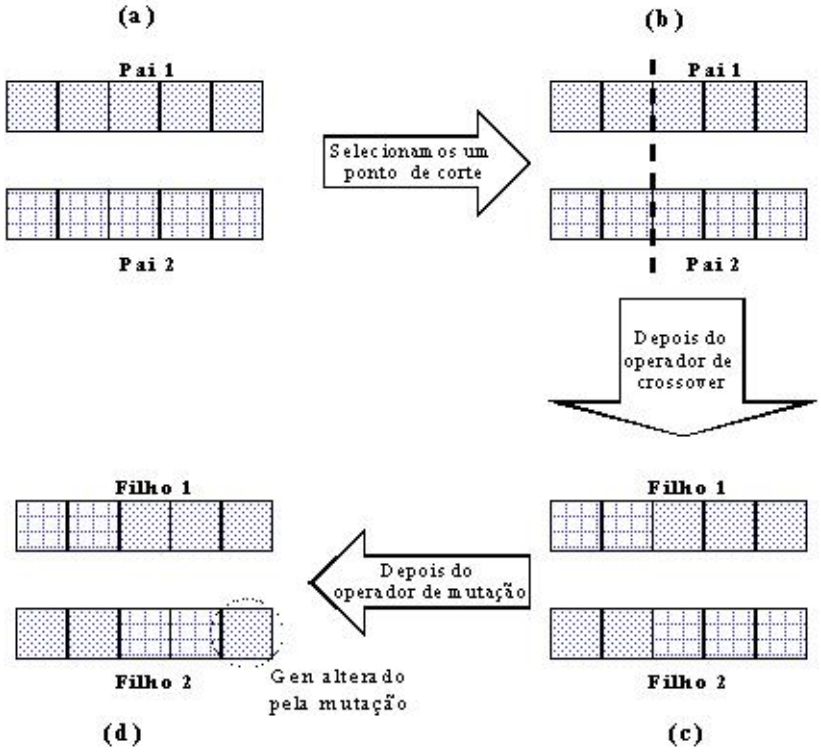


Funcionamento do cruzamento uniforme. Fonte: (LINDEN, 2012) Adaptado pelo autor.

O operador de *mutação* em AG binários, após a geração dos filhos, com uma probabilidade definida pela usuário, troca o valor do gene. Como o alfabeto de um AG binário tem apenas dois valores, então operador de mutação simplesmente inverte o valor do gene mutado. A figura 6 descreve a operação dos operadores de *cruzamento de 1 ponto* e a mutação.

Para uma grande gama de problemas combinatoriais discretos o AG binário produz bons resultados, mas para problemas de otimização reais os mesmos apresentam vários problemas como descrito em 4.1 . Para isso a representação do cromossomo e os operadores de cruzamento e mutação devem ser adaptados.

Figura 6 – Diagrama de sequencia AG



Fonte: (LINDEN, 2012) Adaptado pelo autor.

3.3 Convergência Prematura

Um fenômeno que é observado comumente é um AG convergir muito rapidamente para um ótimo local, mas não necessariamente um ótimo global. Esse fenômeno é chamado de *convergência prematura*. Se temos um individuo próximo a um ótimo local, que não seja o ótimo global, que tenha uma avaliação muito alta em relação ao restante da população, em consequência ao processo de seleção, este mesmo terá muitos descendentes na próxima geração. Este individuo é chamado de super-indivíduo. Em casos extremos o resto da população será extinta, restando apenas esse super-indivíduo. O que normalmente resulta na convergência da população para um ótimo local, não necessariamente global.

A *convergência prematura* está correlacionado com a *perda de diversidade* da população. A *diversidade* é um conceito que indica o quão variada é a população, ou seja o quanto que as regiões do espaço de busca está refletida na população.

3.4 Pressão Seletiva

Um conceito importante ao projetar um GA para o entendimento do processo de busca é a pressão seletiva, em oposição da diversidade populacional.

A pressão seletiva é um conceito que está conectado à direção e velocidade que o

AG vai assumir no espaço de busca. Caso não houvesse a pressão seletiva, o comportamento do AG seria equiparável a uma busca aleatória, onde não existe sentido nem direção. A pressão seletiva associa a vantagem de alguns indivíduos para sobreviver em detrimento dos outros. Se a probabilidade de seleção de alguns indivíduos de serem selecionados forem muito diferentes dos outros, diz-se que possui uma pressão seletiva forte. A pressão seletiva é fraca quando a diferença de probabilidade de seleção entre os indivíduos é pequena, ou seja a mesma está bem distribuída na população.

Quando aumenta a pressão seletiva, acelera a convergência do AG para algum ponto máximo, que pode ser global ou local. Caso os cromossomos que a pressão seletiva esteja beneficiando estejam próximo de uma máximo global então a pressão seletiva tem um característica favorável, já que o AG irá terminar mais rápido o seu processo de busca. Entretanto caso os cromossomos não se localizarem próximo a um máximo global, mas de máximos locais, então o aumento da pressão seletiva pode ser danoso pois fará que os indivíduos deslocar-se inconvenientemente para uma área de um ótimo local.

Para um bom desempenho de um AG, deve haver um equilíbrio entre a Pressão seletiva e a diversidade populacional.

3.5 Função de avaliação

A escolha de umas função de avaliação é fundamental para cada problema tratado. Em problemas de maximização de uma função onde o seu domínio são sempre valores positivos, por simplificação pode-se utilizá-la como a função de avaliação. Mas isso nem sempre é uma boa estratégia.

A relação entre a *função objetivo* e a *função avaliação* deve ser feito com cautela. Neste relacionamento os indivíduos que tem uma melhor função objetivo devem também possuir uma melhor avaliação. Vários métodos de seleção exigem que os valores da função de avaliação tenham sempre valores positivos, como por exemplo o *método da roleta* descrito na seção 3.5.1.1.

No caso da função objetivo possuir valores negativos, podemos transformá-los todos em positivos somando um valor mínimo em todos os valores da função objetivo. Em problemas de minimização transformamos o mesmo em problemas de maximização apenas trocando o sinal da função objetivo.

Para tratar o problema da escala dos valores das funções, sendo que esses influenciam diretamente na pressão seletiva caso utilizamos esses valores diretamente na roleta, foram desenvolvidas várias técnicas que são abordados na seção 3.5.1, como o *escalonamento sigma*(3.5.1.2), *método de ranking*(3.5.1.3) e a *seleção por torneio*(3.5.1.4).

3.5.1 Operador de seleção

O operador de seleção define quais os indivíduos que serão selecionados para participar da reprodução e gerar a nova geração. Esse método deve simular o mecanismo de seleção natural que age nas espécies biológicas Linden (2012). O seu objetivo é privilegiar os indivíduos com avaliações altas, mas sem desprezar completamente os com avaliações baixas.

A operação de seleção é feita baseado no valor retornado pela função de aptidão de cada individuo. Existem vários métodos diferentes para realizar a seleção, cada um com suas vantagens e desvantagens.

Caso tivermos uma seleção muito forte, onde os indivíduos com alta avaliação tiverem uma chance muito maior que os de baixa avaliação de serem selecionados, os indivíduos com as melhores avaliações irão prevalecer muito rapidamente na população, reduzindo a diversidade necessária para o bom desempenho do AG. Por outro lado se tivermos uma seleção muito fraca, onde os indivíduos com avaliação baixa tiverem uma chance de serem selecionados muito próxima dos com alta avaliação, o algoritmo terá uma evolução muito lenta, tomando muito tempo de CPU, ou sem um resultado final satisfatório Melanie (1996).

3.5.1.1 Método da roleta

O AG original de Holland utiliza o *método da roleta* Melanie (1996), onde cada individuo pega uma proporção da roleta conforme a sua avaliação. No método da roleta tradicional, a chance de cada indivíduo ser selecionado é a sua avaliação dividido sobre a soma da avaliação de todos os individuo.

Sendo f_i a função de avaliação do individuo i, N o número total da população e $S = \sum_1^N f_i$ então cada cromossomo terá chance de f_i/S de ser selecionado, sendo também este o tamanho do setor alocado para o cromossomo na roleta. O algoritmos da roleta é descrito a seguir:

1 - Gera um número uniformemente variado entre zero e um.

2 - Soma-se o valor do setor alocado de cada cromossomo, verificando se este valor não excede do valor inicialmente sorteado.

3 - Quando o valor da soma passar do valor inicialmente sorteado, será o cromossomo escolhido.

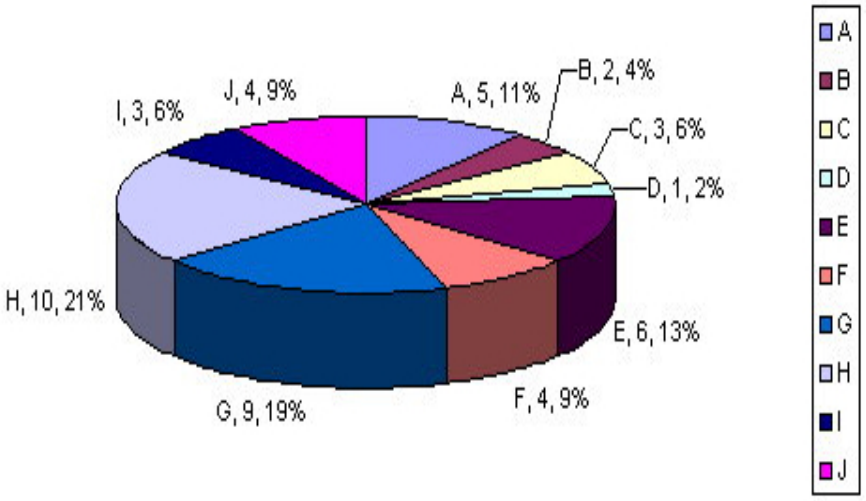
Evidentemente quando maior o setor na roleta, também será maior a probabilidade deste cromossomo ser escolhido.

Os cromossomos da tabela 3.5.1.1, são ilustrado na figura 7.

Um problema do método da roleta é que a mesma é não aceita valores negativos,

Cromossomo:	A	B	C	D	E	F	G	H	I	J
Avaliação:	5	2	3	1	6	4	9	10	3	4

Figura 7 – Roleta viciada.



Roleta viciada com 10 cromossomos, cada cromossomo ocupa um espaço na roleta. Fonte: (RUBICITE, 2014)

então a função de aptidão deve ser projetada de modo que retorne apenas valores positivos.

Esse método em alguns casos pode fazer que o desempenho do AG se degenere Linden (2012). Com o surgimento de super indivíduos, que é quando um ou mais indivíduos possuem uma avaliação muito superior aos outros membros da população. Neste caso esses super indivíduos serão quase sempre escolhidos, ocasionando uma perda muito rápida na diversidade genética nas gerações seguintes.

Outro problema é nos casos onde ocorre uma diferença pequena entres as avaliações, ocasionando que quase todos os indivíduos tem uma chance de ser escolhido quase igual. Sendo que mesmo essas pequenas diferenças nas funções de avaliação expressam uma notável diferença na qualidade das soluções Linden (2012).

3.5.1.2 Escalonamento Sigma

Esses dois problemas são resultado de que o método da roleta é sensível a escala dos valores das função de aptidão. Para resolver esse problema foram desenvolvidos métodos de escalonamento. Entre esses métodos, temos o escalonamento Sigma, que mapeia os valores de avaliação bruto em um valor que é menos susceptível a uma convergência prematura Melanie (1996).

$$E(i,t) = \begin{cases} 1 & \sigma(t) = 0 \\ 1 + \frac{f(i) - \overline{f}(t))}{2\sigma(t))} & \sigma(t) \neq 0 \end{cases} \tag{3.1}$$

Sendo $f(i)$ a avaliação do individuo, $\overline{f}(t)$ a avaliação média da população, e $\sigma(t)$ o desvio padrão das avaliações, todos no instante t.

No inicio o desvio padrão da população tende a ser muito alto, por consequência da inicialização aleatória, os indivíduos mais aptos não deverão dominar rapidamente a população (convergência genética), pois com um desvio padrão alto, a pressão seletiva tende a ser baixa. Conforme for passando as gerações, os indivíduos tende a convergir para uma região do espaço de busca, ficando mais próximos entre si e com valores de avaliações mais próximos, fazendo o desvio padrão cair, e assim aumentado a chance de os melhores indivíduos serem selecionados, aumentado a força da seleção Linden (2012).

3.5.1.3 Método do Ranking

O *método do ranking* ou também chamado de *normalização linear* Linden (2012) é um método alternativo com o propósito de prevenir a dominância de super-indivíduos e uma convergência prematura Melanie (1996). Este método mantêm a pressão seletiva no mesmo nível da primeira até a última geração.

Nesse método, a população é colocada em ordem decrescente pelo valor de avaliação, e então a seleção se baseia na posição do individuo, e não mais do valor de avaliação. Com isso tem-se a vantagem de eliminar o problema de convergência prematura, mas a desvantagem, que muitas vezes é bom saber que um individuo tenha uma avaliação muito melhor que o seu vizinho Melanie (1996).

No método do ranking cada individuo recebe uma avaliação conforme a equação 3.2. Definidos os valores de avaliação dos indivíduos, é utilizado o método da roleta(3.5.1.1) para a seleção.

$$E(i,t) = Min + (Max - Min) * \frac{rank(i,t) - 1}{N - 1} \tag{3.2}$$

Sendo, Min o valor mínimo que será concedido ao individuo com a menor avaliação, Max o valor máximo que será concedido ao individuo com a melhor avaliação, N o tamanho da população, e $rank(i,t)$ a posição do indivíduo i na população ordenada conforme a avaliação, da menor para a maior avaliação, na geração t. A figura 8 8 exemplifica o cálculo de avaliação dos indíviduos pelo ranking.

3.5.1.4 Seleção por torneio

O método de seleção por ranking pode ser computacionalmente custoso demais, pois além de ter que calcular todas as avaliações, ainda tem que ordenar os indivíduos. O método de *seleção por torneio* é similar ao método de seleção por ranking em termos de pressão seletiva, mas é computacionalmente mais eficiente Melanie (1996). O método consiste em escolher de forma aleatória dois ou mais indivíduos da população, o que tiver o

Figura 8 – Tabela seleção ranking

Indivíduo	Avaliação bruta	Roleta %
x1	120	2,44
x2	15	0,31
x3	30	0,61
x4	100	2,03
x5	1000	20,35
x6	250	5,09
x7	5	0,10
x8	350	7,12
x9	45	0,92
x10	2500	50,86
X11	500	10,17

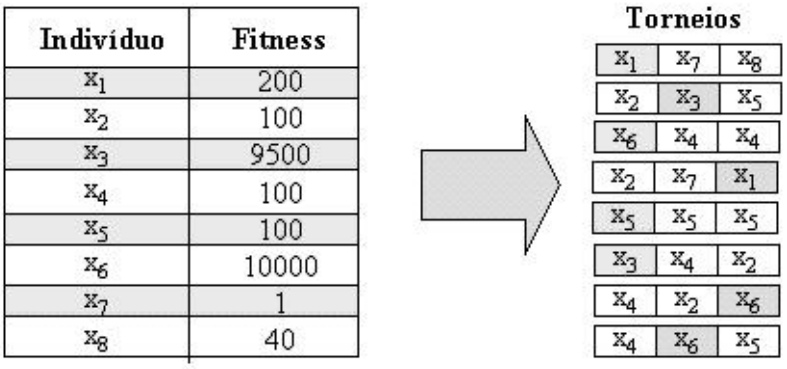
Ordenando os indivíduos:
x7, x2, x3, x9, x4, x1, x6, x8, x11, x5, x10

Indivíduo	Avaliação ranking	Roleta %
x1	1,5	9,09
x2	1,1	6,67
x3	1,2	7,27
x4	1,4	8,48
x5	1,9	11,52
x6	1,6	9,70
x7	1	6,06
x8	1,7	10,30
x9	1,3	7,88
x10	2	12,12
X11	1,8	10,91

Na primeira tabela a "avaliação bruta"dos indivíduos, com o percentual na roleta de cada indivíduo, utilizando o método de avaliação por ranking com $Min = 1$ e $Max = 2$, e os valores de avaliação são calculados para a coluna avaliação ranking, alterando drasticamente o percentual da roleta dos indivíduos. Fonte da imagem: Autor(2014).

melhor avaliação é selecionado. Outra forma de seleção por torneio, para que os indivíduos com avaliações menores possam também serem escolhidos é realizado um sorteio entre os escolhidos, da seguinte forma. Escolhe um valor $k > 0.5$, (por exemplo $0,9$), e então um número n é sorteado, caso $n < k$, é selecionado o individuo com melhor avaliação, caso contrario o de menor avaliação é escolhido. A figura 9 exemplifica a aplicação do método de torneio para $k = 3$.

Figura 9 – Exemplo de seleção por torneio, utilizando 3 individuos



Fonte: (LINDEN, 2012, Pag.205)

3.6 Parâmetros dos AG

O desempenho de um AG é fortemente influenciado pela definição dos parâmetros utilizados. Não existe um conjunto ótimos de parâmetros a ser utilizado que possua um melhor desempenho para todas os problemas tratados. Um conjunto de parâmetros pode funcionar bem para determinado problema, mas pode ter um desempenho inferior para

outro. Por isso é importante analisar como os parâmetros influenciam no comportamento dos AGs, para que se possa estipular os mesmos conforme as exigências do problema.

3.6.1 Tamanho da população

Pela intuição quando maior o tamanho da população N mais fácil o AG irá encontrar a solução, já que se tivermos uma população infinita cobrindo todo o espaço de busca, obteríamos a ponto ótimo já na primeira geração, mas por questões prática temos que trabalhar com populações finitas.

O tamanho da população afeta a eficiência e o desempenho global dos AGs. Utilizando uma população pequena o desempenho do AG poderá cair, pois a população fornece uma pequena cobertura do espaço de busca, podendo assim não explorar regiões promissoras, portanto possui uma probabilidade menor de encontrar o ótimo global. Entretanto uma população muito grande proporciona uma cobertura mais representativa do domínio do problema, e previne convergências prematuras para ótimos locais, no entanto populações grandes demandam mais recursos computacionais, e um maior processamento entre gerações, já que todos os indivíduos terão que ser avaliado em cada geração, além do overhead das operações de seleção, cruzamento e mutação.

Por questão de simplificação e facilidade de implementação, a maioria dos AG possuem tamanho fixo de população. Por costume, população de tamanho entre 50 a 200 indivíduos resolvem a maioria dos problemas, mas pode ser preciso populações maiores para problemas mais complexos Tanomaru (1995).

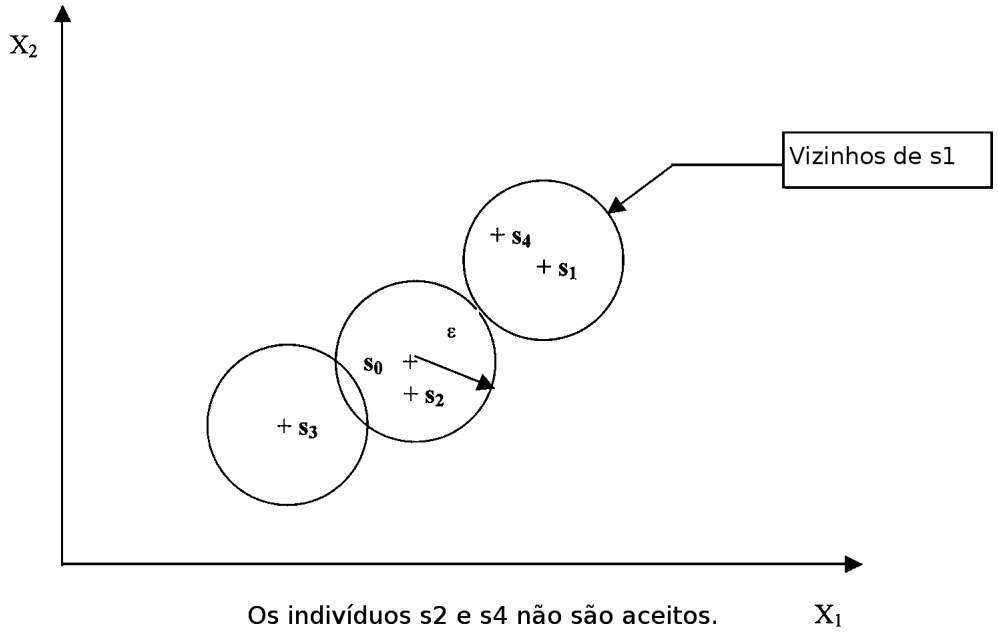
Não existe uma maneira de precisar o melhor tamanho para uma população. Podemos resolver esse problema é utilizar alguma estratégia em que a população varie com o tempo. Entre as estratégias utilizadas temos a *baseada na variabilidade genética da população* e *baseada na idade dos indivíduos* Linden (2012).

3.6.1.1 Geração da população inicial

Uma modo simples de inicializar população é simplesmente atribuindo um valor qualquer dentro do espaço de busca para cada gene do individuo. Por ser um método aleatório, podemos gerar muitos indivíduos numa determinada área de busca e outras áreas ficarem vazias, assim prejudicando a busca global. Para cobrir homogeneamente o espaço de busca e não correr o risco de termos muitos indivíduos numa mesma região. Conforme (SIARRY, 2000) definimos uma área de vizinhança para cada cada individuo gerado. Essa vizinhança é definida como uma região, sendo essa região um circulo $B(s, \varepsilon)$ centralizado em s e com o raio ε, contendo todos os pontos s tal que $|s' - s| <\leq \varepsilon$. Um individuo gerado é aceito na população inicial, se ele não estiver na vizinhança de nenhum outro indivíduo já selecionado, exemplo na figura 10 para duas dimensões, mas esse conceito

pode ser estendido para qualquer número de gerações. Na figura 11 mostra o resultado da distribuição inicial de uma população de 30 indivíduos.

Figura 10 – Área de vizinhança de um indivíduo



Área da vizinhança nos indivíduos da população inicial. Fonte: (SIARRY, 2000) Adaptado pelo autor.

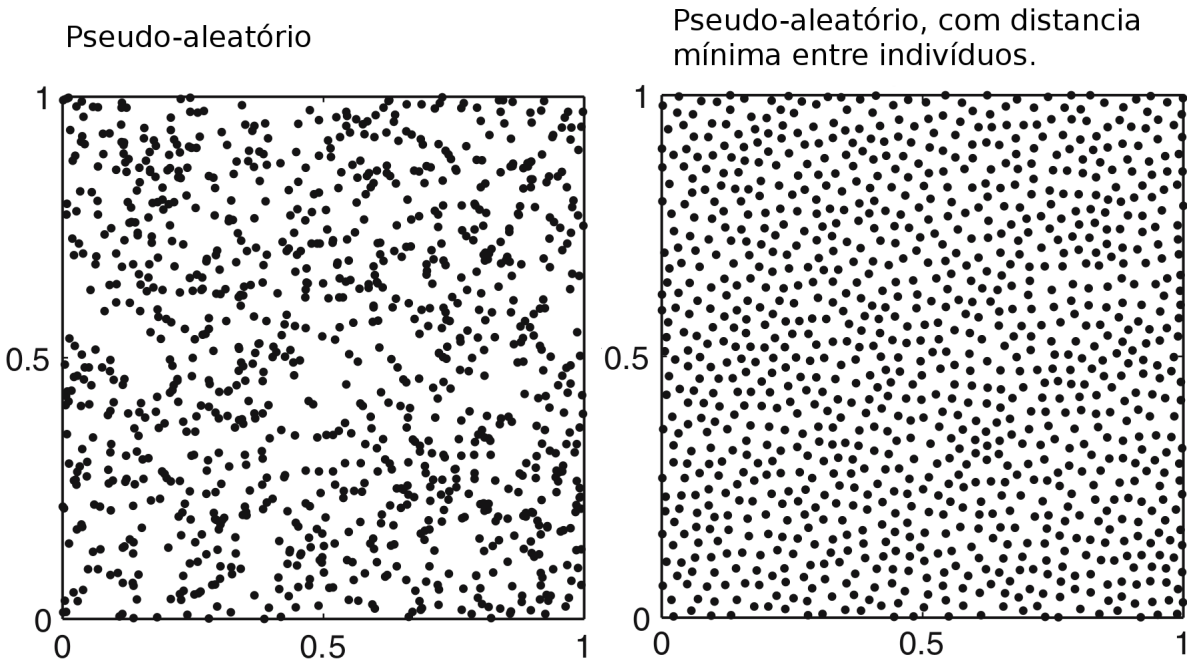
3.6.2 Estratégia Elitista

A *estratégia elitista* consiste na troca de gerações, selecionar k individuo(s) ($k \geq 1$) que tiverem a melhor avaliação e passá-los para a próxima geração, de modo que os seus cromossomos serão preservados nas gerações subsequentes. Com isso garantimos que o melhor individuo da nova geração será no mínimo igual a da geração anterior.

O elitismo normalmente colabora para um melhor desempenho do AG. Pois mantêm dentro da população sempre os melhores indivíduos, o que melhora o aproveitamento da explotação, sem prejudicar o desempenho da exploração. Sendo k o número de indivíduos selecionado pelo elitismo, e n o tamanho da população. já que os $n - k$ indivíduos não serão afetados, considerando que k seja muito menor que n.

O processamento desta técnica é pequeno, já que a avaliação dos indivíduos já é realizada para a seleção, e basta selecionar os k melhores indivíduos daquela geração. Normalmente o número de indivíduos mantidos de uma geração para a outra é de um ou dois indivíduos, podendo assim praticamente desconsiderar o processamento extra.

Figura 11 – Distribuição inicial de uma população



Distribuição inicial de uma população no espaço de busca. Da esquerda utilizando um gerador pseudo-aleatório, da direita respeitando a área de vizinhança. Fonte: (PENTTINEN, 2007) Adaptado pelo autor.

3.6.3 Taxa de mutação

A mutação previne que a população fique estagnada em uma região do espaço de busca. Essa taxa não pode ser muito alta, pois a busca se tornara aleatória, reduzindo o desempenho global do AG.

3.6.4 Condição de parada

Se tratando de problemas de otimização, o ideal seria se parar a execução assim que encontrar o ótimo global. Entretanto na maioria dos problemas não o conhecemos, já que o propósito dos AG é justamente encontrá-los. No caso de funções multimodais, encontrar um ponto ótimo já pode ser o suficiente. Na pratica é difícil afirmar com certeza se um ponto ótimo é um ótimo global. Em consequência disto, deve haver um critério de parada do AG, como um número máximo de gerações ou número máximo de avaliação de função.

Também pode ser utilizado como critério de parada a *estagnação*. A estagnação ocorre quando após um determinado de gerações seguidas não se observa melhora significativa da população(ou do melhor individuo), ou verificar se houve a convergência genética que é a perda de variabilidade cromossômica. Nesses casos podemos reiniciar o AG, ou interromper a execução sem necessariamente ter obtido exito na busca.

4 ALGORITMOS GENÉTICOS CONTÍNUOS

O primeiro AG desenvolvido utiliza uma representação binaria em seu cromossomos, sendo historicamente importante pois foi utilizado nos trabalhos pioneiros de John Holland em 1975 sobre AG Carvalho (1999). O AG binário funciona muito bem quando a variável do problema for quantizada. Quando o espaço de busca do problema for contínuo (representado em $\Re$) é mais coerente representá-las utilizando uma representação de números em ponto flutuante. De acordo com Linden (2012) utilizar cromossomos reais torna igual o genótipo (representação interna) com o fenótipo(representação externa ou valor utilizado no problema). Acabando com os efeitos de interpretação entre as duas representações.

4.1 Representação Real X Binária

Se utilizarmos AG discretos para problemas reais, a representação de cada váriavel é um conjunto de bits do cromossomo. Como a representação interna do cromossomo é diferente das variável do problema, tendo que realizar uma conversão entre as representações. A figura 12 exemplifica a decondificação de um cromossomo binário de 44 bits em dois parâmetros no intervalo $[-100, 100]$.

Um problema de utilizar a codificação binária para parâmetros reais é que não existe uniformidade nos operadores. No operador de mutação, o parâmetro será mais afetado se o bit trocado for o mais significativo, do que se fosse um menos significativo.

Além disso ao utilizar representação real, não precisamos realizar a conversão entre os parâmetros do problema com a representação cromossômica, facilitando assim a criação de operadores de crossover e mutação.

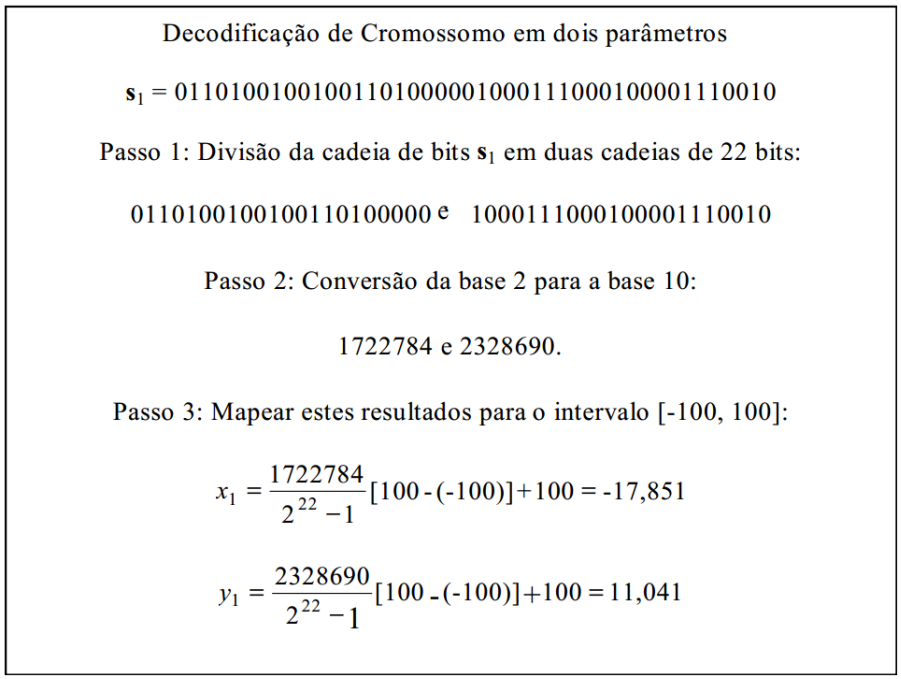
4.2 Representação do cromossomo

No AGC em um problema de N-dimensões os cromossomos são representados N_{var} variáveis, em uma matriz de $1xN_{var}$ elementos, dado pelas variáveis $p_1, p_2, ..., p_{Nvar}$.

$$cromossomo = [p_1, p_2, ..., p_{Nvar}] \tag{4.1}$$

Sendo, $p_1, p_2, ..., p_{Nvar}$ valores das variáveis que representam números em pontos

Figura 12 – Decodificação de um cromossomo binário em dois parâmetros reais.



Fonte: (CARVALHO, 1999)

flutuantes. Essa avaliação deve ser utilizado para calcular a função de avaliação.

$$avaliação = f(p_1, p_2, ..., p_{Nvar}) = f(cromossomo) \tag{4.2}$$

Por exemplo se desejamos otimizar a seguinte função:

$$\frac{\sin\sqrt{x^2+y^2}}{\sqrt{x^2+y^2}} \tag{4.3}$$

Sendo a mesma de 2 dimensões então o avaliação da mesma será:

$$cromossomo = [x, y] \tag{4.4}$$

O avaliação da mesma será:

$$Avaliação = f(x, y) = \frac{\sin\sqrt{x^2+y^2}}{\sqrt{x^2+y^2}} \tag{4.5}$$

Sujeito a $-5 <= x <= 5$ e $-5 <= y <= 5$.

Haupt (2004)

As variáveis do AGC deveriam poder representar qualquer valor real, mas como o mesmo é implementado em computadores reais, existe um número finito que podemos representar para nossa variável, esse valor é limitado pela arquitetura do hardware, ou por

alguma estrutura de dados que represente números em ponto flutuantes(por exemplo, com o uso de bibliotecas como a biblioteca GMP (GMP, 2013)). Logo a precisão numérica das variáveis tem que ser compatível com a precisão desejada para o problema tratado.

4.3 Operadores de cruzamento contínuo

O *cruzamento* é uma característica fundamental para os AG, pois é no processo de cruzamento que novos indivíduos são criados com a troca de material genético dos seus pais. Essa operação é análoga a reprodução sexuada na natureza. Após a escolha de dois pais, os seus genes são misturados e são gerado dois filhos. Nos AGC há vários modos diferentes de realizar o cruzamento. O cruzamento é dependente da forma como o cromossomo é representado, por esse trabalho ser focado em AGC, serão abordados apenas cruzamento para representações numéricas reais. Existe vários outros método para outras representações, como a binária, baseado em ordem, entre outras.

4.3.1 Cruzamento Simples

O método de *cruzamento simples*, escolhe um ou mais pontos de corte no cromossomo, e as variáveis entre estes pontos são simplesmente trocadas entre os pais. Os filhos serão uma mistura dos genes dos dois pais. O problema desse método é que nenhuma informação nova é gerada . Cada valor que foi criado randomicamente na população inicial é propagado para as próximas gerações, apenas em uma diferente combinação. Essa é uma estratégia que funciona bem para a representação binária, mas para valores contínuos apenas troca dois valores, deixando toda a responsabilidade de criar novo material genético pela mutação. Haupt (2004)

Para remediar esse problema é utilizado os operadores de *cruzamento aritméticos*, nos quais realizam algum tipo de combinação linear entre os cromossomos pais (CARVALHO, 1999).

4.3.2 Cruzamento média

No *cruzamento média* é realizado a média entre os dois valores dos pais, para gerar o novo uma única variável para o filho. Sendo p_{i1}e p_{i2} os cromossomos pais, e c o cromossomo filho e i um gene do cromossomo. Para a média aritmética é expressa na equação 4.6.

$$c_i = (p_{i1} + p_{i2})/2 \tag{4.6}$$

Também pode ser utilizado a média geométrica, conforme a equação 4.7.

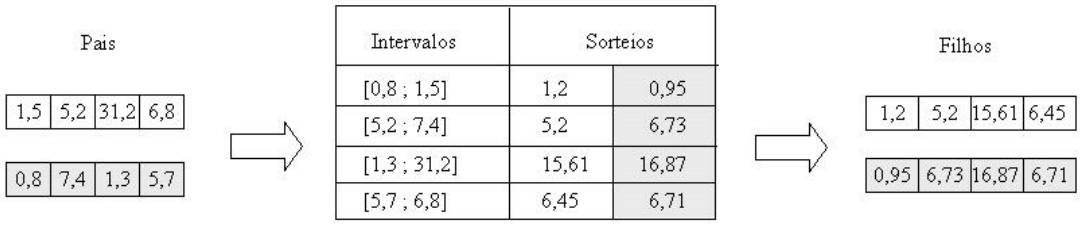
$$c_i = \sqrt{(p_{i1} * p_{i2})} \tag{4.7}$$

O cromossomo média induz os genes para o meio do intervalo entre os genes, acarretando perda de diversidade. Carvalho (1999).

4.3.3 Cruzamento flat

Para evitar essa perda de diversidade, pode-se utilizar o *cruzamento flat*, que se baseia em construir um intervalo fechado, entre os dois ou mais de valores do gene de cada cromossomo, sendo esse intervalo do menor valor até o maior valor dos genes do pai, e sortear uma valor aleatório dentro desse intervalo. A figura 13 exemplifica o cálculo dos filhos pelo operador flat.

Figura 13 – Exemplo da operação do cruzamento flat.



Fonte: (LINDEN, 2012, Pag.261)

4.3.4 Cruzamento Aritmético

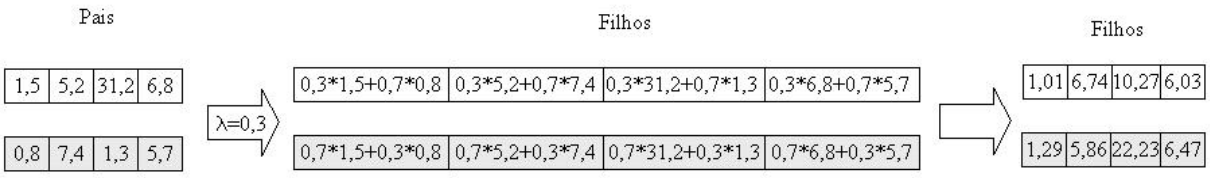
O *cruzamento aritmético* é realizado utilizando uma média ponderada entre os genes dos pais Linden (2012), é definido um valor $\gamma \in [0, 1]$ e calcula-se os filhos conforme as equações a seguir:

$$c_i^{filho1} = \gamma c_i^{pai1} + (1 - \gamma) c_i^{pai2} \tag{4.8}$$

$$c_i^{filho2} = \gamma c_i^{pai2} + (1 - \gamma) c_i^{pai1} \tag{4.9}$$

Sendo que γ pode ser um valor fixo escolhido de forma arbitrária ou por sorteio respeitando $\gamma \in [0, 1]$. A figura 14 exemplifica a geração de dois filhos utilizando crossover aritmético.

Figura 14 – Cruzamento aritmético



Cruzamento aritmético, exemplo utilizando parâmetro $\gamma = 0.3$. Fonte: (LINDEN, 2012, Pag.262)

4.3.5 Cruzamento Linear

No *cruzamento linear* é calculado três filhos conforme as formulas 4.10, 4.11 e 4.12 Linden (2012). Para que mantenha o tamanho da população(dois pais gerando dois filhos), avalia-se os três filhos e o pior deles é descartado, deixando os outros dois com melhor avaliação.

$$c_i^{filho1} = \frac{c_i^{pai1}}{2} + \frac{c_i^{pai2}}{2} \tag{4.10}$$

$$c_i^{filho1} = \frac{3 * c_i^{pai1}}{2} - \frac{c_i^{pai2}}{2} \tag{4.11}$$

$$c_i^{filho1} = -\frac{c_i^{pai1}}{2} + \frac{3 * c_i^{pai2}}{2} \tag{4.12}$$

4.3.6 Cruzamento Blend

O cruzamento $BLX - \alpha$, ou *cruzamento blend*, calcula o filho c, a partir dos pais p_1 e p_2 conforme a equação 4.13

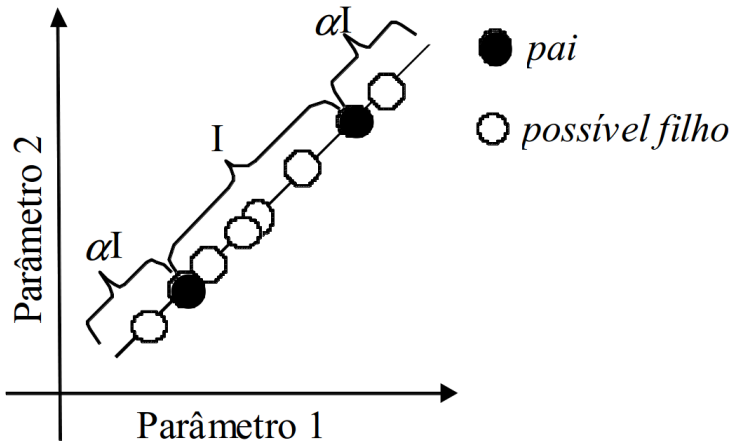
$$c = p_1 + \beta(p_2 - p_1) \tag{4.13}$$

Onde $\beta(-\alpha, 1 + \alpha)$.

A figura 15 ilustra os possíveis filhos caso o β for igual para todos os genes, que é o caso unidimensional. O caso multidimensional é quando o β for diferente para cada gene, é ilustrado na figura 16 . No caso unidimensional, se $\alpha = 0$ os filhos serão gerados sempre dentro do seguimento da linha I entre os pontos dos pais. O parâmetro α estende o seguimento I para cada extremo. Para o caso multidimensional o BLX-α gera os filhos de modo randômico entre em um hiper-retângulo que é definido pelos pontos de seus pais. O BLX-0.5 é frequentemente utilizado Lacerda (2003), pois probabilisticamente metade dos filhos serão gerados dentro e outra metade fora o seguimento da linha I ou do

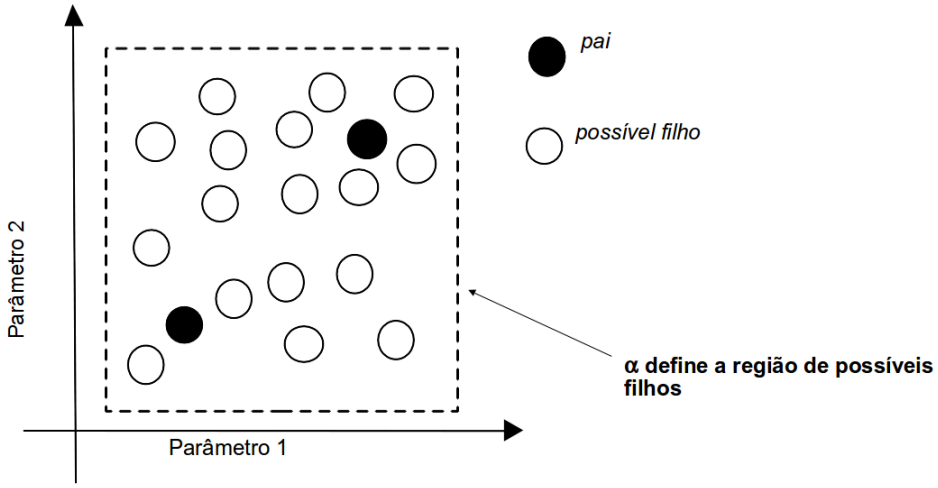
hiper-retângulo, para os casos uni e multidimensionais respectivamente. O BLX-α tem sido aplicado com sucesso em muitos problemas, e é possivelmente um dos cruzamento mais utilizados Lacerda (2003).

Figura 15 – Cruzamento BLX-α unidimensional



Fonte: (CARVALHO, 1999)

Figura 16 – Cruzamento BLX-α multidimensional.



Fonte: (CARVALHO, 1999)

4.3.7 Cruzamento Heurístico

O *cruzamento heurístico* é um cruzamento baseado em direção, que utiliza a informação da função de avaliação ou do gradiente para determinar a direção da busca.

Por exemplo, se temos uma $f()$ que é a função de avaliação que desejamos minimizar, o filho é gerado conforme a equação 4.14 Lacerda (2003).

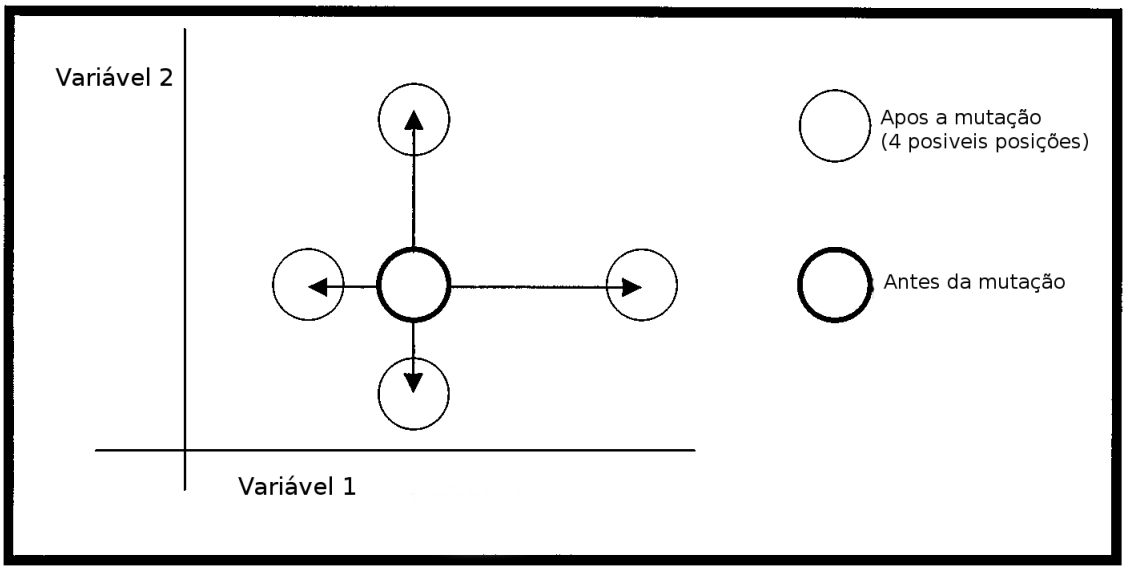
$$c = \begin{cases} p_1 + r(p_1 - p_2)) & se f(p_1) \leq f(p_2)) \\ p_1 + r(p_2 - p_1)) & se f(p_1) > f(p_2)) \end{cases} \tag{4.14}$$

4.4 Operadores de Mutação Contínuos

O operador de mutação é aplicado apos a geração dos filhos, para cada gene, com uma baixa probabilidade, o gene tem seu valor trocado. O operador de mutação que atua sobre o gene, troca por outro valor aleatório conforme o operador de mutação, podendo variar um pouco ou complemente o valor do gene. A figura 17 ilustra o efeito da mutação em um indivíduo com variáveis reais.

O conceito fundamental que o valor da probabilidade de ocorrência de mutação deve ser baixo. Não é possível determinar um valor de probabilidade padrão, sendo que esse deve ser ajustado para cada problema. Caso o valor de mutação for muito alto, o algoritmo genético terá um comportamento parecido com a técnica de random walk, em que a solução é determinada de forma aleatória Linden (2012).

Figura 17 – Efeito da mutação em um indivíduo.



Efeito de uma mutação para de um valor real num indivíduo de duas variáveis. Fonte: (SIARRY, 2000) adaptado pelo autor.

4.4.1 Mutação Uniforme

A *mutação uniforme*, troca o valor de um gene por um número aleatório uniforme no intervalo $[a_i, b_i]$ aceito pelo gene. Se queremos mutar o k-ésimo gene p do cromossomo,

então o filho c será calculado conforme 4.15.

$$c_i = \begin{cases} U(a_i, b_i) & i = k \\ p_i & i \neq k \end{cases} \tag{4.15}$$

4.4.2 Mutação não Uniforme

O operador de mutação pode causar uma grande variação na posição, o que é uma boa estratégia exploratória. Entretanto, após inúmeras gerações quando a população já estiver convergido para boas soluções, a mutação deve ser menos agressiva. Logo um operador de mutação será mais agressivo nas primeiras gerações, para que faça uma boa exploração, mas que ao término a variação seja pequena, para que resulte um ajuste fino na solução, ou seja, que a valor fique próximo ao valor corrente. Por esse motivo é interessante que o operador de mutação se ajuste conforme a população for convergindo para uma boa solução. Linden (2012)

Uma modo de que a mutação tenha um comportamento exploratório no inicio da execução e um comportamento de ajuste fino no seu fim é a aplicação do operador de *mutação não uniforme*. O valor do gene $c_i^{'}$ que está recebendo a mutação é calculado conforme a equação 4.16.

$$c_i^{'} = \begin{cases} c_i + \Delta(t, sup_i - c_i) & \tau = 0 \\ c_i - \Delta(t, c_i - \inf_i) & \tau \neq 0 \end{cases} \tag{4.16}$$

Sendo τ um valor aleatório binário, t a geração corrente, c_i a coordenada que está sofrente a mutação, sub_i e inf_i os valores mínimo e máximo que o gene i suporta, e δ é calculado conforme 4.17.

$$\Delta(t, y) = y * (1 - r^{(1 - \frac{t}{gmax})^b}) \tag{4.17}$$

Sendo y o valor máximo da mutação, g_{max} o número de gerações máximo, r um valor aleatório entre 0 e 1, e b um parâmetro que controla a dependência da mutação com o número de gerações, sendo que quando maior esse valor, mais rápido o δ vai para zero. Linden (2012)

4.4.3 Mutação Gaussiana

A *mutação gaussiana*, troca o valor de um gene p, por um valor aleatório conforme a distribuição normal Lacerda (2003), com média no valor que será trocado, e com o

desvio padrão, geralmente referente aos mesmos k-ésimos genes da população. O filho c é calculado conforme a equação 4.18.

$$c_i = \begin{cases} N(p_i, \sigma) & i = k \\ p_i & i \neq k \end{cases} \tag{4.18}$$

4.4.4 Mutação Limite

A *mutação limite* troca o valor de um gene p por um dos limites dos intervalos permitido pela gene mutado. Essa mutação é utilizada para evitar perda de diversidade para filhos gerados a partir de cruzamento que trazem os genes para o centro do intervalo. Sendo $[a_i, b_i]$ o intervalo permitido para o gene i e $r \in U(a_i, b_i)$. A mutação limite é calculado conforme a equação 4.19.

$$c_i = \begin{cases} a_i & se \quad r < 0.5 \\ b_i & se \quad r >= 0.5 \end{cases} \tag{4.19}$$

5 PROJETO

A implementação do AG foi realizado utilizando a linguagem de programação Octave, já que não existe nenhuma ferramente implementada neste linguagem de um AGC para resolver problemas genéricos de otimização no domínio dos reais. E além de ser uma linguagem de programação interpretada de alto nível, que se destina ao tratamento de problemas para computação numérica, desenvolvida utilizando a filosofia OpenSource.

5.1 COCO - COmparing continuous Optimisers

O COCO (COmparing continuous Optimisers, em português, comparador de otimizadores contínuos) (COCO, 2013) é uma ferramenta de análise (benchmarking) de teste de caixa preta para algoritmos de otimização contínuos. O COCO facilita o experimento dos otimizadores contínuos, além de dispor de um framework experimental para testar os algoritmos, também possui um ferramenta de pós-processamento de imagens e tabelas com os resultados obtidos.

O COCO possui 24 funções sem ruído com parâmetros reais, com um único alvo(um único mínimo global), todas elas definido em $\Re^D$, enquanto o seu domínio de busca é $[-5, 5]^D$. Todas essas funções possuem um ótimo em $[-5, 5]^D$, a dimensão de todas as funções são escaláveis. Além das funções sem ruídos, o COCO também possui outras 30 funções com ruído, porém estas não foram utilizadas neste trabalho.

Por questões de simplificação, para o teste foram utilizadas apenas 9 funções que foram selecionadas de acordo com o resultados obtidos em um pré-teste, devido ao grande tempo demandado para efetuar os testes por completo.

Os testes realizados tiveram como objetivo comparar o desempenho entre cada parâmetro no conjunto das funções. Os parâmetros foram, tamanho da população, tipo de seleção, tipo de cruzamento e tipo de mutação.

A seguir serão expostos os resultados obtidos a partir da presente pesquisa com diferentes parâmetros do AG.

5.2 Fluxo Principal

Como o objetivo deste trabalho é tratar problemas onde todas as variáveis são valores reais, o cromossomo foi representado conforme descrito na seção 4.2. Na implementação o cromossomo é representado como uma matriz coluna, sendo que cada elemento da linha é

um gene e cada coluna representa um indivíduo. Uma população inteira é uma matriz de tamanho $[n, m]$, sendo n o número de genes e m o número total de indivíduos.

A base do algoritmo é implementada no arquivo GA.m. Inicialmente gera uma população inicial com os valores de cada gene gerados de forma pseudoaleatória numa distribuição uniforme no intervalo $[-5, 5]$, conforme (ROS, 2013). Todas as funções testadas neste trabalho são definidas em $\Re^D$ e seus ótimos globais estão em $[-5, 5]^D$. Com a população inicial gerada a mesma é avaliada pelo COCO.

O COCO além de avaliar as funções, guarda os valores das mesmas para que após a execução do algoritmo possa avaliar o seu desempenho e gerar tabelas e gráficos para melhor comparar o desempenhos dos otimizadores testados.

Após a geração da população inicial e de sua avaliação, os pais são selecionado em pares, e então realizado a operação de cruzamento para gerar um par de filhos a partir do cromossomo de dois pais selecionados anteriormente. Os filhos gerados são colocados numa nova população, até que essa nova população tenha o tamanho da população anterior, menos o número de indivíduos que serão repassados da população anterior para a nova por elitismo. Para cada novo indivíduo gerado, o mesmo é submetido pelo operador de mutação, podendo ter ou não algum dos valores dos seus genes alterados. A estratégia de seleção e os operadores de cruzamento e mutação são definidos previamente pelo usuário.

E então é selecionado os k melhores indivíduos da população anterior, e são adicionados na nova população. O valor k deve ser sempre par, por questão de simplificação da implementação, já que os novos indivíduos são sempre gerados em pares.

Como utilizamos o COCO, que é uma ferramenta para testes, já é conhecido o ótimo global, conforme o (ROS, 2013) é considerado que o algoritmo encontrou o ponto ótimo se a diferença da avaliação do melhor individuo com o ótimo global é menor que 10^{-8}. Ou seja $f_{alvo} = f_{opt} + \Delta f$. Sendo f_{opt} o ótimo da função, definido no teste do indivíduo, Δf a precisão, e f_{alvo} o valor que a função deve chegar. Caso o AG não atinga uma precisão desejada apos um número máximo de avaliações pré-definida, o mesmo é interrompido.

5.3 Operadores

Os operadores de seleção, cruzamento e mutação foram implementadas nos arquivos selecao.m, crossover.m e mutacao.m respectivamente. Cada módulo possui vários tipos de operadores diferentes, podendo escolher o tipo que se deseja utilizar sem ter que alterar o fluxo principal no AG.m.

5.3.1 Seleção

Na seleção, sempre que se tem uma nova população, a seleção deve ser inicializada passando o tipo "inicializa"para que os variáveis utilizados nas seleções sejam calculadas. Nos testes do COCO sempre se deseja encontrar o valor mínimo das funções. Como a roleta deve dar um espaço maior para as melhores avaliações(neste caso as menores), então invertemos o sinal de todas as avaliações. Mas também temos que ter valores positivos, já que a roleta não funciona com os mesmos negativos. Para isso a inicialização também trata de deixar os valores positivos. Com os valores brutos tratados, a inicialização calcula os valores de avaliação para serem utilizados nos métodos sigma e ranking.

Na seleção tipo roleta, é realizada conforme descrito em 3.5.1.1 com os valores brutos tratados.

Na seleção tipo sigma e ranking utiliza o método da roleta, mas com os valores de avaliação calculados conforme 3.5.1.2 3.5.1.3 respectivamente.

O torneio é realizado com k indivíduos, sendo $k \geq 2$ conforme 3.5.1.4.

5.3.2 Cruzamento

No operador de cruzamento são gerados os filhos são gerados em pares a partir do cromossomos de dois pais. Foram implementados os operadores *simples*, *flat*, *aritmético* e *cruzamento blend*, conforme descrito em 4.3.1, 4.3.3, 4.3.4 e 4.3.6 respectivamente.

5.3.3 Mutação

O operador de mutação percorre todos os genes de um indivíduo e todos os indivíduos da população, sendo que para cada gene, gera um número pseudoaleatório, caso este seja menor que a probabilidade de ocorrência de mutação, o valor daquele gene é trocado por outro conforme o tipo da mutação.

Foram implementados os tipos de mutação, *uniforme*, *não uniforme*, *gaussiana*, e *limite*, conforme descrito em 4.4.1, 4.4.2, 4.4.3 e 4.19 respectivamente.

5.4 Experimento

O arquivo experimento.m é onde se define cada experimento. Como o AG é um algoritmo estocástico, não conseguimos determinar o seu desempenho com apenas um único teste. Precisando assim executar o AG várias vezes para obtermos um desempenho médio com a semente do gerador de números pseudoaleatórios diferente, ou com instancias diferentes do mesmo problema. Para manter o mesmo padrão proposto pelo COCO, foram testadas com 15 instancias diferentes do para cada função e dimensão. No experimento.m

configurado para que os testes as funções números 1,2,3,7,15,16,17,18,19,20 definidas em (AUGER, 2013) para 2, 3, 5 e 10 dimensões.

6 EXPERIMENTOS E RESULTADOS

Esse capítulo apresenta os resultados dos experimentos realizados, e uma interpretação dos mesmos. Para comparar o quanto afeta cada operador no desempenho global do AGC, para cada grupo de testes foram utilizados os mesmos parâmetros, exceto os parâmetros os que se desejam comparar.

6.1 Resultado Cruzamento

Foram testados os cruzamento *aritmético* 4.3.4, *flat* 4.3.3 e o $BLX-\alpha$ com $\alpha = 0.5$ 4.3.6 foram utilizadas a população fixa de 250, seleção por torneio, e mutação aleatória com taxa de 5%.

Conforme os resultados da figura 18 o cruzamento BLX-alfa teve um desempenho ligeiramente superior aos demais, o que é esperado, pois o mesmo evita que os genes dos filhos convirjam para o centro dos genes dos pais como o caso do *cruzamento aritmético*.

6.2 Resultado Seleção

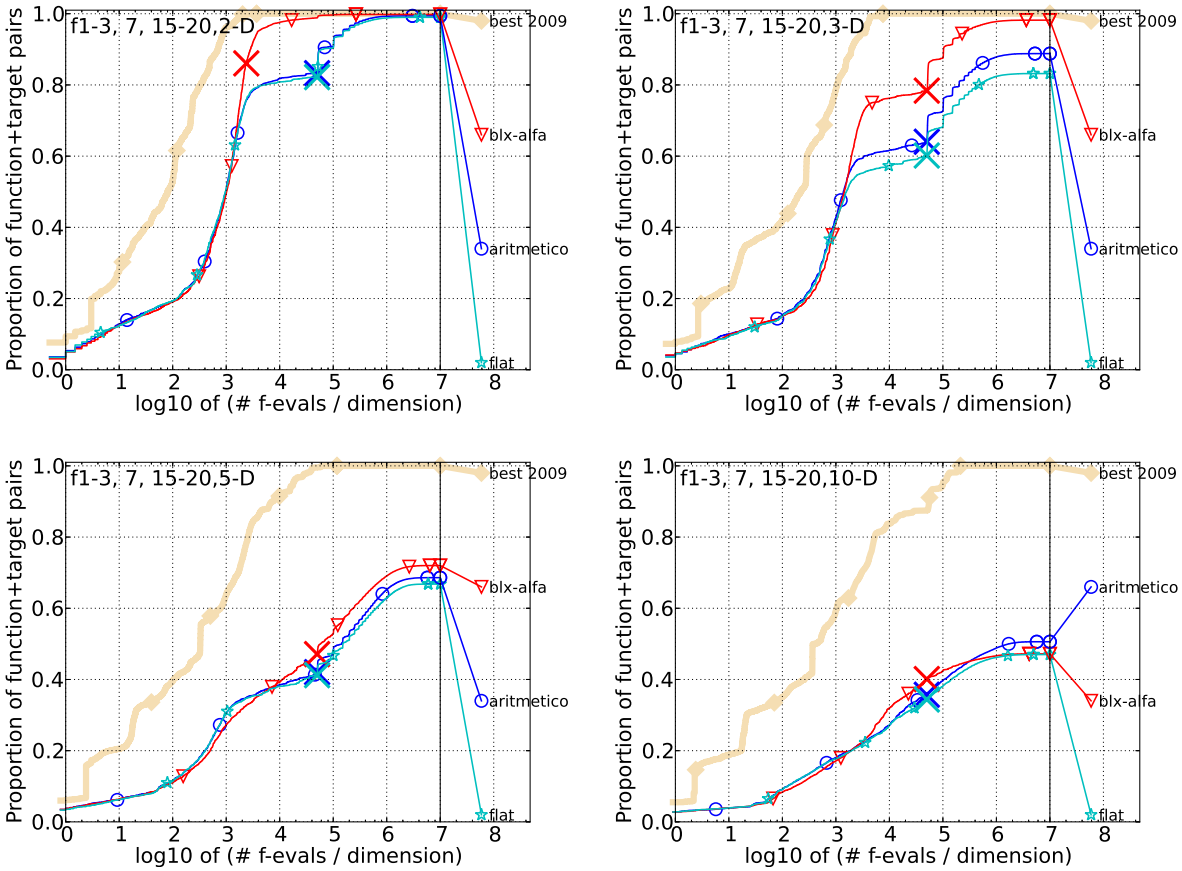
Foram testados os métodos de seleção, da *Roleta* 3.5.1.1 com o valor de avaliação bruto, e a roleta com a os valores de avaliação recalculados pelos métodos *ranking* 3.5.1.3 e o *escalonamento sigma* 3.5.1.2 e o *método do torneio* 3.5.1.4 com 4 indivíduos selecionados . Para esses testes foram utilizadas a população fixa de 250, cruzamento BLX-alfa, e mutação aleatória com taxa de 5%.

Conforme os resultados da figura 19 o desempenho do método do torneio foi significativamente superior aos demais em todos as dimensões testadas. Os métodos sigma e ranking obtiveram um desempenho similar, pelo fato de os dois utilizarem o mesmo método do roleta, mas por tratar os valores da avaliação de modo a evitar que super indivíduos domine a população. Com exceção do teste em 10 dimensões o método do sigma obteve um resultado mais expressivo em relação ao método do ranking. O método da roleta com os valores brutos obtiveram um valor bem inferior aos demais, o que já era esperado devido ao problema de utilizar os valores brutos da avaliação na roleta.

6.3 Resultado Mutação

Foram testados os métodos de mutação *uniforme* 4.4.1 e *não uniforme* 4.4.2. Para esses testes foram utilizadas a população fixa de 250, cruzamento BLX-alfa e seleção por

Figura 18 – Comparação dos desempenho global do AGC para diferentes tipos de cruzamento.

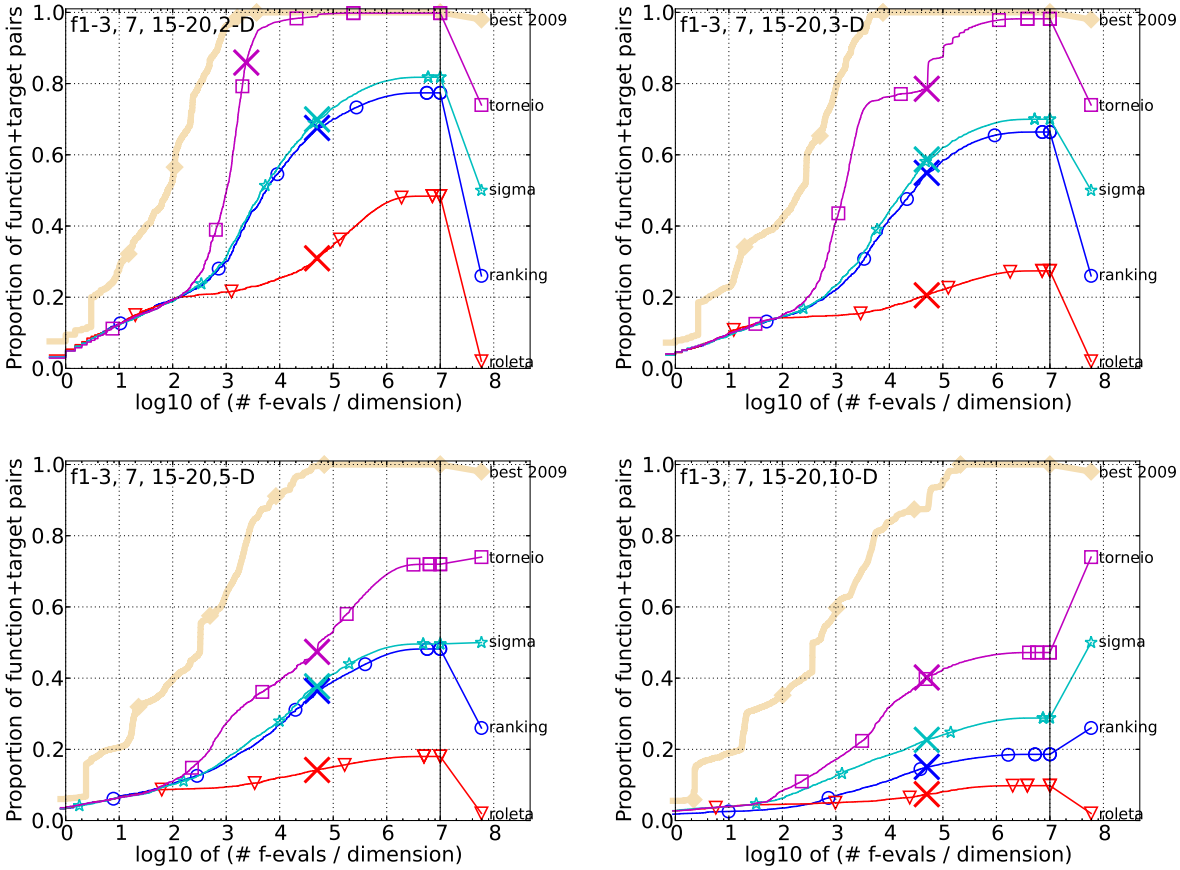


Resultados dos testes para os diferentes crossovers. O gráfico corresponde a proporção de acertos pelo número de avaliações / dimensões em $log10$ para as funções 1-3, 7, e 15-20 do COCO (COCO, 2013). Da direita para esquerda, de cima pra baixo, resultados para 2, 3, 5 e 10 dimensões respectivamente. O ponto marcado com X foram o número máximo de avaliações realizadas.

torneio e taxa de mutação a 1% e 5%.

Conforme os resultados da figura 20 para 2 dimensões a diferença de desempenho entre os operadores de mutação e sua taxa são praticamente inexistente, considerando toda a aleatoriedade dos AGs. Para 3 dimensões o desempenho dos 3 operadores é praticamente o mesmo até aproximadamente $5 * 10^3$ avaliações ou na 20 geração a *mutação uniforme em 5%* tem um desempenho melhor, pois por ter uma maior probabilidade de ocorrência e uma maior pertubação no valor do gene, tirando mais indivíduos de ótimos locais. Para 5 dimensões a mutação uniforme obteve melhor desempenho, com as duas taxas testadas 1% e 5% em relação a não uniforme, o que indica que nem sempre é uma boa estratégia reduzir o impacto que uma mutação pode ter no valor do gene. Para 10 dimensões já temos um desempenho melhor para uma taxa maior de mutação, sendo que a mesma para esse caso está influenciando mais no desempenho global do AG do que o tipo da mutação.

Figura 19 – Comparação dos resultados para diferentes tipos de seleção.

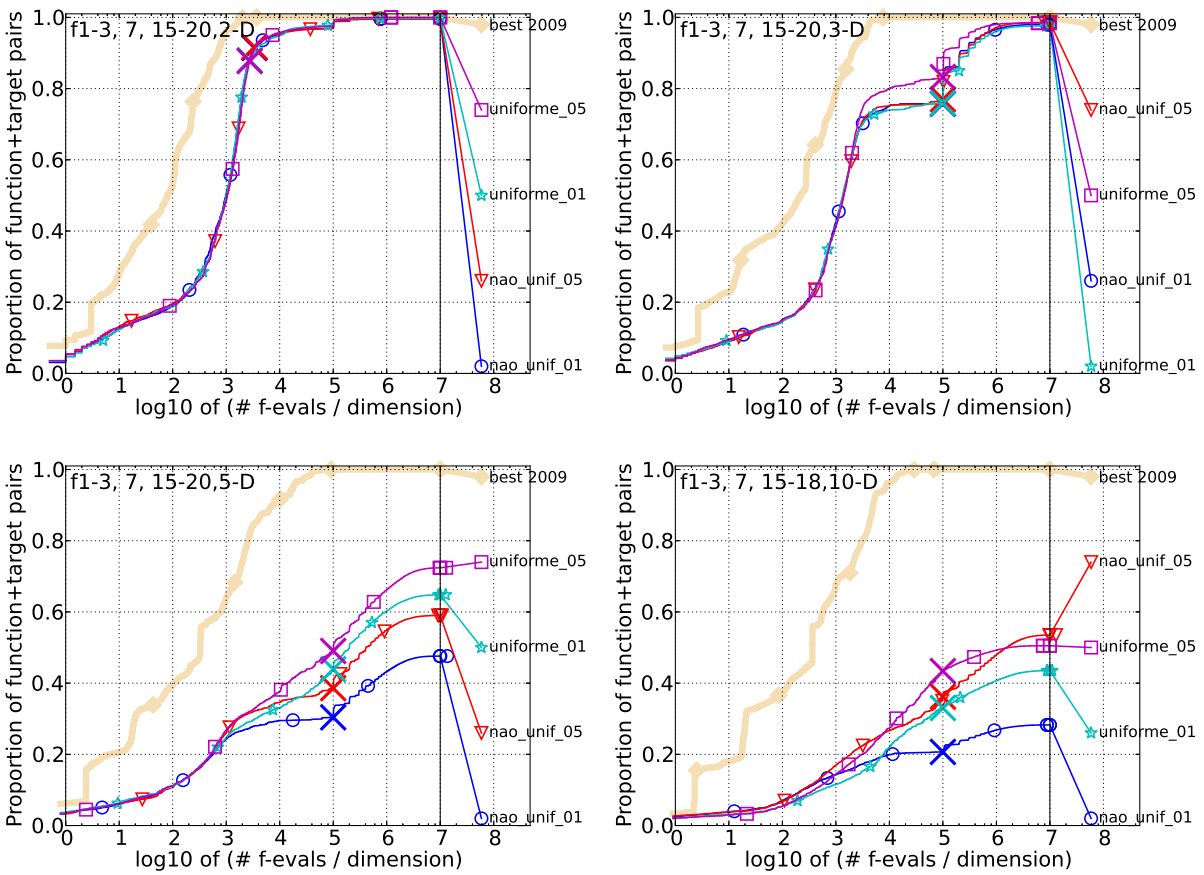


Resultados dos testes para as diferentes seleções. O gráfico corresponde a proporção de acertos pelo número de avaliações / dimensões em $log10$ para as funções 1-3, 7, e 15-20 do COCO (COCO, 2013). Da direita para esquerda, de cima pra baixo, resultados para 2, 3, 5 e 10 dimensões respectivamente. O ponto marcado com X foram o número máximo de avaliações realizadas.

Para os teste em 5 e 10 dimensões, a diferença no ponte máximo de avaliações realizadas(ponto X no gráfico) a diferença entre o melhor e o pior desempenho foi de aproximadamente 20% sendo que para 2 dimensões a diferença foi praticamente zero, o que indica que quanto maior o número de variáveis(dimensão da função), maior é o impacto que a mutação tem sobre o desempenho do AG.

De um modo geral a mutação uniforme teve um desempenho superior em relação ao não-uniforme, o que foi contrario do esperado, pois a mutação não-uniforme tem a estratégia de realizar uma mutação mais agressiva no inicio o que é desejado, pois realiza uma melhor exploração global, e nas gerações finais realizar uma mutação mais suave, de modo a explorar uma região de vizinhança do indivíduo, ao invés da mutação uniforme, que pode deslocar para uma região de busca muito diferente mesmo já estando próximo do ótimo global mesmo nas gerações finais.

Figura 20 – Comparação dos resultados para diferentes tipos de mutação.



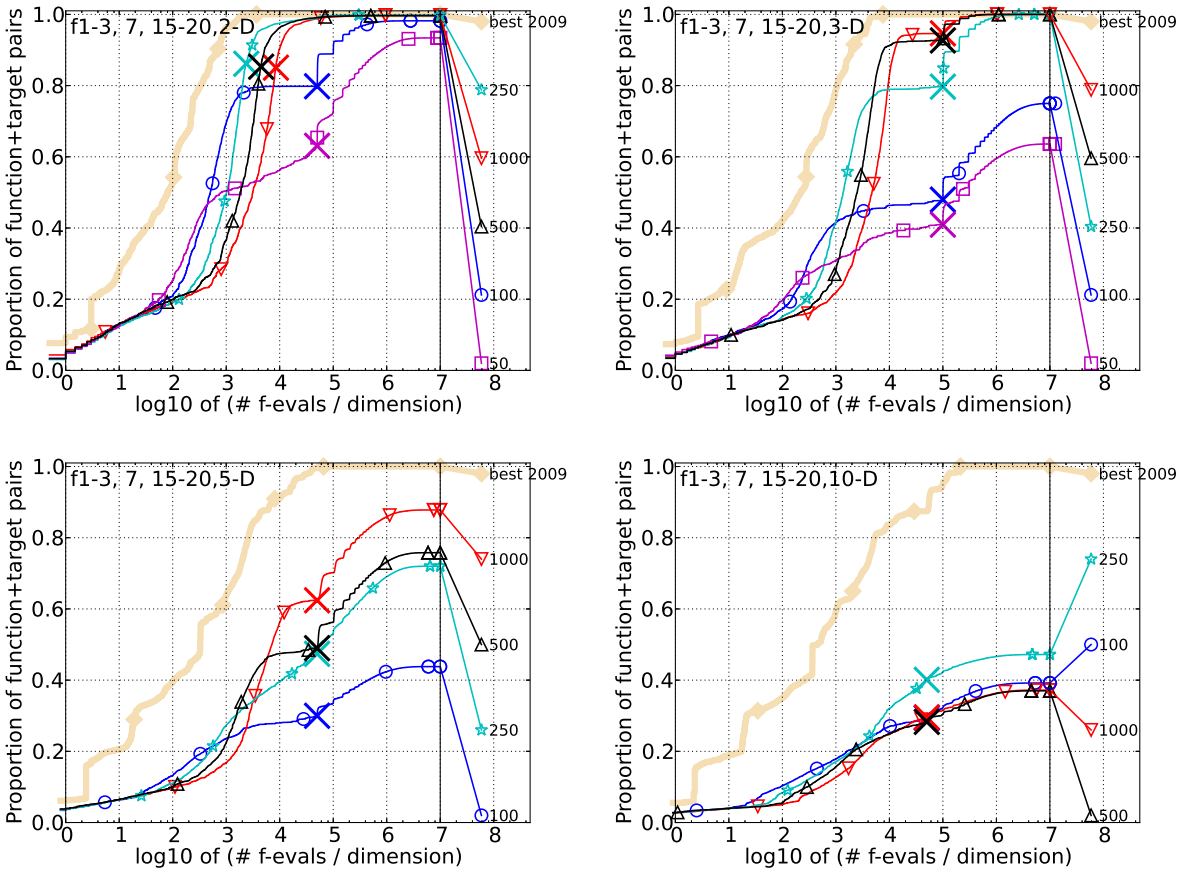
Resultados dos testes para as diferentes mutações. O gráfico corresponde a proporção de acertos pelo número de avaliações / dimensões em $log10$ para as funções 1-3, 7, e 15-20 do COCO (COCO, 2013). Da direita para esquerda, de cima pra baixo, resultados para 2, 3, 5 e 10 dimensões respectivamente. O ponto marcado com X foram o número máximo de avaliações realizadas.

6.4 Resultado para Diferentes Tamanho de população

Foram testados diferentes tamanho de população, O AG com 50, 100, 250, 500 e 1000 indivíduos. Para esses testes foram utilizadas o cruzamento BLX-alfa e seleção por torneio e a mutação não uniforme com a taxa 5%.

Em um AG, quando maior o número de indivíduos numa população, maior é a área do espaço de busca que os indivíduos vão abranger, logo tem uma maior chance de encontrar o ótimo global. Entretanto como em cada geração, deve-se avaliar todos os novos indivíduos da próxima geração, então em cada geração deve-se efetuar o o mesmo número de avaliações do tamanho da população, Então para um mesmo número de avaliações, temos que encontrar um meio termo par que tenhamos um bom desempenho entre, uma população grande e um pequeno número de gerações, ou uma população pequena e um grande número de gerações. A figura 21 mostra o desempenho do AGC com diferentes

Figura 21 – Comparação dos resultados para diferentes tamanho de população.



Resultados dos testes para as diferentes populações. O gráfico corresponde a proporção de acertos pelo número de avaliações / dimensões em $log10$ para as funções 1-3, 7, e 15-20 do COCO (COCO, 2013). Da direita para esquerda, de cima pra baixo, resultados para 2, 5 e 10 dimensões respectivamente. O ponto marcado com X foram o número máximo de avaliações realizadas.

números de população.

7 CONSIDERAÇÕES FINAIS

Neste trabalho foi demonstrado que podemos utilizar otimizadores genéricos como o algoritmos genéticos de forma eficiente para resolver problemas de otimização continua com múltiplas variáveis.

Os resultados obtidos foram satisfatórios para as funções testadas, porém com o curto tempo empregado na pesquisa, impossibilitou a realização de testes mais amplos. Portanto mesmo com um número menor de variáveis foi possível demonstrar a empregabilidade dessas técnicas de computação evolutiva para solucionar problemas complexos.

É importante a realização da comparação entre o desempenho com vários parâmetros diferentes de entrada, pois mesmo que não seja possível definir um conjunto de parâmetros que obtenha o melhor desempenho para qualquer problema tratado, podemos obter conjuntos que sejam promissores para um grupo de funções.

As funções utilizadas neste trabalho possuem o ótimo conhecido, já que utilizamos o COCO (COCO, 2013), cuja função é testar o desempenho dos algoritmos. Entretanto em problemas práticos não conhecemos o ponto ótimo. Caso utilizássemos funções das quais não conhecemos o seu ponto ótimo, podemos definir algum conjunto de parâmetros que tenha um melhor desempenho, mas não teríamos o quão próximo chegou do ótimo. Por isto é importante testarmos o AG com funções nas quais conhecemos o ponto ótimo, para que possamos determinar que parâmetros do AG podemos utilizar quando tivermos que utilizá-lo em situações reais.

Como trabalhos futuros pode ser implementado um mecanismo de população variáveis, onde o AG inicializa com uma população grande para cobrir uma área maior do domínio do problema, e conforme os indivíduos forem convergindo, pode-se reduzir a sua população. Também pode ser implementado formas de inicializar a população de forma que os indivíduos fiquem distribuídos de forma mais homogênea na área de busca. Ainda poderá a implementação para a paralelização do processamento do AG no processo de seleção, reprodução, mutação e avaliação, já que os mesmos em cada etapa não depende do resultado dos outros indivíduos. Desta forma a execução poder utilizar mais de um núcleo do processador ou até mesmos outras máquinas disponíveis pela rede, reduzindo o tempo de cada execução.

REFERÊNCIAS

AUGER, S. N. R. A. *Real-Parameter Black-Box Optimization Benchmarking 2010: Presentation of the Noiseless Functions*. 2013. <http://coco.lri.fr/downloads/download13.09/bbobdocfunctions.pdf>. Citado na página 42.

BORTOLOSSI, H. *Cálculo Diferencial a Várias Variáveis: Uma Introdução à Teoria da Otimização*. 2 edição. ed. [S.l.]: Editora Loyola, 2002. Citado na página 9.

CARVALHO, E. A. Sistemas inteligentes: aplicações a recursos hídricos e ciências ambientais. In: _____. Porto Alegre: Universidade/UFRGS : Associação Brasileira de Recursos Hídricos., 1999. cap. Capitulo 3 :Introdução aos algoritmos genéticos., p. 99–150. Citado 6 vezes nas páginas 13, 30, 31, 32, 33 e 35.

COCO. *COmparing Continuous Optimisers: COCO*. 2013. <http://coco.gforge.inria.fr>. Acessado em 1 de maio de 2014. Citado 6 vezes nas páginas 39, 44, 45, 46, 47 e 48.

GMP. *The GNU Multiple Precision Arithmetic Library*. 2013. <http://gmplib.org/>. Acessado em 25 de novembro de 2013. Citado na página 32.

HAUPT, R. S. *Practical Genetic Algorithms*. Second edition. [S.l.]: A JOHN WILEY and SONS, INC., PUBLICATION, 2004. Citado 2 vezes nas páginas 31 e 32.

LACERDA, E. *Model Selection of RBF Networks Via Genetic Algorithms*. Tese (Doutorado) — Universidade Federal de Pernambuco, 2003. Citado 4 vezes nas páginas 34, 35, 36 e 37.

LINDEN, R. *Algoritmos Genéticos*. 3 edição. ed. [S.l.]: Editora Ciência Moderna, 2012. Citado 16 vezes nas páginas 9, 12, 16, 17, 20, 21, 23, 24, 25, 26, 27, 30, 33, 34, 36 e 37.

MELANIE, M. *An Introduction to Genetic Algorithms*. [S.l.]: A Bradford Book The MIT Press, 1996. Citado 3 vezes nas páginas 23, 24 e 25.

NORVIG, S. P. *Inteligência Artificial*. 2 edição. ed. [S.l.]: Editora Ciência Moderna, 2012. Citado na página 9.

PENTTINEN, H. K. A. On initial populations of a genetic algorithm for continuous optimization problems. *J Glob Optim 37, p.405–436*, 2007. Citado na página 29.

PRADO, S. J. *Otimização por Colônia de Partículas*. 2005. <http://www.sbmac.org.br/eventos/cnmac/cd_xxviii_cnmac/posters/060posterCNMAC2005_jair_prado.pdf>. Acessado em 1 de dezembro de 2013. Citado na página 12.

ROS, N. A. S. R. *Real-Parameter Black-Box Optimization Benchmarking: Experimental Setup*. 2013. <http://coco.lri.fr/downloads/download13.09/bbobdocexperiment.pdf>. Citado na página 40.

RUBICITE, R. *Genetic Algorithms"*. 2014. <http://www.rubicite.com/Tutorials/GeneticAlgorithms.aspx>. Acessado em 30 de novembro de 2013. Citado na página 24.

SARAMAGO, S. *Métodos de Otimização Randômica: algoritmos genéticos e "simulated annealing"*. 2012. <http://www.sbmac.org.br/arquivos/notas/livro_06.pdf>. Acessado em 25 de novembro de 2013. Citado na página 12.

SHAMSUDDIN, Z. S. A review of population-based meta-heuristic algorithms. *Int. J. Advance. Soft Comput. Appl., Vol. 5, No. 1, March 2013*, 2013. Citado 3 vezes nas páginas 13, 14 e 15.

SIARRY, R. P. A continuous genetic algorithm designed fot the global optimization of multimodal functions. *Journal of Heuristics, 6:191-213(2000)*, 2000. Citado 3 vezes nas páginas 27, 28 e 36.

STEWART, J. *Cálculo*. 5. ed. [S.l.]: Cengage Learning, 2008. Citado na página 13.

TANOMARU, J. Motivação, fundamentos e aplicações de algoritmos genéticos. *II Congresso Brasileiro de Redes Neurais*, 1995. Citado na página 27.

Anexos

ANEXO A – CÓDIGO FONTE

```
function xbest = GA(
  num_genes,
  tam_populacao,
  elitismo,
  taxa_mutacao,
  tipo_selecao,
  tipo_crossover,
  tipo_mutacao,
  selecao_roleta_min = 1,  # Valor minimo para a roleta em sele   o.
  selecao_roleta_max = 2,  # Valor m ximo para a roleta em sele   o.
  selecao_num_torneio = 2, # Numero de individuos para a torneio.
  mutacao_grep_DP = 1 #Desvio padr o utilizado na muta   o grep.
  )

  if rem(elitismo, 2) != 0
    error('o valor de elitismo deve ser multiplo de 2.');
  endif

  xMax = 5;
  xMin = -5;
  xpop = rand(num_genes,tam_populacao) .* (xMax .- xMin) .+ xMin;
  pop_avaliada = avalia_pop(xpop);
  continua = true;
  CON_alvo = fgeneric('ftarget');

  maximoAvaliacoes = 50000 * num_genes;
  max_geracoes = maximoAvaliacoes ./ tam_populacao;

  iteracao = 1;
  while continua
    # Inicializa variaveis de sele   o
    selecao('inicializa',pop_avaliada,selecao_roleta_min,selecao_roleta_max
    ,selecao_num_torneio);

    nova_pop = zeros(num_genes,1);
    for j=1:2:(tam_populacao - elitismo)
      pai1 = selecao(tipo_selecao);
      pai2 = selecao(tipo_selecao);
      filhos = crossover(tipo_crossover, [pai1 , pai2], .5);
      nova_pop = [nova_pop , filhos];
    endfor
    nova_pop(:,1) = [];
```

```
42      nova_pop = mutacao(tipo_mutacao, nova_pop, taxa_mutacao, iteracao++,
        max_geracoes,mutacao_grep_DP);
43
44      if elitismo > 0
45        [fvalues, idx] = sort(pop_avaliada(1,:));
46
47        melhores = zeros(num_genes,1);
48        for i=1:elitismo
49          melhores = [melhores , xpop(:,idx(i))];
50        end
51        melhores(:,1) = [];
52        nova_pop = [nova_pop , melhores];
53      endif
54      xpop = nova_pop;
55      aux = fgeneric(xpop); #avalia popula  o
56      pop_avaliada = [aux; xpop];
57
58      melhor_avaliacao = min(pop_avaliada(1,:));
59      continua = fgeneric('evaluations') < maximoAvaliacoes ...
60                  & (melhor_avaliacao .- CON_alvo) > 1e-8;
61
62    endwhile
63
64    [fvalues, idx] = sort(pop_avaliada(1,:));
65
66    fbest = fvalues(1);
67    xbest = xpop(:,idx(1));
68    iteracao
69  endfunction;
```

Code A.1 – GA

```
1  #Fun  o filhos = crossover(tipo, pais, alfa)
2  # Descri  o: Gera dos filhos a partir do genes de dois pais.
3  # Parametros:
4  #   tipo: O tipo de crossover a ser efetuado.
5  #     'simples': Define um ponto de corte, toma os valores
6  #                do pai a esquerda e do outro pai a direita.
7  #      'flat': Definir um intervalo fechado entre os cromossomos pais
8  #              e escolher aleatoriamente um valor dentro deste intevalo.
9  #      'aritmetico': Genes gerado s o uma m dia ponderada entre genes dos
       pais.
10 #      'BLX-Alfa': Gera um valor que pode ser estendido fora do intervalo
       entre
11 #                  entre o cromossomo, o parametro alfa extende o intervalo
       do
12 #                  seu valor para ambos os lados.
```

```
13 #   alfa: Define o quanto do intervalo poder  ser excedido na opera  o
       do
14 #          BLX-Alfa
15 # Retorno: Dois individuos filhos gerado a partir da combina  o dos genes
16 #          dos seu pais.
17 #
18 #
19 function filhos = crossover(
20                tipo,
21                pais,
22                alfa = .5
23                )
24
25   num_genes = rows(pais);
26   filhos = zeros(num_genes,2);
27 # ----------------------------------------------------
28   if strcmpi(tipo, 'simples')
29
30     ponto_corte = floor(rand * num_genes + 1);
31
32     if(ponto_corte > num_genes)
33       ponto_corte = num_genes;
34
35     endif
36     for i=1:num_genes
37       if i < ponto_corte
38         filhos(i,1) = pais(i,1);
39         filhos(i,2) = pais(i,2);
40       else
41         filhos(i,1) = pais(i,2);
42         filhos(i,2) = pais(i,1);
43       end
44     endfor
45 # ----------------------------------------------------
46   elseif strcmpi(tipo, 'flat')
47
48     for i=1:num_genes
49       gen_menor = min(pais(i,1),pais(i,2));
50       gen_maior = max(pais(i,1),pais(i,2));
51       filhos(i,1) = gen_menor .+ rand .* (gen_maior .- gen_menor);
52       filhos(i,2) = gen_menor .+ rand .* (gen_maior .- gen_menor);
53     endfor
54 # ----------------------------------------------------
55   elseif strcmpi(tipo, 'aritmetico')
56     for i=1:num_genes
57       valor = rand;
58       filhos(i,1) =  valor .* pais(i,1) .+ (1 .- valor) .* pais(i,2);
```

```
59        filhos(i,2) =  valor .* pais(i,2) .+ (1 .- valor) .* pais(i,1);
60      endfor
61  # ------------------------------------------------------
62    elseif strcmpi(tipo, 'BLX-Alfa')
63      minimo = -alfa;
64      maximo = 1 .+ alfa;
65      betaa = minimo .+ rand .* (maximo - minimo);
66        filhos(:,1) = pais(:,1) .+ betaa .* (pais(:,2) .- pais(:,1));
67        filhos(:,2) = pais(:,2) .+ betaa .* (pais(:,1) .- pais(:,2));
68  # ------------------------------------------------------
69    else
70      error('Tipo de crossover n o encontrado.');
71    end
72  # ------------------------------------------------------
73  endfunction
```

Code A.2 – crossover.m

```
1  #Fun  o muta  o(tipo, populacao, prob_ocorrencia, geracao_atual,
       geracao_maxima, b, desvio_padrao)
2  #Descri  o: Altera o valor de algum gene, conforme probabilidde de
       muta  o.
3  #Par metros:
4  # tipo: Define o tipo de muta  o que dever  ser operada.
5  #   'Aleatoria': O valor do gene  troca por um valor entre o min e o
       m ximo aceito pelo gene.
6  #       'nao_uniforme': O valor do gene  alterado por um valor pr ximo
       do atual, conforme o
7  #                       o n mero de gera oes correntes, nas primeiras
       gera oes a muta  o
8  #           mais agressiva, nas  ltimas  gera  es a muta  o  mais
       branda.
9  #   'gausiana': O valor do gene  alterado somando um valor conforme um
       desvio
10 #         desvio padr o passado por par metro.
11 #   'limite': O valor do gene  alterado para valor m ximo ou minimo
       suportado pelo gene.
12 # populacao: Popula ao que dever  receber a muta ao.
13 # geracao_atual: O n mero da gera  o atual.
14 # geracao_maxima: O n mero m ximo de gera  es.
15 # desvio_padrao: o valor do desvio padr o utilizado na muta  o do tipo
       gausiana.
16 #
17 # Retorno: pop_mutada: a popula ao com as muta oes efetuadas.
18
19 function pop_mutada = mutacao(
20                   tipo,
21                   populacao,
```

```
22                  prob_ocorrencia,
23                  geracao_atual,
24                  geracao_maxima,
25                  desvio_padrao
26                  )
27
28  pop_mutada = populacao;
29  tamanho_populacao = columns(populacao);
30  num_genes = rows(populacao);
31
32  for j=1 : tamanho_populacao
33    for i=1 : num_genes
34      if(rand < prob_ocorrencia)
35# ------------------------------------------------
36        if strcmpi(tipo, 'aleatoria')
37        pop_mutada(i,j) = unifrnd(-5,5); # Random Uniform Distribution
38# ------------------------------------------------
39        elseif strcmpi(tipo, 'nao_uniforme')
40          xMin = -5;
41          xMax = 5;
42          b = 6;
43          if rand < 0.5
44            y = xMax - populacao(i,j);
45          else
46            y = -(populacao(i,j) .- xMin);
47          end
48          pop_mutada(i,j) = populacao(i,j) .+ y .* rand .^ ((1-(
    geracao_atual./geracao_maxima)).^b);
49# ------------------------------------------------
50        elseif strcmpi(tipo, 'gausiana')
51          pop_mutada(i,j) = normrnd(populacao(i,j),desvio_padrao); #Random
    Normal Distribution
52# ------------------------------------------------
53        elseif strcmpi(tipo, 'limite')
54          if rand < 0.5
55            pop_mutada(i,j) = -5;
56          else
57            pop_mutada(i,j) = 5;
58          end
59# ------------------------------------------------
60        else
61          error('Tipo de muta  o n o encontrado.');
62        end
63      endif
64    endfor
65  endfor
66 endfunction
```

Code A.3 – mutacao.m

```
# Fun  o sele  o(tipo,pop_avaliada, r_min, r_max, num_torneio)
# Descri  o: Selecionado um individuo dentro da popula  o, com uma
#           probabilidade conforme crit rios definidos pela fun  o.
# Parametros:
#   tipo: o tipo de sele  o a ser efetuada.
#     'inicializa': Inicializa variaveis utiizadas nas sele  es.
#     'roleta': Sele  o do tipo roleta, os individuos s o selecionado,
#               com uma probabilidade conforme a sua avalia  o.
#     'ranking': Vers o similiar a roleta, s  que os individuos s o
#                ordenados e recebem uma avalia  o entre r_min e r_max.
#     'torneio': Um num_torneio s o selecionados, o melhor entre esses
#                selecionados  escolhido.
#     'sigma': Quando maior o desvio padr o dos individuos menor a
    press o
#              seletiva, quando menor o desvio padr o, maior  a
#              press o seletiva.
#   r_min: O valor da avalia  o minima para a roleta.
#   r_max: O valor da avalia  o m xima para a roleta.
#   num_torneio: o n mero de individuos a serem sorteados no torneio.
#
# Retorno: Uma solu  o selecionado dentro da popula  o com uma
#          probabilidade conforme a sua avalia  o.

function pai = selecao(
                       tipo,
                       pop_avaliada,
                       r_min,
                       r_max,
                       num_torneio
                       )

  persistent SEL_tamanho_populacao;
  persistent SEL_num_genes;
  persistent SEL_soma_avaliacao;
  persistent SEL_pop_avaliada;
  persistent SEL_torneio_numero;
  persistent SEL_pop_avaliada_ranking;
  persistent SEL_soma_avaliacao_ranking;
  persistent SEL_pop_avaliada_sigma;
  persistent SEL_soma_avaliacao_sigma;

# ------------------------------------------------

  if strcmpi(tipo, 'inicializa')
```

```
44      SEL_pop_avaliada = pop_avaliada;
45      SEL_tamanho_populacao = columns(SEL_pop_avaliada);
46      SEL_num_genes = rows(SEL_pop_avaliada) .-1;
47      SEL_roleta_min = r_min + eps;
48      SEL_roleta_max = r_max + eps;
49      SEL_torneio_numero = num_torneio;
50
51      # para que procure sempre o m nimo.
52      SEL_pop_avaliada(1,:) = -SEL_pop_avaliada(1,:);
53
54      minAvaliacao = min(SEL_pop_avaliada(1,:));
55      if(minAvaliacao < 0) #Todas as avalia   es devem ser positivas
56        SEL_pop_avaliada(1,:) = abs(1.1 .* minAvaliacao) .+ SEL_pop_avaliada
        (1,:);
57      endif
58
59      SEL_soma_avaliacao = sum(SEL_pop_avaliada(1,:));
60
61      [fvalues, idx] = sort(SEL_pop_avaliada');
62      idx = idx(:,1)';
63      SEL_pop_avaliada_ranking = SEL_pop_avaliada;
64
65      passo = (SEL_roleta_max .- SEL_roleta_min) ./ (SEL_tamanho_populacao .-
        1);
66      aux_avaliacao = SEL_roleta_max;
67      for i=1:SEL_tamanho_populacao
68        SEL_pop_avaliada_ranking(1,idx(SEL_tamanho_populacao .- i .+ 1)) =
        aux_avaliacao;
69        aux_avaliacao .-= passo;
70      endfor
71
72      SEL_soma_avaliacao_ranking = sum(SEL_pop_avaliada_ranking(1,:));
73
74      SEL_pop_avaliada_sigma = SEL_pop_avaliada;
75
76      media_avaliacoes = mean(SEL_pop_avaliada_sigma(1,:));
77      desvio_padrao_avaliacoes = std(SEL_pop_avaliada_sigma(1,:));
78
79      if(desvio_padrao_avaliacoes != 0)
80        SEL_pop_avaliada_sigma(1,:) = 1 + ((SEL_pop_avaliada(1,:) .-
        media_avaliacoes) ...
81                                          ./ (2 .* desvio_padrao_avaliacoes));
82      else
83        SEL_pop_avaliada_sigma(1,:)= 1;
84      end
85
86      SEL_soma_avaliacao_sigma = sum(SEL_pop_avaliada_sigma(1,:));
```

```
87 # ------------------------------------------------------
88   elseif strcmpi(tipo, 'roleta')
89
90     aux = SEL_pop_avaliada(1,1);
91     j = 1;
92     sorteio = 0 .* SEL_soma_avaliacao;
93     while aux < sorteio & j < SEL_tamanho_populacao
94       aux += SEL_pop_avaliada(1,++j);
95     endwhile
96     pai = SEL_pop_avaliada(2:SEL_num_genes+1,j);
97 # ------------------------------------------------------
98   elseif strcmpi(tipo, 'ranking')
99     aux = SEL_pop_avaliada_ranking(1,1);
100    j = 1;
101    sorteio = rand .* SEL_soma_avaliacao_ranking;
102    while aux < sorteio & j < SEL_tamanho_populacao
103      aux += SEL_pop_avaliada_ranking(1,++j);
104    endwhile
105    pai = SEL_pop_avaliada_ranking(2:SEL_num_genes+1,j);
106 # ------------------------------------------------------
107 elseif strcmpi(tipo, 'sigma')
108   aux = SEL_pop_avaliada_sigma(1,1);
109   j = 1;
110   sorteio = rand .* SEL_soma_avaliacao_sigma;
111   while aux < sorteio & j < SEL_tamanho_populacao
112     aux += SEL_pop_avaliada_sigma(1,++j);
113   endwhile
114   pai = SEL_pop_avaliada_sigma(2:SEL_num_genes+1,j);
115 # ------------------------------------------------------
116   elseif strcmpi(tipo, 'torneio')
117     competidores = zeros(SEL_num_genes+1,1);
118     for i=1:SEL_torneio_numero
119       j = floor(rand*SEL_tamanho_populacao + 1);
120       if j > SEL_tamanho_populacao # se rand = 1
121         j = SEL_tamanho_populacao;
122       endif
123       competidores = [competidores, SEL_pop_avaliada(:,j)];
124     endfor
125     competidores(:,1) = [];
126     [fvalues, idx] = sort(competidores');
127     idx = idx(:,1)';
128     pai = competidores(2:SEL_num_genes+1,idx(SEL_torneio_numero));
129 # ------------------------------------------------------
130   else
131     error('Tipo de sele   o n o encontrado.');
132   end
133 # ------------------------------------------------------
```

```
134 endfunction
```

Code A.4 – selecao.m

```
1  function xbest = experimento(
2    diretorio_testes,
3    tam_populacao,
4    elitismo,
5    taxa_mutacao,
6    tipo_selecao,
7    tipo_crossover,
8    tipo_mutacao,
9    dimensions,
10   selecao_roleta_min = 1,  # Valor minimo para a roleta em sele   o.
11   selecao_roleta_max = 2,  # Valor m ximo para a roleta em sele   o.
12   selecao_num_torneio = 4, # Numero de individuos para a torneio.
13   mutacao_grep_DP = 1 #Desvio padr o utilizado na muta   o grep.
14   )
15
16 addpath('results');  % should point to fgeneric.m etc.
17 datapath = diretorio_testes;  % different folder for each experiment
18 % opt.inputFormat = 'row';
19 opt.algName = 'Genetic Algorithm';
20
21 display('------------------------------------------------------');
22 display('----------INICIO-NOVO-EXPERIMENTO----------');
23 display('------------------------------------------------------');
24
25 opt.comments = strcat('tamPop = ',num2str(tam_populacao), ...
26                       ' elitismo = ', num2str(elitismo), ...
27                       ' taxa_mutacao = ', num2str(taxa_mutacao), ...
28                       ' tipo_selecao = ', tipo_selecao, ...
29                       ' tipo_crossover = ', tipo_crossover, ...
30                       ' tipo_crossover = ', tipo_crossover, ...
31                       ' tipo_mutacao = ', tipo_mutacao, ...
32                       ' selecao_roleta_min = ', num2str(selecao_roleta_min)
    , ...
33                       ' selecao_roleta_max = ', num2str(selecao_roleta_max)
    , ...
34                       ' selecao_num_torneio = ', num2str(
    selecao_num_torneio)
35                     )
36
37
38 maxfunevals = '10 * dim'; % 10*dim is a short test-experiment taking a few
    minutes
39                           % INCREMENT maxfunevals successively to larger
    value(s)
```

```
40 minfunevals = 'dim + 2';  % PUT MINIMAL SENSIBLE NUMBER OF EVALUATIONS for
       a restart
41 maxrestarts = 5;%e4;         % SET to zero for an entirely deterministic
       algorithm
42
43 %dimensions = [2, 3, 5, 10, 20, 40];  % small dimensions first, for CPU
       reasons
44 %dimensions = [2, 5, 10];  % small dimensions first, for CPU reasons
45 %functions = benchmarks('FunctionIndices');  % or benchmarksnoisy(...)
46 functions = [1,2,3,7,15,16,17,18,19,20];  % or benchmarksnoisy(...)
47 instances = [1:5, 31:40];  % 15 function instances
48 %instances = [1:3];  % 15 function instances
49
50 more off;  % in octave pagination is on by default
51
52 t0 = clock;
53 rand('state', sum(100 * t0));
54 for dim = dimensions
55   for ifun = functions
56     for iinstance = instances
57       fgeneric('initialize', ifun, iinstance, datapath, opt);
58
59       % independent restarts until maxfunevals or ftarget is reached
60       for restarts = 0:maxrestarts
61         if restarts > 0  % write additional restarted info
62           fgeneric('restart', 'independent restart')
63         end
64
65         GA(
66           dim,                  # numero de genes.
67           tam_populacao,        # tamanho_pop.
68           elitismo,             # elitismo.
69           taxa_mutacao,         # probabilidade de muta  o.
70           tipo_selecao,         # tipo_selecao. 'roleta' 'ranking' 'torneio'
       'sigma'
71           tipo_crossover,       # tipo_crossover. 'simples' 'flat' '
       aritmetico' 'BLX-Alfa'
72           tipo_mutacao,         # tipo_mutacao.  'aleatoria' 'nao_uniforme' '
       gausiana' 'limite'
73           selecao_roleta_min, # selecao_roleta_min
74           selecao_roleta_max, # selecao_roleta_max
75           selecao_num_torneio,# selecao_num_torneio
76           1                     # mutacao_grep_DP
77           );
78
79         if fgeneric('fbest') < fgeneric('ftarget') || ...
80            fgeneric('evaluations') + eval(minfunevals) > eval(maxfunevals)
```

```
81            break;
82          end
83        end
84
85        disp(sprintf(['  f%d in %d-D, instance %d: FEs=%d with %d restarts,'
    ...
86                      ' fbest-ftarget=%.4e, elapsed time [h]: %.2f'], ...
87                     ifun, dim, iinstance, ...
88                     fgeneric('evaluations'), ...
89                     restarts, ...
90                     fgeneric('fbest') - fgeneric('ftarget'), ...
91                     etime(clock, t0)/60/60));
92
93        fgeneric('finalize');
94      end
95      disp(['      date and time: ' num2str(clock, ' %.0f')]);
96    end
97    disp(sprintf('------- dimension %d-D done -------', dim));
98  end
99
100 endfunction
```

Code A.5 – experimento.m

ANEXO B – ARTIGO

Otimização de Funções Reais Multidimensionais Utilizando Algoritmo Genético Contínuo

Gustavo Pinho Kretzer de Souza[1],

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC - Brasil

gustavokretzer@gmail.com

Abstract. *Increasingly man seeking the best possible use of resources existents. Optimization is a large area of interest as in a large range of mathematical applications is common to want to find great function points. The genetic algorithm is a stochastic search approach based on the concepts of reproduction and natural evolution of Charles Darwin. This work aims the implementation of a continuous genetic algorithm and show the viability of using genetic algorithm to the optimization of real functions with multiple variables. Initially the algorithm generates an initial population. Individuals in the population applies genetic operators of selection, crossover and mutation adapted to real variables, in order to increase the effectiveness of the algorithm to converge to acceptable solutions. Subsequently the performance of the algorithm is evaluated for different alternatives and algorithm parameters.*

Resumo. Cada vez mais o homem busca o melhor aproveitamento possível dos recursos existentes. Otimização é uma grande área de interesse pois em uma grande gama de aplicações matemáticas é comum desejar encontrar pontos ótimos de funções. O algoritmo genético é uma abordagem estocástica de busca baseado nos conceitos de reprodução e evolução natural de Charles Darwin. Este trabalho objetiva a implementação de um algoritmo genético contínuo e mostrar a sua viabilidade de utilizar o algoritmo genético para a otimização de funções reais com múltiplas variáveis. Inicialmente o algoritmo gera uma população inicial. Nos indivíduos da população aplica-se os operadores genéticos de seleção, cruzamento e mutação adaptados para variáveis reais, de forma a aumentar a eficácia do algoritmo em convergir para soluções aceitáveis. Posteriormente o desempenho do algoritmo é avaliado para diferentes alternativas e parâmetros do algoritmo.

1. Introdução

Cada vez mais o homem busca o melhor aproveitamento possível dos recursos existentes. A otimização consiste na busca do melhor entre todos os valores possível para dadas variáveis, para uma função de um determinado objetivo e com as limitações existentes. Em uma grande gama de aplicações matemáticas, é comum se desejar encontrar o valor máximo ou mínimo de funções, ou seja, encontrar um valor para todas as variáveis da mesma, na qual o valor da função seja o ótimo.

Para resolver esses problemas podemos utilizar os algoritmos genéticos são métodos de busca baseado no processo biológico de evolução e seleção natural de Charles Darwin (Linden, 2012), onde as características favoráveis tornam-se mais

comuns nas gerações sucessivas e as características desfavoráveis tendem a desaparecer. No algoritmo genético é mantida uma população de possíveis soluções do problema, sendo essas avaliadas por uma função de aptidão. Ao longo de várias gerações são geradas novas populações a partir da população anterior, sendo esses novos indivíduos fruto do cruzamento e mutações das soluções anteriores, este processo iterativo vai-se evoluindo as soluções, convergindo com uma certa probabilidade para o máximo global (Norvig, 2012).

É neste contexto, que este trabalho propõe uma investigação dos algoritmos genéticos contínuos, voltados para a solução de problemas combinatoriais cujo espaço de busca são os números reais, uma implementação e testes com diferentes parâmetros no algoritmo para um medida de desempenho do mesmo para diferentes funções.

2. Objetivos

O trabalho a ser realizado, diante do exposto, apresenta os seguintes objetivos:

2.1. Objetivo Geral

Desenvolver um algoritmo genético contínuo e realizar testes para avaliar o desempenho do mesmo com diferentes funções e parâmetros.

2.2. Objetivos Especificos

- Analisar os conceitos teóricos de algorítimos de otimização de meta-heurísticas.

- Implementar o algoritmo genético em uma linguagem de programação.

- Analisar as estratégias efetuadas pelo algoritmo ao longo da execução.

- Comparar o desempenho para as estratégias e funções utilizadas.

3. Problemas de otimização

Problemas de otimização contínuos, são aqueles em que as variáveis são continuas, ou seja, onde se desejar minimizar ou maximizar uma determinada função numérica normalmente com várias variáveis, em algum contexto que possa existir restrições. O máximo global é definido como

> Uma função f tem máximo absoluto (ou máximo global) em c se $f(c) \geq f(x)$ para todo $x \in D$ \$, onde D é o domínio de f. Analogamente, f tem um mínimo absoluto em c se $f(c) \leq f(x)$ para todo $x \in D$\$, e o número f(c) é denominado valor mínimo de $f \in D$\$. (STEWART, 2008)

Ou seja, o objetivo é encontrar esse valor c. A figura 1mostra os máximos locais e o máximo global de uma função.

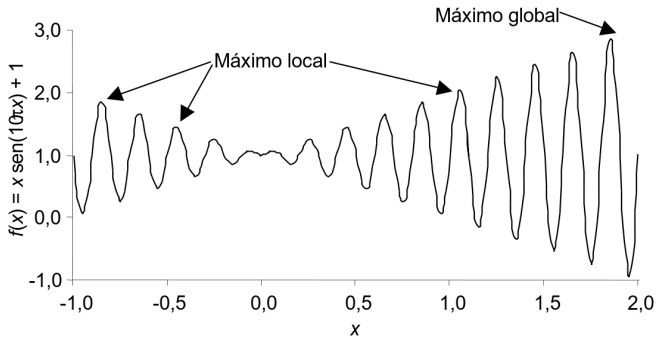


Figura 1: Gráfico da função f (x) = x(sen(10πx)) + 1, Fonte: (CARVALHO, 1999)

4. Algoritmos de meta-heurísticas de otimização

A palavra heurísticas que tem origem no grego heuriskein que significa descobrir, são procedimentos fundamentado em experiência para resolver problemas, que indica uma solução que não tem garantia que a mesma seja ótima ou seja é um conhecimento momentâneo, que não são verificáveis, são aplicados onde a busca exaustiva é impraticável. São métodos que devem ter baixo custo computacional e que encontram uma solução satisfatória (não exata) do problema.

Meta-heurísticas são métodos heurísticos para resolver problemas de otimização de forma genérica, geralmente são utilizados quando não se conhece algum algoritmo eficiente para resolver tais problemas. Esses algoritmos utilizam a aleatoriedade e conhecimento histórico dos resultados anteriores para se guiarem pelo espaço de busca. De acordo com a definição meta-heurística é

> Um processo de geração iterativo que guia uma heurística subordinada por combinação de forma inteligente de diferentes conceitos para explorar e explotar o espaço de busca, aprendendo estratégias que são usadas para estrutura da informação na ordem de encontrar eficientemente uma solução próxima da ótima. (SHAMSUDDIN, 2013)

5. Algoritmos genéticos

Algoritmo genético é um método que simula o processo de evolução natural (biológica), destinando-se a solucionar problemas de otimização. O AG pode ser entendido como uma interpretação matemática e algorítmica das teorias de Darwin, no qual as hipóteses podem ser resumida: A evolução é maneira que age sobre os cromossomos do organismo, e não sobre o organismo que os transporta. Logo o que ocorrer durante a vida do organismo não altera os seus cromossomos, no entanto, os cromossomos refletem diretamente nas características do organismo. A seleção natural faz com que os cromossomos que criam organismos mais bem adaptado ao ambiente, sobrevivam e reproduzam mais que outros organismos com cromossomos menos adaptados. A reprodução é o ponto no qual a evolução se define. O processo de recombinação dos cromossomos (entre dois pais) podem gerar cromossomos dos filhos bem diferente dos pais.

Algoritmos genéticos é uma técnica heurística de otimização global utilizada para encontrar soluções aproximadas em problemas de otimização. Este algoritmo utiliza técnicas que foram inspiradas pela biologia evolutiva como seleção natural, hereditariedade, mutação e recombinação. AG não é um algoritmo de busca de solução

ótima, mas sim uma heurística que encontra boas soluções, já que o mesmo utiliza da aleatoriedade, pode encontra soluções diferentes a cada execução, podendo esses serem ótimo locais ou globais (Linden, 2012).

O primeiro AG proposto por Holland é conhecido como Standart Genetic Algorithm ou Simple Genetic Algorithm é descrito em 6 passos.

1 - Inicie uma população com os cromossomos gerados aleatoriamente, de tamanho N.

2 - Avalie cada cromossomo da população utilizando a função de avaliação.

3 - Gere uma nova população de tamanho N a partir do cruzamento de cromossomos selecionados da população anterior. Aplique mutação nestes cromossomos.

4 - Remova a população anterior, trocando pela nova população criada.

5 - Avalie cada cromossomo da população utilizando a função de avaliação.

6 - Caso a solução ideal seja encontrado, ou o tempo do algoritmo se esgote (número máximo de gerações, ou avaliações, ou algum outro critério de parada), retorna o cromossomo com a melhor avaliação. Caso contrario retorne para o passo 3.

5.1. Operador de seleção

O operador de seleção define quais os indivíduos que serão selecionados para participar da reprodução e gerar a nova geração. Esse método deve simular o mecanismo de seleção natural que age nas espécies biológicas (Linden, 2012). O seu objetivo é privilegiar os indivíduos com avaliações altas, mas sem desprezar completamente os com avaliações baixas.

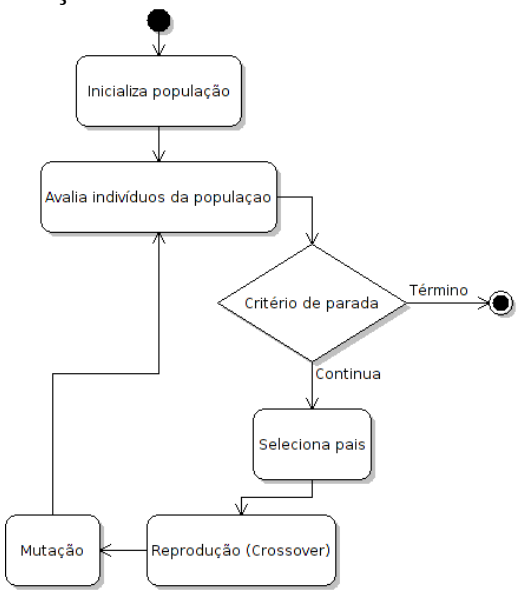


Figura 2: Diagrama de sequência do AG. Fonte: Autor (2014)

6. Algoritmos genéticos contínuos

O primeiro AG desenvolvido utiliza uma representação binaria em seu cromossomos, sendo historicamente importante pois foi utilizado nos trabalhos pioneiros de John Holland em 1975 sobre AG (Carvalho, 1999). O AG binário funciona muito bem quando a variável do problema for quantizada. Quando o espaço de busca do problema for contínuo (representado em $\mathbb{R}$) é mais coerente representá-las utilizando

uma representação de números em ponto flutuante. De acordo com (Linden, 2012) utilizar cromossomos reais torna igual o genótipo (representação interna) com o fenótipo(representação externa ou valor utilizado no problema). Acabando com os efeitos de interpretação entre as duas representações.

7. Projeto

A implementação do AG foi realizado utilizando a linguagem de programação Octave, já que não existe nenhuma ferramente implementada neste linguagem de um AGC para resolver problemas genéricos de otimização no domínio dos reais.

Foi utilizado o COCO (COmparing continuous Optimisers, em português, comparador de otimizadores contínuos) (COCO, 2014) que é uma ferramenta de análise (benchmarking) de teste de caixa preta para algoritmos de otimização contínuos.

8. Experimentos e Resultados

Esse capítulo apresenta os resultados dos experimentos realizados, e uma interpretação dos mesmos. Para comparar o quanto afeta cada operador no desempenho global do AGC, para cada grupo de testes foram utilizados os mesmos parâmetros, exceto os parâmetros os que se desejam comparar.

Resultados para diferentes mutações. Foram testados os métodos de mutação *uniforme* e *não uniforme*. Para esses testes foram utilizadas a população fixa de 250, cruzamento BLX-alfa e seleção por torneio e taxa de mutação a 1% e 5%.

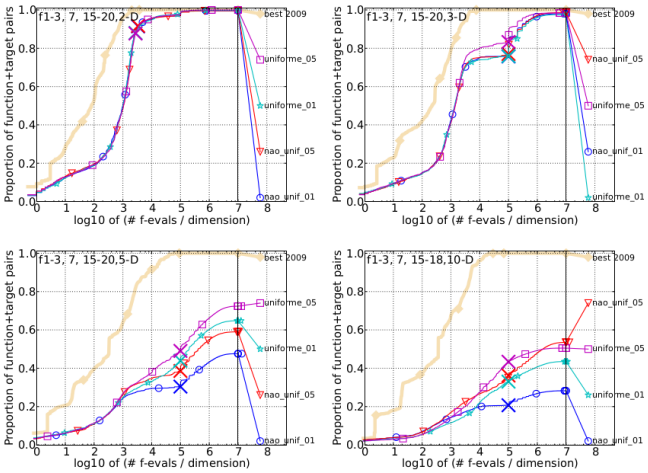


Figura 3: Resultados dos testes para as diferentes mutações. O gráfico corresponde a proporção de acertos pelo número de avaliações / dimensões em log10 para as funções 1-3, 7, e 15-20 do COCO (COCO, 2013). Da direita para esquerda, de cima pra baixo, resultados para 2, 3, 5 e 10 dimensões respectivamente. O ponto marcado com X foram o número máximo de avaliações realizadas.

9. Considerações finais

Neste trabalho foi demonstrado que podemos utilizar otimizadores genéricos como o algoritmos genéticos de forma eficiente para resolver problemas de otimização continua com múltiplas variáveis.

Os resultados obtidos foram satisfatórios para as funções testadas, porém com o curto tempo empregado na pesquisa, impossibilitou a realização de testes mais amplos. Portanto mesmo com um número menor de variáveis foi possível demonstrar a

empregabilidade dessas técnicas de computação evolutiva para solucionar problemas complexos.

É importante a realização da comparação entre o desempenho com vários parâmetros diferentes de entrada, pois mesmo que não seja possível definir um conjunto de parâmetros que obtenha o melhor desempenho para qualquer problema tratado, podemos obter conjuntos que sejam promissores para um grupo de funções.

As funções utilizadas neste trabalho possuem o ótimo conhecido, já que utilizamos o COCO (COCO, 2014), cuja função é testar o desempenho dos algoritmos. Entretanto em problemas práticos não conhecemos o ponto ótimo. Caso utilizássemos funções das quais não conhecemos o seu ponto ótimo, podemos definir algum conjunto de parâmetros que tenha um melhor desempenho, mas não teríamos o quão próximo chegou do ótimo. Por isto é importante testarmos o AG com funções nas quais conhecemos o ponto ótimo, para que possamos determinar que parâmetros do AG podemos utilizar quando tivermos que utilizá-lo em situações reais.

Como trabalhos futuros pode ser implementado um mecanismo de população variáveis, onde o AG inicializa com uma população grande para cobrir uma área maior do domínio do problema, e conforme os indivíduos forem convergindo, pode-se reduzir a sua população. Também pode ser implementado formas de inicializar a população de forma que os indivíduos fiquem distribuídos de forma mais homogênea na área de busca. Ainda poderá a implementação para a paralelização do processamento do AG no processo de seleção, reprodução, mutação e avaliação, já que os mesmos em cada etapa não depende do resultado dos outros indivíduos. Desta forma a execução poder utilizar mais de um núcleo do processador ou até mesmos outras máquinas disponíveis pela rede, reduzindo o tempo de cada execução.

Referências

CARVALHO, E. A. Sistemas inteligentes: aplicações a recursos hídricos e ciências ambientais. Porto Alegre: Universidade/UFRGS : Associação Brasileira de Recursos Hídricos., 1999. cap. Capitulo 3 :Introdução aos algoritmos genéticos., p. 99–150.

COCO. COmparing Continuous Optimisers: COCO. 2013. <http://coco.gforge.inria.fr>.

LINDEN, R. Algoritmos Genéticos. 3 edição. ed. [S.l.]: Editora Ciência Moderna, 2012.

SHAMSUDDIN, Z. S. A review of population-based meta-heuristic algorithms. Int. J.

Advance. Soft Comput. Appl., Vol. 5, No. 1, March 2013, 2013.

SARAMAGO, S. Métodos de Otimização Randômica: algoritmos genéticos e “simulated annealing". 2012. <http://www.sbmac.org.br/arquivos/notas/livro_06.pdf>.

NORVIG, S. P. Inteligência Artificial. 2 edição. ed. [S.l.]: Editora Ciência Moderna, 2012.

STEWART, J. Cálculo. 5. ed. [S.l.]: Cengage Learning, 2008.