UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

Algoritmos Genéticos

Computação Inspirada pela Natureza

Aluno: Samuel de Souza Lopes

Professor: Dr. Fabricio Aparecido Breve

2018

1. Introdução e Objetivos

Desde os primórdios da computação, diversos cientistas buscam criar inteligência e vida artificial. Alan Turing, John von Neumann, Norbert Wiener, entre outros cientistas da computação foram motivados principalmente em criar programas de computador providos de inteligência, com a habilidade semelhante à vida de se reproduzir e com a capacidade de aprendizado adaptativo e de controlar seus ambientes. Esses pioneiros da ciência da computação estavam tão interessados em biologia e psicologia quanto em eletrônica, e olhavam para os sistemas naturais como metáforas orientadoras de como alcançar suas visões. Essas atividades de computação biologicamente motivadas aumentaram e diminuíram ao longo dos anos, mas desde o início da década de 1980 passaram por um ressurgimento na comunidade de pesquisa em computação. Dentre essas atividades, a computação evolucionária, da qual os algoritmos genéticos estão presentes, é um campo bastante explorado (MITCHELL, 1998). Tais algoritmos são simulações dos conceitos de seleção natural como crossover e mutação para resolver problemas de otimização. Esta classe de algoritmos evolutivos são os mais antigos, conhecidos e amplamente utilizados (SIMON, 2013). Sua versão mais popular foi apresentada em 1975 por J. Holland (DE CASTRO, 2006).

Diante do exposto, o objetivo deste trabalho é mostrar três aplicações relacionadas a algoritmos genéticos, a saber: Reconhecimento de Padrões, Maximização de Funções e Minimização de Funções. Na seção 2, os detalhes das implementações, bem como os testes e resultados são mostrados. Por fim, na seção 3 são feitas conclusões acerca do trabalho.

2. Aplicações

Nesta seção são detalhadas as implementações de três aplicações relacionadas a algoritmos genéticos, a saber: reconhecimento de padrões, maximização de funções e minimização de funções. A linguagem de programação e o ambiente de desenvolvimento utilizado foram *Python* e *PyCharm*, respectivamente.

Ao final de cada implementação, foram feitos testes e os resultados obtidos foram discutidos. Vale ressaltar que os algoritmos implementados são estocásticos, ou seja, com

diferentes execuções os gráficos e valores ótimos encontrados podem variar. As especificações do computador utilizado para fazer os testes de execução dos algoritmos implementados é:

- **Processador:** Intel® Core™ i3-3217U CPU @ 1.80GHz × 4;
- **Memória RAM:** 4 GB;
- **Placa de Vídeo:** Intel® Ivybridge Mobile (integrada)
- **Sistema Operacional:** Ubuntu 16.04 LTS.

2.1. Reconhecimento de padrões

O objetivo desta aplicação é implementar um algoritmo genético que reconheça o padrão, representado pela *bitstring*, [1 1 1 1 0 1 1 0 1 1 1 1], sendo que a população utilizada é composta por 8 indivíduos.

Detalhamento do algoritmo

Para a implementação deste algoritmo, cada indivíduo é representado por uma *bitstring* de 12 caracteres, sendo que a população é representada por uma matriz de 8 linhas e 12 colunas, onde cada linha representa um indivíduo da população.

Inicialmente, o algoritmo cria uma população aleatória e seus indivíduos são avaliados. A função de aptidão avalia os indivíduos desta população, calculando a quantidade de bits iguais entre cada indivíduo da população e o padrão a ser reconhecido. Quanto maior a semelhança entre o indivíduo e o padrão, maior é a sua aptidão. Caso algum indivíduo seja igual ao padrão procurado, ou seja, a aptidão do indivíduo é 12, a execução do algoritmo é encerrada. Caso contrário, o algoritmo executa a etapa de seleção em que é selecionado inicialmente, via elitismo, os dois indivíduos da população atual com maior aptidão para compor a geração subsequente. Os outros indivíduos da geração seguinte são selecionados via torneio, em que 3 indivíduos da população são escolhidos aleatoriamente para formar uma sub-população temporária. Deste grupo, o indivíduo com a melhor aptidão é selecionado para compor a geração subsequente. Esta técnica é executada seis vezes, sendo que seis indivíduos são selecionados e se juntam aos outros dois indivíduos selecionados via elitismo para compor a nova população. Esta nova população pode passar por crossover ou por mutação, de forma a modificar os bits que a representam. O tipo de crossover utilizado foi o de 1 ponto, em que

apenas um ponto de crossover é selecionado em ambos os pais e todos os bits além do ponto são trocados entre os progenitores, gerando dois filhos. Ao final da iteração, a nova população é avaliada e se algum indivíduo for igual ao padrão, o algoritmo retorna a quantidade de iterações necessárias para encontrá-lo. Caso contrário, a etapa de seleção é iniciada, onde a entrada é a nova população.

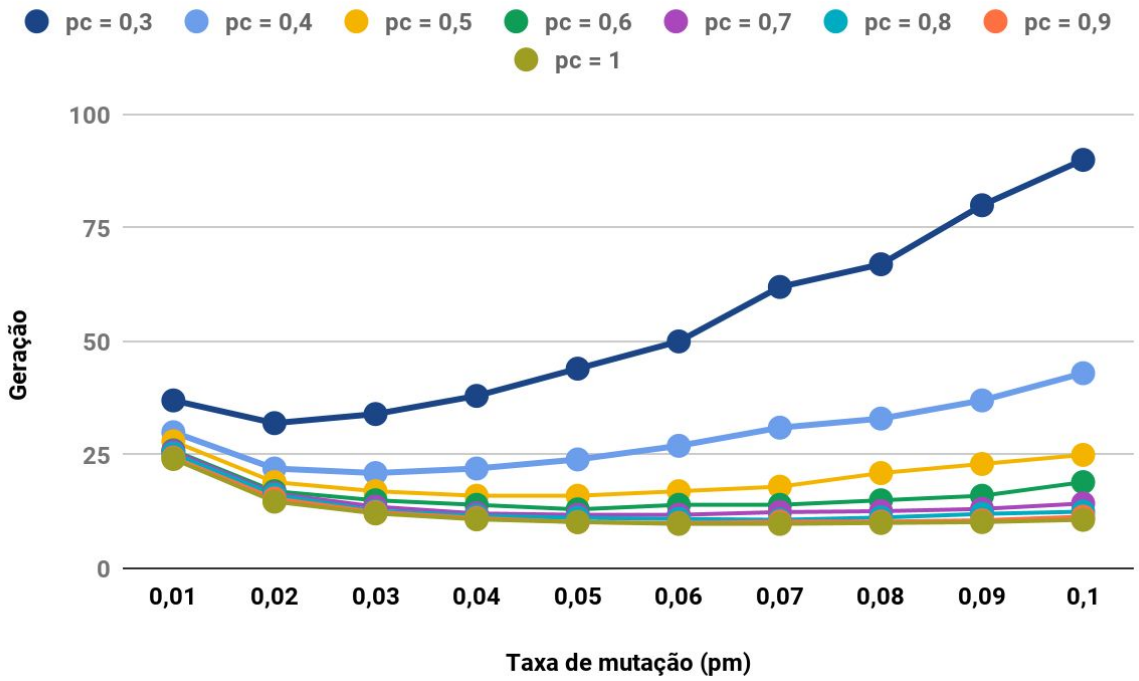
Experimentos e Testes

Ao final da implementação do algoritmo, testes foram feitos, sendo divididos em 3 etapas, a saber:

1. Desempenho do algoritmo em diferentes taxas de crossover e mutação, bem como o número de iterações e o tempo que os algoritmos demoram para encontrar o padrão;
2. Comportamento do algoritmo em situações que ocorre apenas crossover ou apenas mutação;
3. Gráfico que mostre o valor mínimo, máximo e médio da função de aptidão ao longo das iterações.

Na etapa 1, o algoritmo foi testado com diferentes taxas de crossover (pc) e mutação (pm), de forma a identificar os valores de pc e pm que maximizem o desempenho do algoritmo. A Figura 1 mostra um gráfico contendo a média de 1500 execuções do algoritmo em diferentes combinações de pc e pm, onde o eixo horizontal representa as taxas de pm e o eixo vertical representa a média de gerações necessárias para encontrar o padrão em cada combinação de pc e pm. É possível perceber que quanto maior o valor de pc, menos gerações são necessárias para encontrar o padrão, sendo que os melhores valores de pc estão no intervalo [0,7; 1] e de pc no intervalo [0,6; 0,1]. Nos testes de execução, a melhor média foi de 9,8 iterações para encontrar o padrão, com desvio padrão de 4,54, onde pc e pm valiam 1 e 0,07, respectivamente.

Figura 1: Execução do algoritmo com diferentes valores de pc e pm



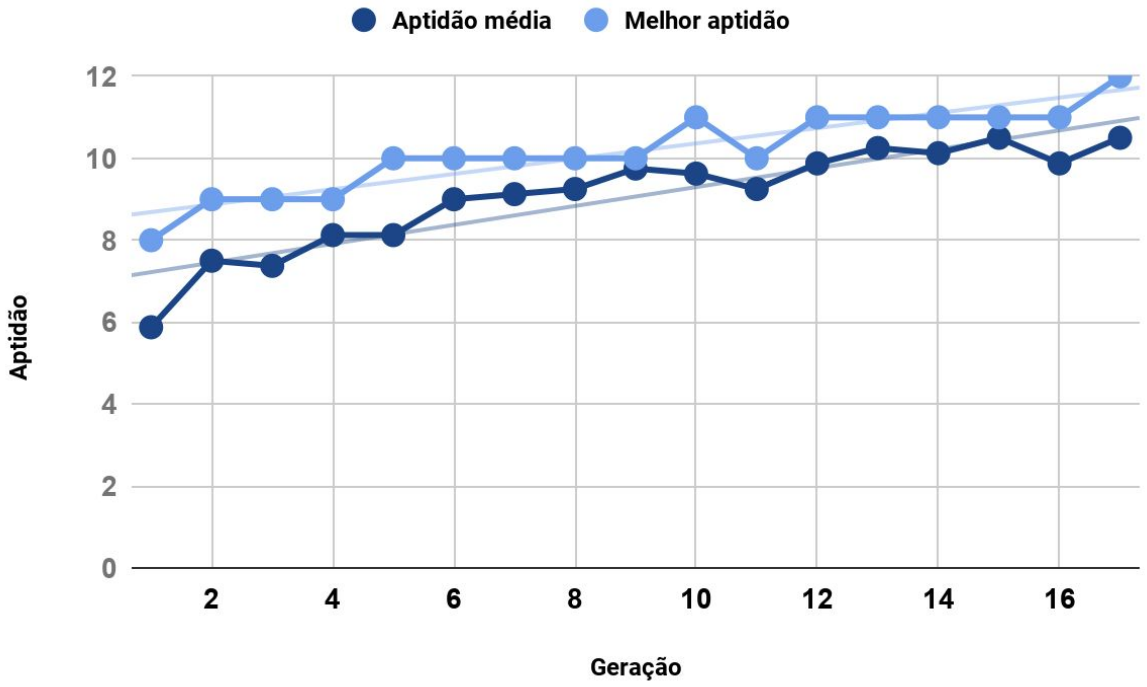
Fonte: Elaborado pelo próprio autor.

Na etapa 2, o algoritmo foi executado em condições onde promove apenas crossover e apenas mutação nas populações geradas ao longo das iterações, sendo atribuídos os valores 1 e 0,07 a pc e pm, respectivamente. Quando há apenas crossover, a aptidão mínima, média e máxima não é alterada ao longo das iterações e, consequentemente, o algoritmo não encontra o padrão. Isto ocorre porque a mutação previne que uma dada população fique estagnado em um valor, além de possibilitar que se chegue em qualquer ponto do espaço de busca. Quando o valor de pm é igual a 0, o algoritmo não consegue gerar indivíduos que consigam explorar novos pontos do espaço de busca, fazendo com que tanto a aptidão mínima, média e máxima fiquem estagnadas ao longo das iterações. Por sua vez, quando há apenas mutação, o algoritmo consegue encontrar o padrão. Como a mutação garante a introdução e manutenção da diversidade genética na população, o algoritmo encontra o padrão, porém necessita de mais iterações para encontrá-lo. Executando 100 testes e atribuindo aos parâmetros pc e pm os valores 0 e 0,7, respectivamente, o algoritmo executou, em média, mais de 1000 iterações para encontrar o padrão. Isto era esperado pois, mesmo sem crossover, a mutação permitiu a introdução e a manutenção da diversidade genética dos indivíduos ao longo da iteração,

porém a falta de crossover diminuiu a velocidade de mudança dos indivíduos ao longo das gerações.

Na etapa 3, o algoritmo foi executado com os valores atribuídos a pc e pm foram 1 e 0,09, respectivamente, sendo que a melhor aptidão e a aptidão média, ao longo das iterações, foram armazenadas. Por meio da Figura 2, que mostra um gráfico onde o eixo horizontal representa as gerações e o eixo vertical mostra a maior aptidão e a aptidão média, é possível observar que tanto a aptidão máxima quanto a média aumentam ao longo das iterações. Isto ocorre porque o elitismo assegura que os indivíduos mais aptos estarão presentes nas gerações subsequentes, garantindo que tanto a aptidão média quanto a máxima tendam a aumentar ao longos das gerações.

Figura 2: Melhor aptidão e aptidão média ao longo das iterações



Fonte: Elaborado pelo próprio autor.

2.2. Maximização de funções

O objetivo da aplicação 2 é implementar um algoritmo genético que maximize a função $g(x) = 2^{-2((x-0.1)/0.9)^2}(sin(5\pi x))^6$ no intervalo contínuo [0,1], utilizando uma representação de *bitstring*. Neste intervalo, o valor máximo de g(x) é 1.

Detalhes da implementação do algoritmo

A população utilizada é composta por 30 indivíduos, sendo que cada indivíduo é representado por uma *bitstring*, em que a parte inteira é representada por 1 bit e a parte fracionária é representada por 24 bits, conforme mostra a Figura 3. Para que os valores de x possam ter precisão de 7 casas decimais, foi necessário utilizar 24 bits para representar a parte fracionária.

Figura 3: Representação de um indivíduo da população.

Fonte: Elaborado pelo próprio autor.

Inicialmente, o algoritmo cria uma população aleatória e seus indivíduos são avaliados. A função de aptidão avalia os indivíduos da população, convertendo-os para valores decimais e calculando g(x), sendo que os indivíduos mais aptos são os que possuem maior valor de g(x). Estes valores são utilizados para selecionar os indivíduos mais aptos para compor a geração subsequente. Na etapa de seleção, os dois indivíduos mais aptos são selecionados, via elitismo, para compor a nova população. Os outros indivíduos da nova população são selecionados via torneio, como utilizado na Aplicação 1, sendo que o critério de seleção é o indivíduo com maior valor de g(x). Com a nova população formada, seus indivíduos podem passar pelos processos de crossover de 1 ponto e mutação e, após passar pelos dois processos, a nova população é avaliada e caso os últimos 1000 valores de g(x) calculados forem diferentes em menos de 10^{-7}, o algoritmo entende que houve convergência, retornando o maior valor de g(x).

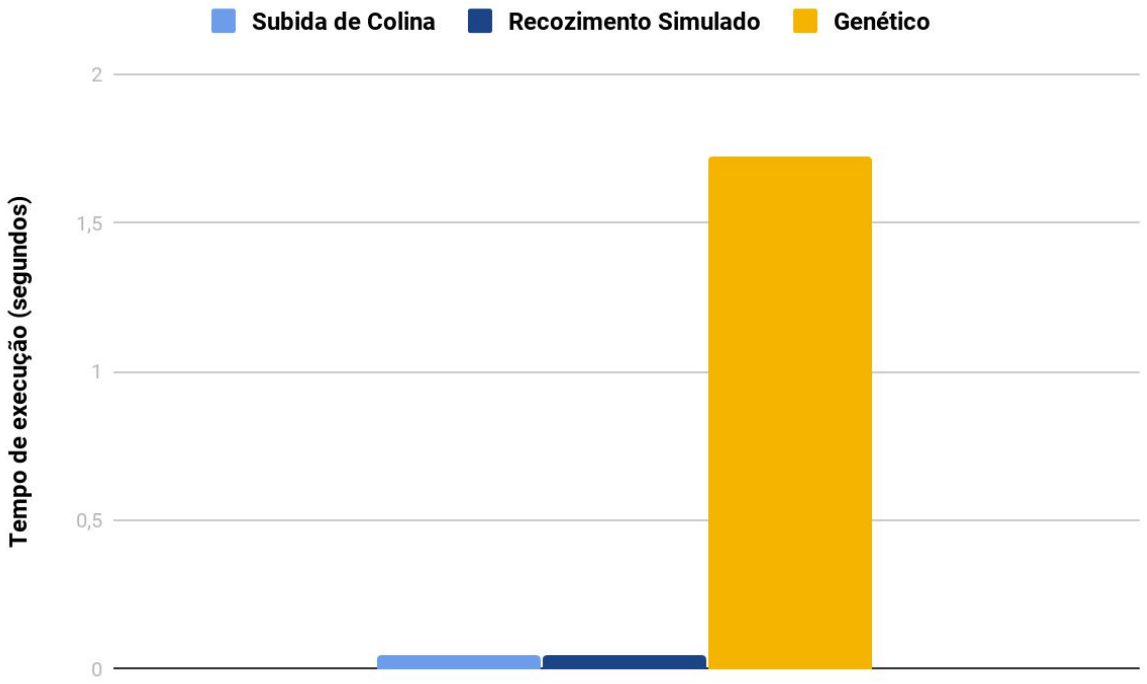
Experimentos e Testes

Ao final da implementação do algoritmo, testes foram feitos, sendo divididos em 2 etapas, a saber:

1. Comparação com os resultados obtidos com os algoritmos subida da colina e recozimento simulado;

2. Gráficos que mostrem o valor máximo e médio da função de aptidão ao longo das iterações.

Figura 4: Tempo de execução dos algoritmos até convergir.



Fonte: Elaborado pelo próprio autor.

Na etapa 1, os algoritmos subida de colina e recozimento simulado foram testados para calcular max g(x) - máximo global de g(x) . Os dois algoritmos utilizam o mesmo critério de convergência utilizado no algoritmo genético proposto, sendo que foram feitos 20 testes em cada algoritmo. Pela Tabela 1, é possível perceber que os valores de max g(x) propostos pelo algoritmo genético estão mais próximos e menos dispersos de 1 do que os resultados propostos pelo subida de colina e recozimento simulado, atingindo eficácia de 90%. No entanto, a Tabela 1 e a Figura 4 mostram que o algoritmo genético possui custo computacional significativamente maior se comparado aos outros algoritmos estudados pois, mesmo executando quantidades parecidas de iterações, o algoritmo genético demanda mais tempo para executar seus passos. A principal diferença entre essas três abordagens é que os algoritmos evolutivos são técnicas de pesquisa baseadas em população, enquanto o subida de colina e recozimento simulado trabalham com um único indivíduo. Assim, como o algoritmo

genético implementado avalia 30 indivíduos a cada iteração, além de executar funções de crossover e de mutação para perturbar os bits da população ao longo das gerações, o algoritmo genético executa mais passos que os outros algoritmos estudados pois tanto o subida de colina quanto o recozimento simulado avaliam e perturbam apenas 1 valor de x a cada iteração.

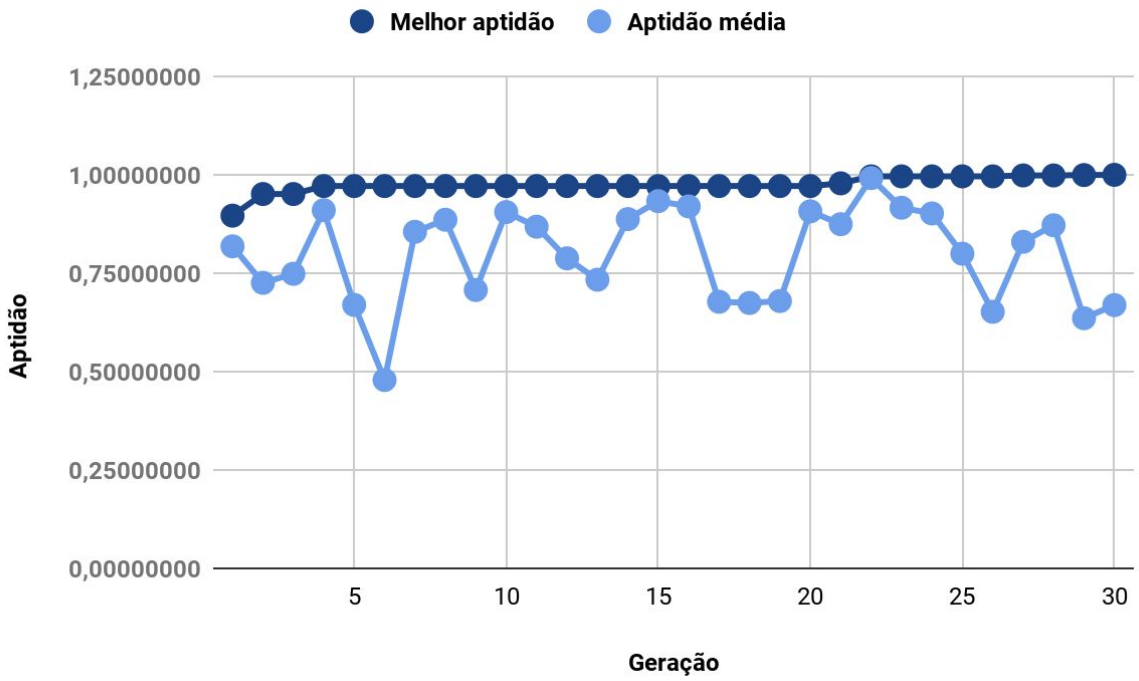
Tabela 1: Comparativo entre os algoritmos estudados

	Subida de Colina	Recozimento SImulado	Genético
Porcentagem de vezes que o algoritmo atingiu valores próximos a max g(x)	25%	30%	90%
Média dos valores de max g(x)	0,65852084	0,7310754	0,9919669
Desvio padrão dos valores de max g(x)	0,2387436	0,2419547	0,0203046
Tempo médio de execução (segundos)	0,0440546	0,0501303	2,8780335
Média de iterações para convergir	1486,25	1468,9	1756
Desvio padrão para convergir	338,73	292,32	634,29

Fonte: Elaborado pelo próprio autor.

Por fim, na etapa 2 o algoritmo foi executado para verificar o comportamento da melhor aptidão e da aptidão média ao longo das gerações, conforme ilustrado na Figura 5, onde o eixo horizontal representa as gerações e o eixo vertical representa o melhor valor e o valor médio da função de aptidão. É possível observar que a melhor aptidão tende a aumentar ao longo das gerações, de forma a predominar os indivíduos da população mais aptos ao longo do tempo, ou seja, indivíduos que possuam os valores de x que , aplicados a g(x), gerem valores próximos a 1. O valor médio de aptidão oscila ao longo das gerações pois a busca por uma melhor aptidão é feita ao longo de toda a execução, sendo que a melhor aptidão é pouco alterada por conta do elitismo.

Figura 5: Melhor aptidão e aptidão média ao longo das iterações.



Fonte: Elaborado pelo próprio autor.

2.3. Minimização de funções

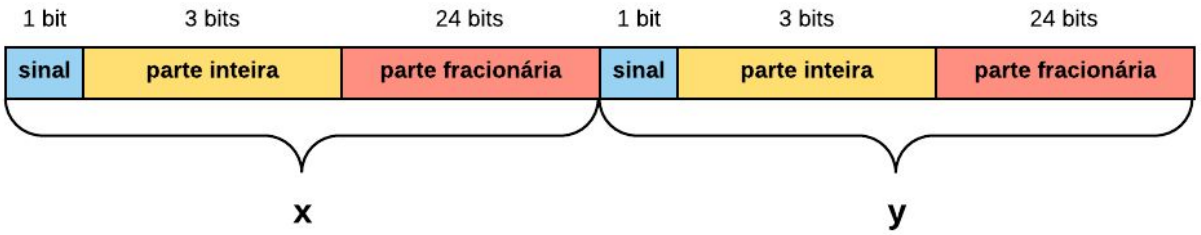
O objetivo da aplicação 3 é implementar um algoritmo genético que minimize a função $f(x,y) = (1-x)^2 + 100(y-x^2)^2$, onde x e y variam no intervalo [-5, 5], utilizando uma representação de *bitstring*. Neste intervalo, o valor mínimo de f(x,y) é 0.

Detalhamento do algoritmo

A população utilizada é composta por 30 indivíduos, sendo que cada indivíduo é representado em uma linha da matriz da população. Conforme mostra a Figura 6, cada indivíduo armazena uma dupla (x,y), sendo que x e y são compostos pelo bit de sinal, onde 0 indica que o valor é negativo e 1 indica que é positivo; pela parte inteira, representado por 3 bits; e pela parte fracionária, representado por 24 bits. Assim como na aplicação anterior, o resultado obtido possui precisão de 7 casas decimais.

Este algoritmo foi implementado de forma similar a Aplicação 2, porém, como o objetivo é minimizar f(x,y), o elitismo seleciona os menores valores de f(x,y), assim como os menores valores são selecionados via torneio.

Figura 6: Representação de um indivíduo da população.

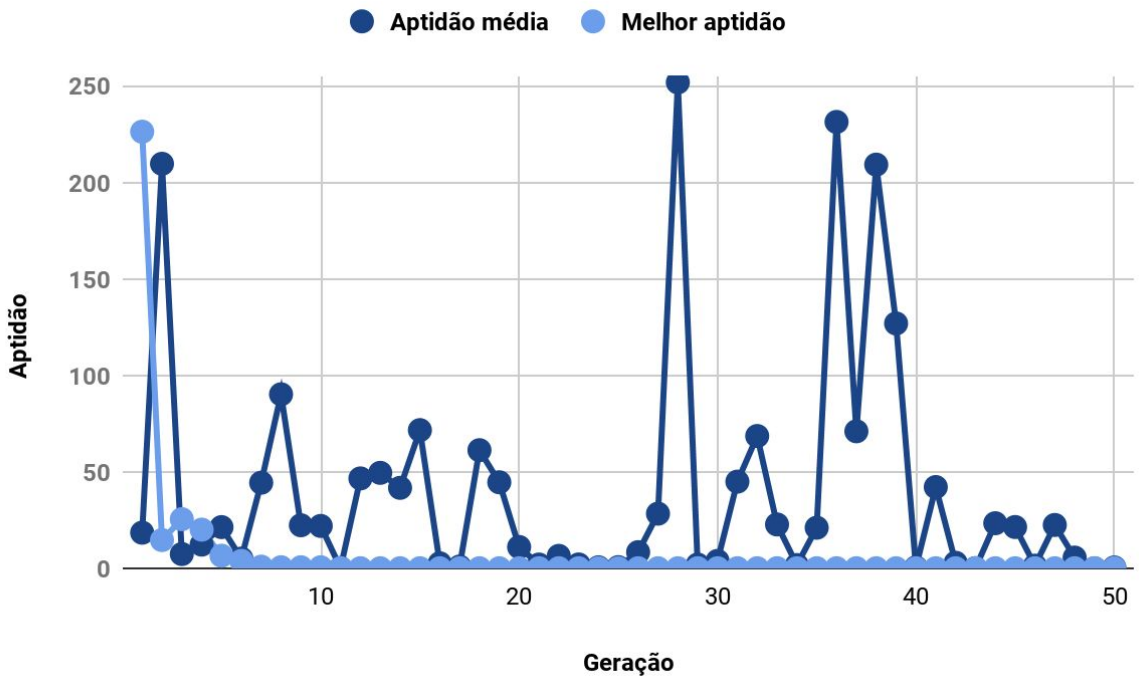


Fonte: Elaborado pelo próprio autor.

Experimentos e Testes

Ao final da implementação do algoritmo, o mesmo foi executado para verificar o comportamento do valor máximo e médio da função de aptidão ao longo das primeiras 35 gerações, conforme ilustrado na Figura 7, que mostra a variação da melhor aptidão e da aptidão média dos elementos a cada geração. É possível observar que a aptidão média tende a diminuir para valores próximos a zero já nas primeiras gerações, mostrando a predominância dos indivíduos da população mais aptos ao longo do tempo, ou seja, indivíduos que possuam os valores de x e y que, aplicados a f(x,y), gerem valores próximos a 0. Multiplicando a aptidão média por 10^{-2}, é possível perceber que os valores médios de aptidão oscilam ao longo das gerações. Isto ocorre porque a busca por uma aptidão melhor é contínua, sendo que a melhor aptidão é pouco alterada por conta do elitismo.

Figura 7: Melhor aptidão e aptidão média ao longo das iterações.



Fonte: Elaborado pelo próprio autor.

3. Conclusões

O algoritmo de reconhecimento de padrões obteve melhor desempenho quando as taxas de crossover e mutação estiveram no intervalo [0,7; 1] e [0,06; 0,09], respectivamente. Para encontrar uma *bitstring* de 12 bits no espaço de busca gerado, o algoritmo foi bastante eficaz já que necessitou, em média, 10 iterações, com desvio padrão menor que 5, quando os valores atribuídos às taxas de mutação e crossover foram 1 e 0,9, respectivamente. O algoritmo só é eficaz se encontrar o padrão. Assim, quanto mais explorado é o espaço de busca, melhor é o desempenho do algoritmo pois altas taxas de crossover aumentam as chances de novos indivíduos surgirem na população.

Para maximizar funções, o algoritmo genético se mostrou mais eficaz do que o subida de colina e o recozimento simulado já que, mesmo executando uma quantidade parecida de iterações e utilizando o mesmo critério de convergência, o algoritmo genético, seguido do recozimento simulado, convergiu para valores próximos do máximo global da função mais vezes do que o subida de colina. Devido a natureza estocástica, o recozimento simulado e o algoritmo genético são capazes de escapar de soluções ótimas locais. A exploração mais

ampla do espaço de busca no algoritmo genético é realizada pelo uso de múltiplos indivíduos e operadores genéticos, que permitem a criação de indivíduos com valores distintos e também o compartilhamento de características com seus pais. Enquanto o subida de colina e o recozimento simulado geram novos pontos candidatos na vizinhança do ponto atual, o algoritmo genético permite a análise de pontos na vizinhança de duas ou mais soluções candidatas através do uso de operadores genéticos, como crossover e mutação (DE CASTRO, 2006).

Por fim, o algoritmo genético que objetiva minimizar f(x,y) foi implementado. Efetuando 100 testes de execução, onde 300 iterações foram executadas em cada execução, o algoritmo se mostrou eficaz pois o valor médio do mínimo global de f(x,y) obtido foi de 0,2618372, com um desvio padrão de 0,3912464. Assim, podemos concluir que o algoritmo converge para valores próximos ao mínimo global em poucas gerações.

Diante do exposto, o que se pode concluir é que algoritmos genéticos são uma boa opção para resolver problemas que não apresentam uma única solução, além de simplificar a formulação e solução de problemas de otimização. Se o objetivo é obter soluções ótimas próximas ao ideal, como no problema de minimização de funções, algoritmos genéticos podem contribuir para encontrar soluções ótimas para esta classe de problemas.

4. Referências Bibliográficas

DE CASTRO, Leandro Nunes. **Fundamentals of natural computing: basic concepts, algorithms, and applications**. CRC Press, 2006.

MITCHELL, Melanie. **An introduction to genetic algorithms**. MIT press, 1998.

SIMON, Dan. **Evolutionary optimization algorithms**. John Wiley & Sons, 2013.