UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA ELÉTRICA

GUSTAVO ALVES PACHECO

IMPLEMENTAÇÃO DO ‘A SIMPLE EXAMPLE’

Uberlândia-MG

2019

GUSTAVO ALVES PACHECO

IMPLEMENTAÇÃO DO ‘A SIMPLE EXAMPLE’

Trabalho apresentado ao curso de Engenharia de Computação da Universidade Federal de Uberlândia, como requisito parcial de avaliação da disciplina de Algoritmos Genéticos.

Prof.: Keiji Yamanaka

Uberlândia-MG

2019

SUMÁRIO

1. INTRODUÇÃO

Um algoritmo genético é um procedimento de busca e otimização, que se inspira na teoria da evolução e da seleção natural de Darwin, na qual os indivíduos mais aptos a sobreviverem em determinado meio tendem a deixar mais descendentes que os demais. Com isso, as características hereditárias mais favoráveis são passadas a cada geração.

O algoritmo genético não tenta simular e imitar completamente o processo evolutivo, mas se utiliza de vários aspectos e conceitos do mesmo, para alcançar o objetivo de encontrar a melhor solução para determinado problema.

Em um AG (algoritmo genético) simples, 5 processos são fundamentais. São eles:

- Inicialização da população
- Avaliação da aptidão individual
- Seleção dos mais aptos
- Recombinação entre indivíduos
- Mutações esporádicas

Tais processos são ilustrados no fluxograma da Figura 1, abaixo.

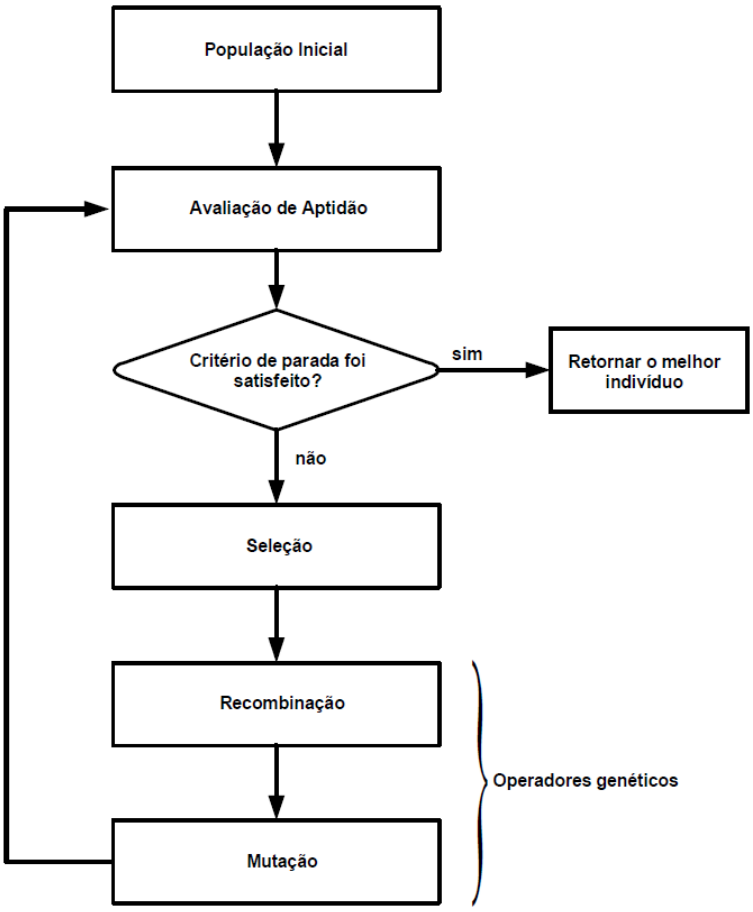

Figura 1: Esquema de um algoritmo genético básico

Um aspecto importante a se levar em conta é a forma de representação das informações a serem trabalhadas, chamadas no contexto dos AG's de cromossomos, ou indivíduos. Em algoritmos mais simples, estruturas binárias são preferíveis, visto que tal método facilita o processo de recombinação e de mutação dos cromossomos. Durante a inicialização, esses indivíduos são gerados aleatoriamente, e o conjunto deles compõe a chamada população.

Após a primeira geração, as populações passam por operadores genéticos (crossover/recombinação e mutação), que geram novos indivíduos, os quais carregam, em teoria, as melhores características das gerações anteriores.

Devido à crescente expansão da discussão sobre AG's, várias estratégias surgiram para atender diferentes tipos de problemas. Estas, por sua vez, se baseiam, em sua grande maioria, em alterações e adaptações deste modelo básico. Algumas das quais reestruturam a forma de representação, aumentando a precisão das medidas, ou alterando a estrutura de dados utilizadas. Outras modificam o procedimento de seleção de indivíduos, para que não ocorram convergências incorretas para mínimos e máximos locais e há aquelas que alteram completamente a forma de tratamento dos operadores genéticos.

Em meio a esse cenário em expansão, muito material se encontra disponível para estudo e pesquisa nas áreas de Algoritmos Genéticos. Entretanto, tanto conteúdo pode representar uma sobrecarga para o iniciante na área.

Tendo em mente essa problemática, um artigo foi desenvolvido por M. Tomassini, do *Ecole Polytechnique Federale de Lausanne*, de forma a guiar novos estudantes e entusiastas da área. O relatório a seguir descreve uma implementação para o exemplo citado no capítulo 3.

2. OBJETIVOS

- Aprimorar o conhecimento sobre algoritmos genéticos e obter experiência prática na implementação dos mesmos.
- Implementar o algoritmo genético descrito na sessão 3 do artigo "A Survey of Genetic Algorithms" de M. Tomassini.
- Realizar testes modificando os coeficientes de probabilidade (crossover e mutação) e analisar o impacto dos mesmos durante a execução.

3. MATERIAIS E MÉTODOS

Para implementação do algoritmo genético foi utilizada a linguagem de programação Common Lisp, compilando-a com o SBCL (Steel Bank Common Lisp).

Como interface de desenvolvimento, foi utilizado o Emacs, configurado com a plataforma SLIME (The Superior Lisp Interaction Mode for Emacs) para melhor produtividade.

Para confecção da interface gráfica, utilizou-se o pacote LTK, uma *binding* para CL (Common Lisp) do kit de desenvolvimento TK, que utiliza a linguagem TCL.

Como métodos, utilizou-se uma abordagem *top-down*, com auxílio de um framework SCRUM para desenvolvimento ágil. O código produzido segue majoritariamente uma abordagem funcional, mas possui elementos procedurais, para facilitar a leitura de algumas funções.

Durante o desenvolvimento, a documentação de funções se mostrou fundamental. Principalmente, para que algumas estruturas de dados fossem melhor especificadas, e a execução do código fosse correta.

4. RESULTADOS E DISCUSSÕES

No geral, a implementação seguiu à risca o que foi passado como exemplo. Por causa disso, houve um déficit de informações a respeito da quantidade de indivíduos a participarem do processo de Crossover. Logo, foi tomada a liberdade de parear aleatoriamente um número de indivíduos igual ao número de indivíduos da população. Tal estratégia se mostrou eficaz, embora menos eficiente do que utilizar números menores.

Além disso, por conta de falhas na interpretação do texto de exemplo, o código precisou ser descartado e refeito, já que na primeira versão, as operações genéticas ocorriam simultaneamente à seleção dos indivíduos aptos, sendo que na verdade deveria executar cada processo de uma vez. Por causa desse erro, a convergência não ocorria, e o processo ficava completamente aleatório.

Outra problemática encontrada, mais simples, foi a respeito da geração do número aleatório r, com o qual são comparados valores de probabilidade de crossover e de mutação. O problema se encontrava na precisão dos números gerados. Por várias vezes, a mutação não ocorria, já que os números eram gerados com resoluções baixas e sempre eram maiores que os valores de p_m, beirando a 0,01. Este erro foi corrigido gerando um número aleatório de 0 a 1000000 e o dividindo por 1000000.

A maior dificuldade encontrada possui suas origens no próprio problema a ser resolvido. Desejava-se encontrar o valor mínimo de uma função, porém a estratégia da roleta, utilizando o cálculo de aptidão informado, retornava os indivíduos de maiores valores, tendo em mente que a aptidão cumulativa era baseada no próprio valor da função, e por isso era diretamente proporcional a esse valor.

Para que se conseguisse minimizar a função, foi necessário inverter os valores dos indivíduos antes do cálculo da aptidão. Dessa forma, os indivíduos com menores valores iniciais passavam a ser os com maior aptidão e, consequentemente, fossem escolhidos mais frequentemente.

Foi extremamente importante a implementação de uma GUI (Figura 2) para visualização em tempo real das evoluções. Apenas por causa dela foi possível que diversos erros envolvendo lógica de programação fossem detectados.

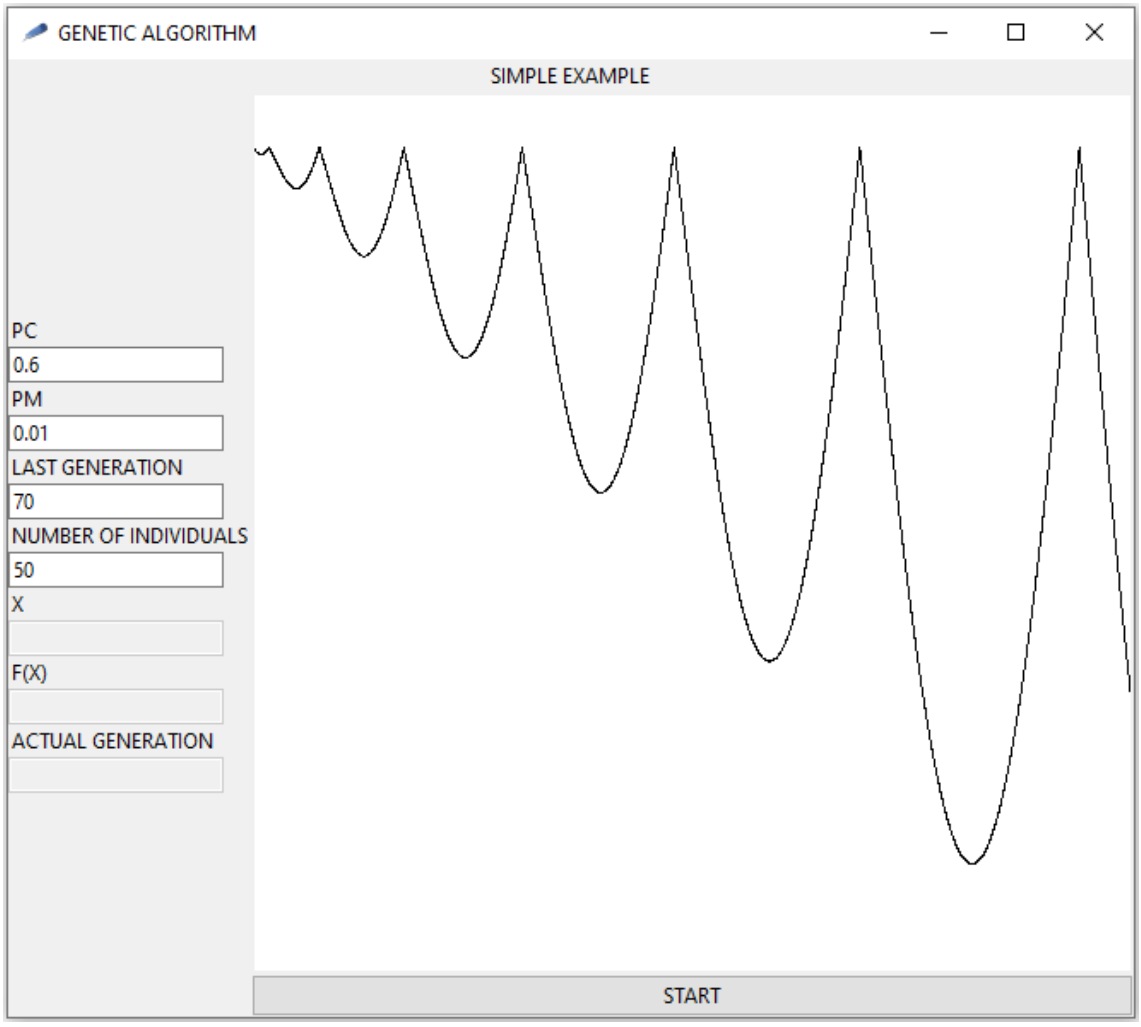

Figura 2: Interface antes da execução do código. Valores são alteráveis, mas possuem padrões igual aos citados no artigo de exemplo

Utilizando os valores recomendados no artigo, foi obtido o seguinte resultado, exibido na figura 3, abaixo. Detalhe para os campos ‘X’ e ‘F(X)’, que exibem os valores do melhor indivíduo.

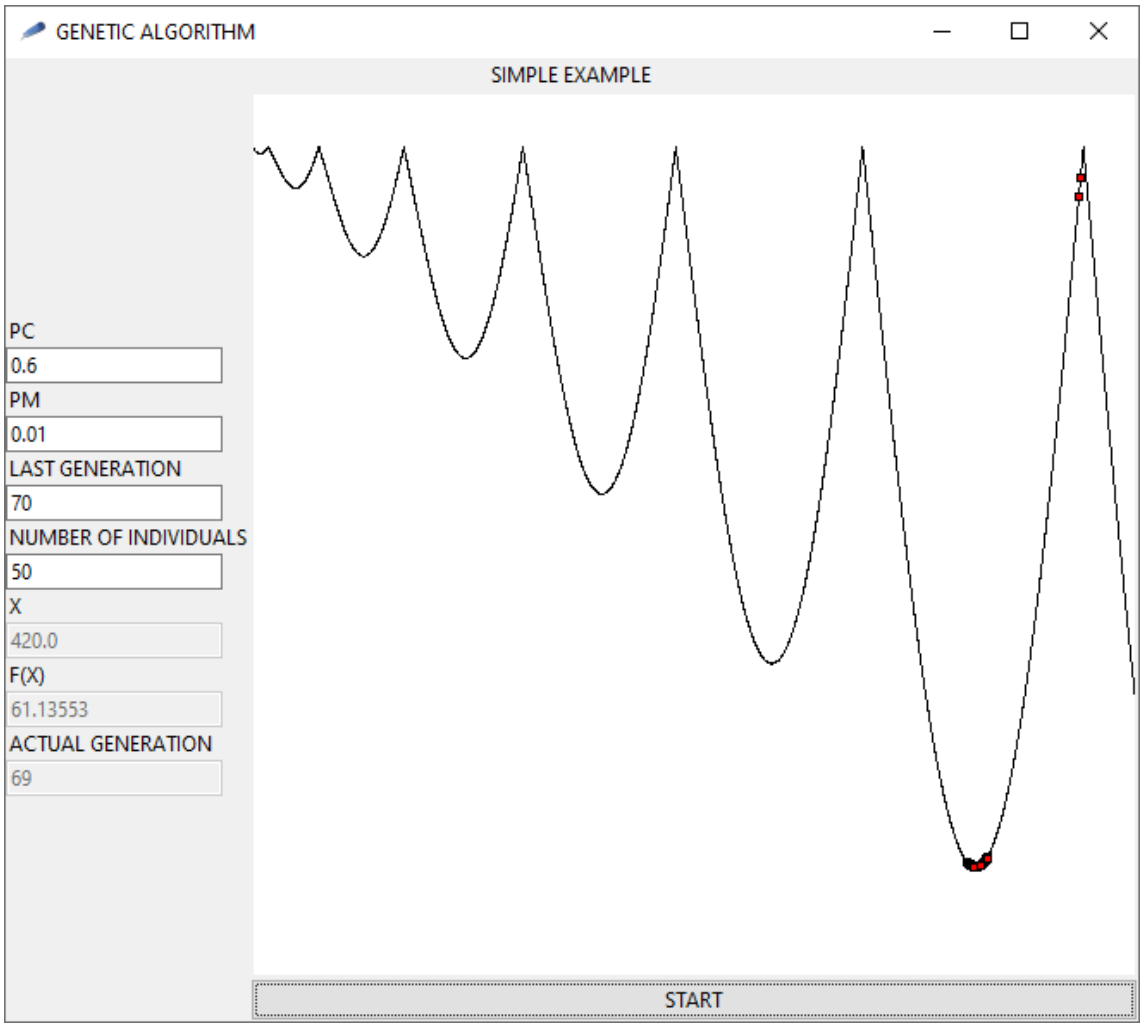

Figura 3: Interface após a execução do algoritmo, utilizando valores default

Nota-se que grande parte dos pontos convergiram para o mínimo global, mas que alguns indivíduos ainda se encontram bem longe do resultado. Resultados no qual a precisão era alta foram obtidos diminuindo o valor de p_m, como mostra a figura 4. Entretanto, essa estratégia é interessante apenas nos casos onde a amostragem geral possui baixa resolução e, com isso, um grande espectro de valores já é abordado logo após o *start*, evitando a convergência incorreta em mínimos locais.

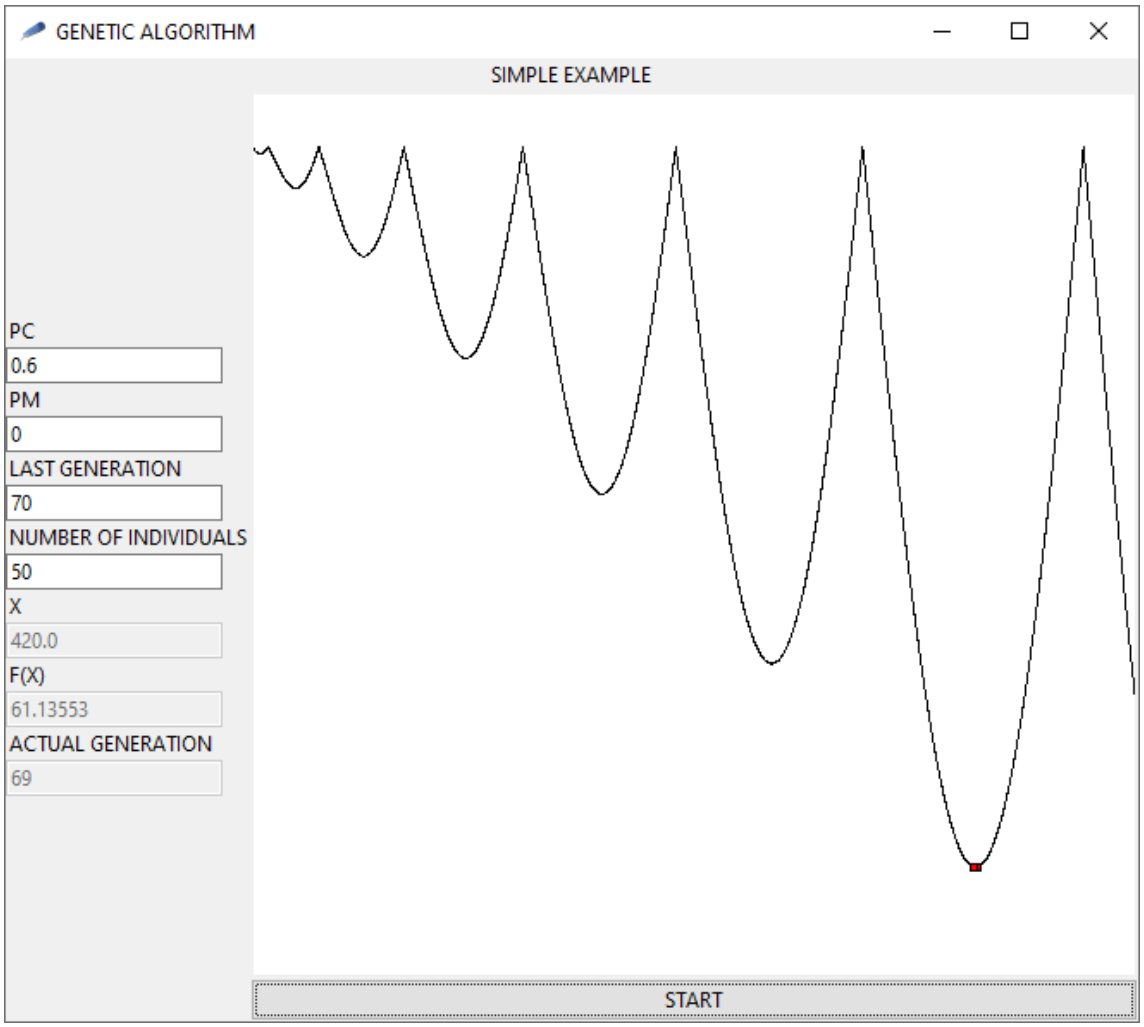

Figura 4: Algoritmo sem operação de mutação

É interessante analisar, também, a situação inversa, na qual não há o processo de crossover. Apesar de não apresentar uma convergência eficiente, tal teste se mostrou eficaz, mesmo que ainda existam indivíduos longe da solução, como mostra a figura 5. Para aplicações mais simples, essa filtragem é bem natural, mas em situações mais complexas, a convergência poderia não acontecer corretamente, ou poderiam demorar várias vezes mais que o processo com crossover.

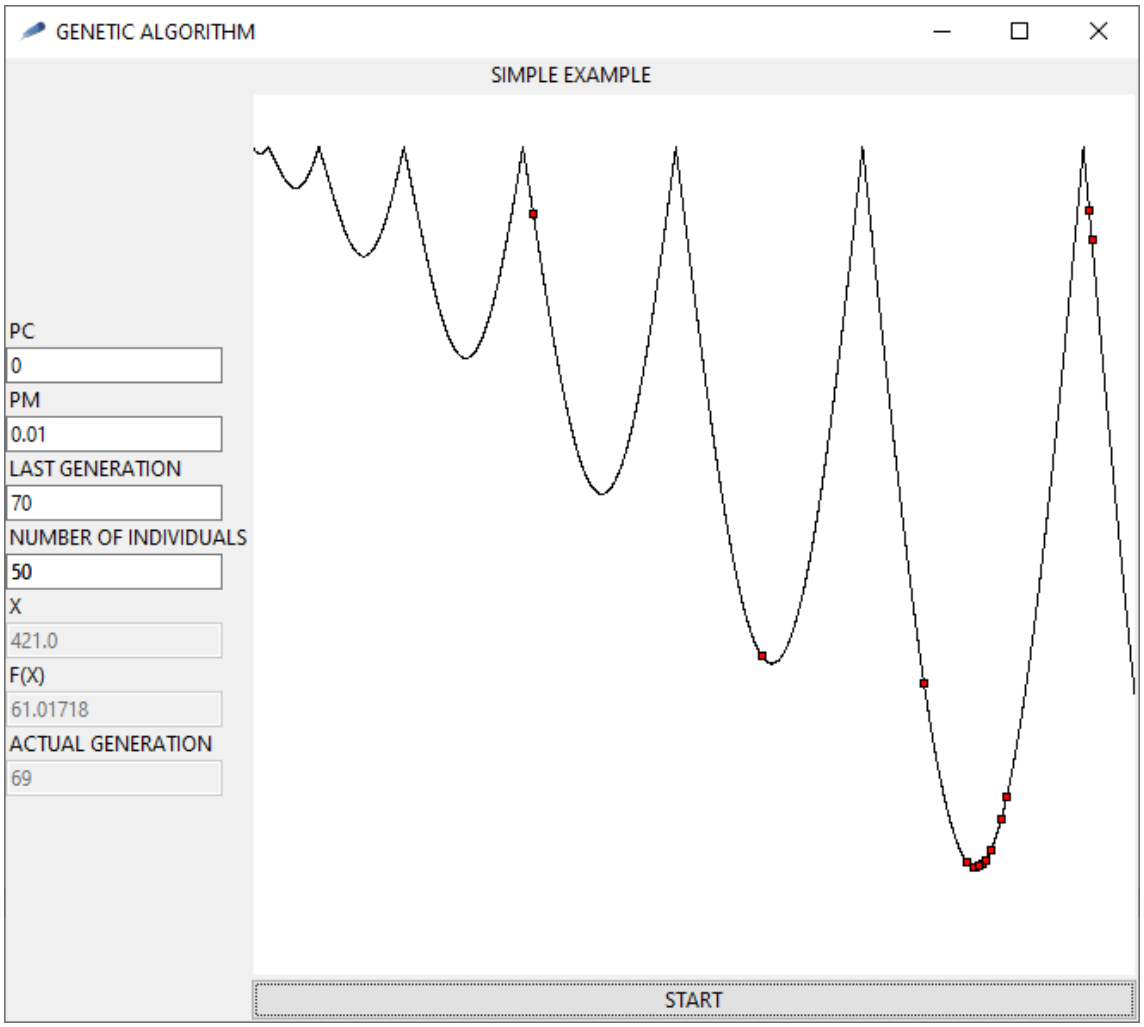

Figura 5: Algoritmo sem operação de recombinação/crossover

Um ótimo resultado foi obtido ao aumentar um pouco a taxa de crossover, e diminuir a de mutação. Com essa configuração, a convergência foi rápida e os indivíduos de fuga foram mínimos. A figura 6 ilustra essa situação.

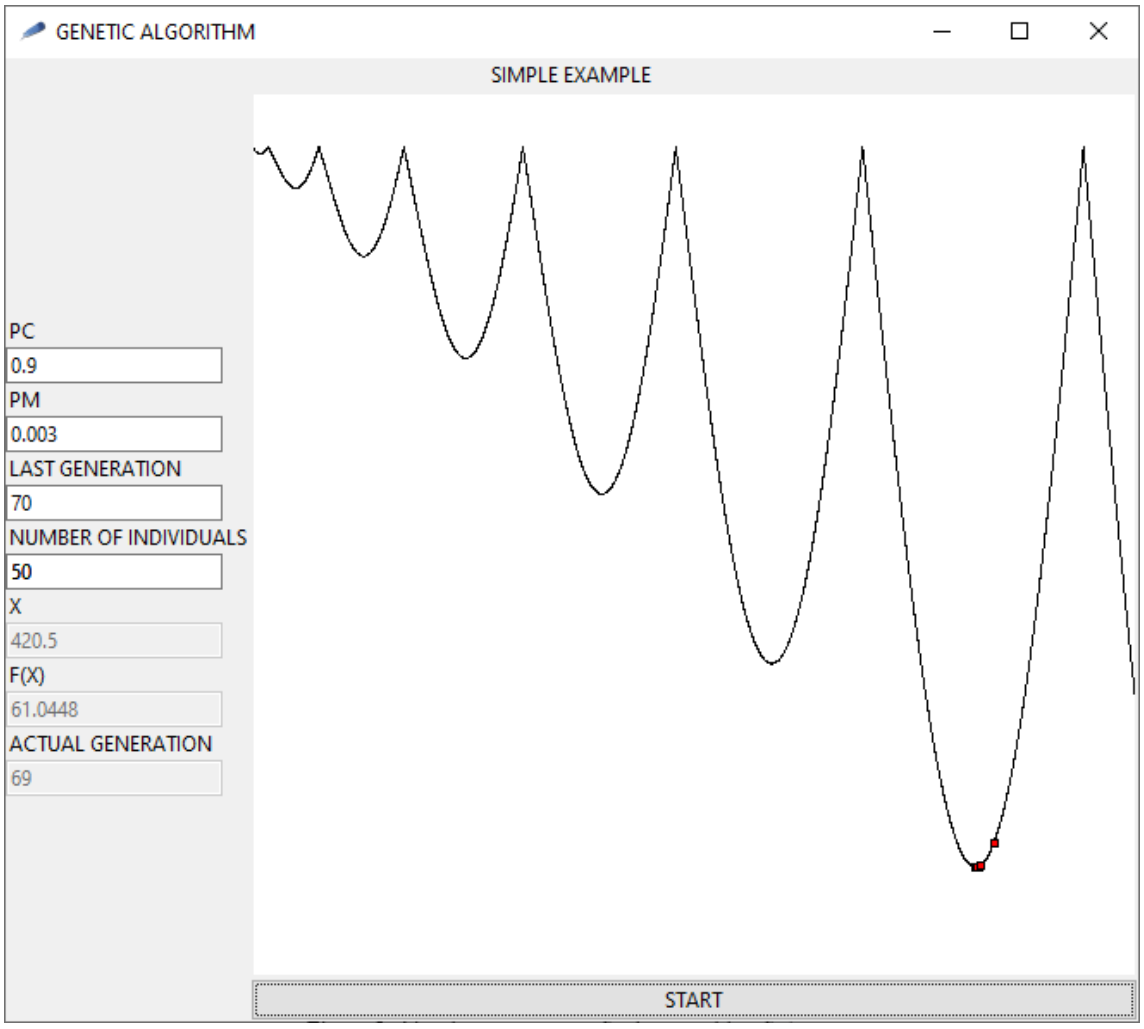

Figura 6: Situação de ótimo resultado

5. CONCLUSÃO

Apesar da resolução de amostragem ser baixa, o algoritmo foi capaz de solucionar o problema de minimização, com certo grau de confiabilidade e rapidez. É visível a possibilidade de melhoria, adotando processos e operações mais complexos. Mas para um problema mais simples, como esse, o resultado foi bem satisfatório. Com relação ao aprendizado, é indiscutível a melhoria tanto nos conhecimentos a respeito do funcionamento de AG's, quanto nas habilidades referentes à programação.

6. REFERÊNCIAS BIBLIOGRÁFICAS

Mallawaarachchi, V. (7 de Julho de 2017). *Introduction to Genetic Algorithms*. Fonte: Towards Data Science: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

Montesanti, J. d. (s.d.). *Seleção natural*. Fonte: InfoEscola: https://www.infoescola.com/evolucao/selecao-natural/

Tomassini, M. (s.d.). A Survey of Genetic Algorithms. *Annual Reviews of Computational Physics, Volume III*.

Yamanaka, P. K. (Agosto de 2017). *ALGORITMOS GENÉTICOS: Fundamentos e Aplicações*.

Zini, É. d., Neto, A. B., & Garbelini, E. (18 de Novembro de 2014). ALGORITMO MULTIOBJETIVO PARA OTIMIZAÇÃO DE PROBLEMAS RESTRITOS APLICADOS A INDÚSTRIA. *Congresso Nacional de Matemática Aplicada à Indústria*.