Ministério da Educação

Secretaria de Educação Profissional e Tecnológica

Instituto Federal Catarinense

Campus Rio do Sul

WILIAM WESSNER

SOFTWARE EDUCACIONAL DE APOIO AO ENSINO DE ALGORITMOS GENÉTICOS

Rio do Sul

2018

WILIAM WESSNER

SOFTWARE EDUCACIONAL DE APOIO AO ENSINO DE ALGORITMOS GENÉTICOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Instituto Federal Catarinense – campus Rio do Sul, para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Daniel Gomes Soares, Msc.

Rio do Sul

2018

WILIAM WESSNER

SOFTWARE EDUCACIONAL DE APOIO AO ENSINO DE ALGORITMOS GENÉTICOS

Este Trabalho de Curso foi julgado adequado para a obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo curso de Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia Catarinense – Campus Rio do Sul.

Rio do Sul (SC), 11 de dezembro de 2018

_____

Prof. e Orientador Daniel Gomes Soares, Msc.

Instituto Federal de Educação, Ciência e Tecnologia Catarinense – Campus Rio do Sul

BANCA EXAMINADORA

_____

Prof. Cristhian Heck, M.Eng.

Instituto Federal de Educação, Ciência e Tecnologia Catarinense – Campus Rio do Sul

_____

Prof. Fábio Alexandrini, Dr.

Instituto Federal de Educação, Ciência e Tecnologia Catarinense – Campus Rio do Sul

AGRADECIMENTOS

Inicialmente agradeço ao meu orientador Daniel Gomes Soares, que foi fundamental no desenvolvimento deste trabalho, dedicando-se constantemente, fazendo correções com comentários sempre pertinentes.

Agradeço aos meus pais, que me apoiaram e incentivaram desde o início do curso.

Agradeço a todos os professores da Instituição que tive o prazer de ser aluno.

Por fim, agradeço a todos os amigos e colegas, que dividiram comigo as felicidades e tristezas do cotidiano.

RESUMO

Durante as últimas décadas os Algoritmos Genéticos (AGs) ganharam destaque por ser uma técnica poderosa para resolver problemas de busca e otimização, e consequentemente, o ensino desta técnica está se tornando cada vez mais comum. As aulas práticas com o uso de ferramentas didáticas são fundamentais para que os alunos consolidem e apliquem os conceitos aprendidos nas aulas teóricas. Este trabalho apresenta o GeneticsA, que é um software educacional de apoio ao ensino de AG, que tem como objetivo enriquecer o aprendizado, fazendo que o aluno consiga compreender como ocorreu a fases de avaliação, seleção, *crossover* e mutação. O GeneticsA permite que o usuário escolha as funções de avaliação (Simples, Normalização linear, Windowing e Escalonamento Sigma), os métodos de seleção (Roleta viciada, Método do torneio e Amostragem Estocástica), os operadores de crossover (Um ponto, Dois pontos e o operador Uniforme) e o problema de maximização de funções. O usuário também pode configurar parâmetros como tamanho da população, taxa de mutação, quantidade máxima de gerações e o número de genes. O software foi desenvolvido na linguagem de programação Java e para a criação da interface foi utilizada a API JavaFX.

Palavras chave: Algoritmo Genético. Inteligência Artificial. Software Educacional.

ABSTRACT

During the last decades Genetic Algorithms (GAs) have gained prominence as a powerful technique to solve search and optimization problems, and as a result, the teaching of this technique is becoming more and more common. The practical classes with the use of didactic tools are fundamental for students to consolidate and apply the concepts learned in theoretical classes. This work presents GeneticsA, which is an educational software to support the teaching of GA, which aims to enrich learning by enabling the student to understand how the evaluation, selection, crossover and mutation phases occurred. GeneticsA allows the user to choose the evaluation functions (Simple, Linear Normalization, Windowing and Sigma Scheduling), selection methods (Roulette Addicted, Tournament Method and Stochastic Sampling), crossover operators the Uniform operator) and the function maximization problem. The user can also configure parameters such as population size, mutation rate, maximum number of generations and number of genes. The software was developed in the Java programming language and JavaFX API was used to create the interface.

Keywords: Artificial Intelligence. Educational Software. Genetic Algorithm.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTA DE TABELA

LISTA DE SIGLAS

AE - Algoritmo Evolucionário

AG - Algoritmo Genético

API - Application Programming Interface

CAI - Computer Aided Instruction

CSS - Cascading Style Sheets

EE - Estratégia Evolucionária

EGALT - Educational Genetic Algorithms Learning Tool

GAV - Genetic Algorithm Viewer

GUI - Graphical User Interface

IA - Inteligência Artificial

ICAI - Intelligent Computer Aided Instruction

JDK - Java Development Kit

MVC - Model View Controller

NP - Non-Deterministic Polynomial time

PE - Programação Evolutiva

PG - Programação Genética

RF - Requisitos Funcionais

RNF - Requisitos Não Funcionais

XML - Extensible Markup Language

SUMÁRIO

1. INTRODUÇÃO

A Inteligência Artificial (IA) é um das áreas mais recentes em ciência e tecnologia, abrangendo uma enorme variedade de subcampos, do geral (aprendizagem e percepção) até tarefas específicas, como jogos de xadrez e demonstração de teoremas matemáticos (RUSSELL; NORVIG, 2013).

Os Algoritmos Evolucionários são um campo da IA - que usa modelos computacionais dos processos naturais de evolução como uma ferramenta para resolver problemas. Existe uma grande variedade de modelos computacionais baseados nesta estratégia, o que eles têm em comum é o conceito de simulação da evolução das espécies através de seleção, mutação e reprodução (LINDEN, 2008).

Conforme Barcellos (2010, p.14) “os Algoritmos Genéticos (AG), juntamente com Estratégias Evolucionárias (EE) e Programação Evolutiva (PE), formam uma classe de algoritmos de pesquisa baseados em evolução natural”.

Segundo Parreiras (2006) os AG são inspirados na teoria da evolução natural e pela genética. De acordo com Soares (1997) na natureza existe um processo de seleção dos seres vivos, os indivíduos mais preparados para a competição dominam os mais fracos e sobrevivem. Através da herança as características boas provavelmente passarão aos seus descendentes, tendo a mutação como principal aliado, para evitar que a maioria dos membros tenha o mesmo material genético.

Linden (2008) destaca que os AG realizam buscas direcionadas e inteligentes, sendo muito bons para resolver problemas de busca com espaços de busca intratavelmente grande, que não podem ser resolvidas por técnicas tradicionais. Devido a estas características marcantes, os AG são consideradas técnicas importantes, tanto que, são ensinadas em muitos cursos de Computação.

Muitos pesquisadores e estudantes tem dificuldades em programação, tornando assim, a implementação de um AG um grande desafio. O desenvolvimento de um software didático voltado para o ensino de AG ajudará o discente a direcionar seus estudos na compreensão das fases de um AG, e não com detalhes de programação.

Atualmente existem ferramentas de ensino para as mais diversas áreas, visto que auxiliam no aprendizado. No entanto, não é o que se percebe no ensino de AG, pois existe uma escassez de ferramentas que demonstrem a técnica sem a necessidade de implementação. O presente trabalho propõe o desenvolvimento de um software para auxiliar no processo de ensino-aprendizagem de Algoritmos Genéticos. Segundo Juca (2006, p.2) “as novas

tecnologias mostram que, quando utilizadas adequadamente, auxiliam no processo da construção do conhecimento, tornando o processo de ensino-aprendizagem mais estimulante e eficaz".

1.1. PROBLEMATIZAÇÃO

Os AGs são técnicas probabilísticas, e não técnicas determinísticas, ou seja, um AG com a mesma população inicial e o mesmo conjunto de parâmetros pode encontrar soluções diferentes cada vez que é executado. Os AGs não são processos puramente estocásticos, mas sim uma busca direcionada no espaço de soluções. (LINDEN, 2012). Devido ao fato de não ser uma técnica determinística, o aluno tem dificuldade em compreender como aconteceu todos os ciclos de um AG até chegar à solução obtida.

Os AGs entram em cena para resolver problemas NP-completos[1] ou incapazes de obter solução. A aplicabilidade de AGs é praticamente infinita, sempre que existir uma necessidade de busca ou otimização, um AG pode ser considerado como uma ferramenta de solução (LINDEN, 2012). Devido a essas características marcantes, os AGs vêm sendo estudados e utilizados por pesquisadores dos mais diversos campos de atuação. Mesmo com a existência de frameworks para implementação de AGs, o nível de conhecimento técnico para efetivo uso ainda é um problema, pois existem muitos pesquisadores não diretamente relacionadas à computação e que precisam entender a técnica para depois aplica-la em suas áreas.

Segundo Lemos e Fernandes (2014) dentro da grande quantidade de conteúdos que existem na disciplina de IA, o professor tem um semestre para ensinar os conceitos aos alunos da disciplina e muitas vezes este processo acaba sendo meramente teórico.

Baseando-se neste cenário abordado, surge o questionamento:

- O que poderia ser feito para melhorar o desempenho e a motivação dos alunos no ensino de AGs?

1.1.1. Solução proposta

Atualmente existem poucas ferramentas didáticas para auxiliar no ensino de AG, percebe-se que há uma escassez deste tipo de software. O presente trabalho propõe o

[1] Um problema é dito NP-Completo se não se conhece algoritmo de ordem polinomial capaz de resolvê-lo.

desenvolvimento de um software didático com interface gráfica, onde o discente possa acompanhar e entender o funcionamento das fases de um AG.

A finalidade deste software é enriquecer o aprendizado de AGs, fazendo que o aluno consiga compreender como ocorreu a fases de avaliação, seleção, crossover, mutação e de controle da população.

1.1.2. Delimitação do escopo

A ferramenta didática proposta deve permitir ao usuário testar exemplos didáticos, mostrar como a população se desenvolve ao longo de cada evolução, escolher parâmetros como o tamanho da população, taxa de mutação, função de avaliação, métodos de seleção e quantidade máxima de gerações.

O foco do software é auxiliar o usuário na compreensão de cada fase do AG. O software vai permitir que o usuário visualize a população inicial, veja a avaliação de cada cromossomo dentro da população, consiga ver os pais que foram selecionados, mostre os filhos que foram gerados e as mutações que tenha ocorrido, e a seleção dos sobreviventes desta geração.

A ferramenta didática proposta é desktop e implementada na linguagem de programação Java. O projeto fez o uso da API JavaFX. O software proposto não tem como objetivo mostrar o funcionamento de Sistemas Híbridos, que são sistemas que acabam combinando AG com outras técnicas de IA.

1.1.3. Justificativa

A sociedade passa por profundas mudanças caracterizadas pela valorização da informação, onde os processos de aquisição do conhecimento assumem um papel de destaque (MERCADO, 2002). De acordo com Valente et al. (1999), com essa valorização do conhecimento, requer o repensar dos processos educacionais, principalmente aqueles que estão diretamente relacionados com a formação de profissionais e com os processos de aprendizagem.

Uma pesquisa realiza por Ferreira (2010) revelou que 93,94% dos docentes do IF Baiano – Campus Guanambi acreditam que a informática facilita a aprendizagem do aluno. Segundo Ferreira (2010, p.151) “torna-se essencial introduzir a informática na prática de

ensino como uma maneira de atrair benefícios para o desempenho pedagógico, garantindo sua eficácia".

Os computadores estão propiciando uma verdadeira revolução no processo de ensino, inúmeros programas estão sendo desenvolvidos para auxiliar no processo de ensino-aprendizagem (VALENTE, 2008). Os softwares didáticos estão sendo utilizados nas mais diversas áreas do conhecimento. No ensino de AG é difícil encontrar ferramentas didáticas, pois praticamente não existem sistemas com esta finalidade. O desenvolvimento do software didático de AG proposto nesse trabalho deve servir como uma ferramenta de ensino, tornando as aulas mais práticas e dinâmicas.

Os softwares didáticos são importantes instrumentos que podem ajudar o discente na construção do seu próprio conhecimento. Um software didático de AG pode se tornar indispensável no desenvolvimento de atividades pedagogicamente importantes, como por exemplo, mostrar como uma população se desenvolve ao longo de cada evolução em um AG, favorecendo assim explorações enriquecedoras no processo de ensino-aprendizagem.

1.2. OBJETIVOS

1.2.1. Objetivo Geral

Desenvolver um software de apoio ao ensino de Algoritmos Genéticos.

1.2.2. Objetivos Específicos

Os objetivos específicos do trabalho são:

a) Identificar os operadores genéticos, módulos de população, as funções de avaliação, métodos de seleção e exemplos didáticos que serão implementados no software;
b) Identificar possíveis desafios que serão implementados no software;
c) Modelar o software focado no ensino de AGs;
d) Implementar o software educativo;
e) Testar a eficiência do software no ensino de Algoritmos Genéticos com alunos de graduação.

1.3. METODOLOGIA

Esta seção apresenta a metodologia empregada para o desenvolvimento do trabalho.

1. Revisão Bibliográfica: uma pesquisa bibliográfica em livros, artigos científicos e monografias acadêmicas sobre AGs e softwares educativos.
2. Análise do funcionamento de um AG: esta fase teve como objetivo estudar todas as fases de um AG, que são: - avaliação, seleção, combinação, mutação e de controle da população, e estudar as variações de cada fase.
3. Especificação de requisitos e modelagem: nesta etapa foram levantados os requisitos funcionais e não funcionais do sistema, além da modelagem do software através de Diagrama de Classe, Diagrama de Caso de Uso, e Diagrama de Sequência, facilitando assim o desenvolvimento do software.
4. Desenvolvimento: nesta etapa foi utilizada a linguagem de programação Java e a API JavaFX.
5. Fase de testes: Esta fase do trabalho consistiu em aplicar testes com os discentes, a fim de obter a confirmação, ou não, da eficácia do software.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordados conceitos relacionados a Algoritmos Genéticos e softwares educacionais, os quais são fundamentais para a realização e clareza deste trabalho. No que diz respeito a Algoritmos Genéticos, serão abordados seus fundamentos, estrutura, características das fases, vantagens, e suas respectivas aplicações, necessárias para o conhecimento e posteriormente aplicação da técnica. No contexto dos softwares e o processo educacional são discutidos sobre as teorias de aprendizagem e aspectos importantes acerca dos softwares educativos.

2.1. SOFTWARES E O PROCESSO EDUCACIONAL

Segundo Valente et al. (1999) a aprendizagem pode ocorrer basicamente de duas maneiras: memorização ou construção de conhecimento. Na memorização a informação não é processada e, portanto, não está passível de ser aplicada em situações de resolução de problemas e desafios. Já o conhecimento construído está incorporado aos esquemas mentais que são colocados para funcionar diante de situações-problema ou desafios.

O software educacional pode ser uma importante ferramenta para ajudar o aluno na construção do seu conhecimento, proporcionando novas descobertas, desenvolvendo e enriquecendo o processo de aprendizagem.

De acordo com Cenci e Bonelli (2012) mais importante que o software em si, é o modo como este será utilizado. O docente tem um papel fundamental e indispensável na escolha do software educativo, fazendo uma escolha que esteja fundamentada em sua proposta pedagógica. A proposta de ensino não deve ser feita para inserir um software, pelo contrário, o software deve ser escolhido de acordo com a proposta pedagógica adotada.

2.1.1. Teorias de ensino e aprendizagem

Segundo Vieira (1999, p.2) “um software para ser educativo deve ser pensado segundo uma teoria sobre como o sujeito aprende, como ele se apropria e constrói seu conhecimento”. Diante desse pensamento tem-se como objetivo sintetizar algumas teorias de aprendizagem e explicar algumas de suas implicações no processo de ensino e aprendizagem.

Na visão de Valente (1999) o ensino tradicional é baseado na transmissão de conhecimento, fazendo que o aluno fique passivo, sem capacidade crítica, formando assim um

profissional obsoleto e com uma visão de mundo limitada. Freire (1967) defende uma educação libertadora, sendo um processo onde o educador convida os educandos a reconhecer e desvelar a realidade criticamente. Nesta concepção existe uma hierarquia horizontal, fazendo com que o professor e o aluno aprendam juntos com a intensa interação.

Já na teoria cognitivista enfatiza o processo de cognição, onde o indivíduo atribui significados à realidade em que se encontra. Nesta teoria existe uma preocupação com o processo de compreensão, transformação, armazenamento e uso da informação envolvido na cognição e procura regularidades nesse processo mental (OSTERMANN; CAVALCANTI, 2011).

Na teoria de Piaget (1970) o crescimento cognitivo acontece através da assimilação e da acomodação.

> Todo esquema de assimilação é construído e toda abordagem à realidade supõe um esquema de assimilação. Quando a mente assimila, ela incorpora a realidade a seus esquemas de ação, impondo-se ao meio. Muitas vezes, os esquemas de ação da pessoa não conseguem assimilar determinada situação. Neste caso, a mente desiste ou se modifica. Quando a mente se modifica, ocorre o que Piaget chama de acomodação. As acomodações levam à construção de novos esquemas de assimilação, promovendo, com isso, o desenvolvimento cognitivo (OSTERMANN; CAVALCANTI, 2011, p.33).

Segundo Piaget (1970) ensinar é provocar o desequilíbrio, desde que não seja tão grande a ponto de não permitir o equilíbrio. Piaget é um precursor da linha construtivista e têm influenciado educadores de todo o mundo.

Na visão de Vygotski (1991, p. 61) “o aprendizado é um aspecto necessário e universal do processo de desenvolvimento das funções psicológicas culturalmente organizadas e especificamente humanas”. Na perspectiva vygotskyana, objetivo da educação seria o desenvolvimento da consciência construída culturalmente.

2.1.2. Softwares educacionais

Valente et al. (1999) destaca que o computador pode ser um importante recurso para facilitar o processo de conhecimento. O desenvolvimento de softwares educacionais ganhou grande impulso, porque pode tornar o processo de ensino-aprendizagem mais estimulante ao aluno.

Segundo Juca (2006) o software educativo tem como objetivo favorecer os processos de ensino-aprendizagem, sendo desenvolvido especialmente para construir o conhecimento relativo a um conteúdo didático. Na visão de Giraffa (2009) qualquer software pode ser um

software educativo, desde que o professor contextualize no processo de ensino e aprendizagem utilizando uma metodologia adequada.

2.1.2.1. Tipos de softwares educacionais

Segundo Giraffa (2009) os softwares educacionais podem ser divididos em dois grandes grupos: os CAI (*Computer Aided Instruction*), fundamentado na teoria *behaviorista*/comportamentalista[2] e os ICAI (*Intelligent Computer Aided Instruction*).

> As diferenças mais profundas entre os ICAI e os CAI tradicionais estão nas formas com que se concebem os seus projetos. Os CAI induzem o aluno a uma resposta correta mediante uma série de estímulos cuidadosamente planejados. Por outro lado, os ICAI pretendem simular algumas das capacidades cognitivas do aluno e utilizar os resultados como base das decisões pedagógicas a serem tomadas (GIRAFFA, 2009, p. 22-23).

O grupo do ICA é formado por softwares das seguintes modalidades: tutoriais, jogos educacionais, programas de reforço ou exercício e de simulação (GIRAFFA, 2009).

Os tutoriais são um tipo de software que apresenta a informação de acordo com uma sequência pedagógica particular, ou o discente pode escolher o conteúdo que desejar. Para verificar se a informação foi ou não processada, são apresentados ao aprendiz situações-problema, em geral, o problema apresentado se resume em verificar se o discente memorizou a informação fornecida (VALENTE, 1999).

Já os softwares de programação permitem que o discente crie seu próprio protótipo de programas, sem que tenham que possuir conhecimentos avançados de programação. As características disponíveis no processo de programação ajudam o aprendiz a encontrar seus erros, e ao professor compreender o processo pelo qual o aprendiz construiu conceitos e estratégias envolvidas no programa. Quando o discente utiliza uma linguagem de programação como o Logo, a programação permite a realização do ciclo descrição - execução - reflexão - depuração – descrição (VIEIRA, 1999).

Segundo Valente (1999) no software de simulação um determinado fenômeno pode ser simulado no computador, bastando para isso que um modelo desse fenômeno seja implementado na máquina. Ao usuário da simulação, cabe a alteração de certos parâmetros e a observação do comportamento do fenômeno, de acordo com os valores atribuídos.

[2] Teoria da psicologia que tem como principal objeto de estudo o comportamento.

Nos softwares de exercícios e reforço o discente se restringe a virar a página de um livro eletrônico ou realizar exercícios, as atividades exigem apenas o fazer, o memorizar informação, não importando a compreensão do que se está fazendo (VIEIRA, 1999).

De acordo com Moratori (2003) os jogos educativos computadorizados são desenvolvidos para motivar e desafiar o discente, as estratégias de um jogo são integradas a fim de alcançar um objetivo educacional determinado. Quando um jogo é utilizado adequadamente, promove o interesse e a motivação que por sua vez, aumentam a atenção do aluno e criam a sensação de que aprender é divertido.

O software proposto neste trabalho se enquadra na categoria de simulação, visto que o discente vai poder escolher os parâmetros de entrada e o software apresenta a evolução do AG, possibilitando ao discente acompanhar e entender o funcionamento das fases de um AG.

2.1.2.2. A influência dos softwares educacionais no processo de ensino-aprendizagem

Os softwares educativos podem influenciar o desenvolvimento do raciocínio cognitivo, além de poder proporcionar um ambiente lúdico, sendo assim uma importante ferramenta para enriquecer o processo de ensino-aprendizagem. Um software educacional pode direcionar a novas descobertas e servir para amadurecer conceitos (KONRATH; FALKEMBACH; TAROUCO, 2005).

Juca (2006) destaca que quando um software educativo apresenta uma nova ideia, ou seja, um elemento desconhecido e externo ao discente, o software deve propiciar condições de praticar este novo elemento e compará-lo com situações já vivenciadas para que possa torná-lo um elemento conhecido e interno.

2.2. ALGORITMOS GENÉTICOS

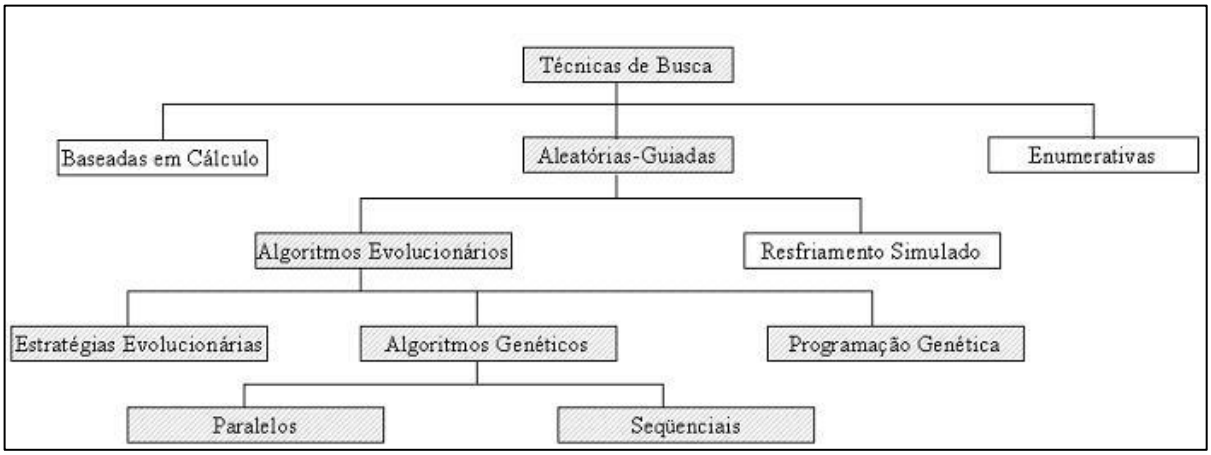
2.2.1. Algoritmos Evolucionários

Os Algoritmos Evolucionários (AE) são baseados nos princípios de seleção natural. Em seu famoso trabalho sobre a Origem das Espécies, Charles Darwin (1859) foi o primeiro a propor o conceito de evolução. É uma explicação para o desenvolvimento biológico de espécies, com seleção e sobrevivência do mais apto. Os seres vivos são o melhor exemplo para mostrar que a evolução natural é um processo de otimização bem-sucedido, que está em andamento há quatro bilhões de anos. Novas espécies podem surgir em semanas ou até dias

como bactérias, enquanto a evolução de outras espécies permanece estável por longos períodos (KRAMER, 2017).

Segundo Kita (2011) os AE são algoritmos de otimização meta-heurística baseados na população. Soluções candidatas para o problema de otimização são definidas como indivíduos em uma população, e a evolução da população leva a encontrar melhores soluções. Os AEs se enquadram como uma técnica de busca que é aleatória-guiada, como mostra a Figura 1.

Figura 1 - Diagrama que posiciona os algoritmos evolucionários como técnica de busca



Fonte: Linden (2012, p.46)

Machado (2005, p. 28) destaca que os “AE não necessitam de derivadas, como alguns métodos de otimização tradicionais, nem conhecimento acerca do espaço de busca, o que permite que eles sejam utilizados em quase todos os tipos de problemas de otimização”.

Estratégia Evolucionária (EE), Programação Genética (PG) e Algoritmo Genético (AG) são algoritmos evolutivos muito populares. A PG envolve a evolução de programas que resolvem problemas, os programas podem ser representados de várias maneiras, por exemplo, como árvores ou como programas *assembler*. Durante a avaliação da qualidade da solução, o programa é executado e seu desempenho é medido (KRAMER, 2017).

As EEs fazem parte de uma classe de algoritmos evolutivos, e são utilizados para resolver problemas de otimização de parâmetros (BACK; HOFFMEISTER; SCHWEFEL, 1991). Já os AGs que são o foco deste trabalho serão explicados na seção a seguir.

2.2.2. Algoritmos Genéticos

Embora as raízes da computação com inspiração evolucionária remontam aos primórdios da Ciência da Computação, os Algoritmos Genéticos foram inventados nos anos

60 por John Holland. Suas razões para estudar tais sistemas foram além do desejo de um melhor algoritmo de busca e otimização. Tais métodos são considerados abstrações úteis para o estudo da evolução em si, tanto em cenários naturais quanto artificiais (COLEY, 1999). Um AG permite que uma população composta por muitos indivíduos evolua sob regras de seleção especificadas para um estado que maximize a aptidão (*fitness*) dos indivíduos da população (HAUPT; HAUPT, 2004).

Neste ponto no trabalho, é útil introduzir formalmente algumas das terminologias biológicas que serão usadas ao longo do trabalho. No contexto dos AGs, os termos biológicos são usados no espírito da analogia com a biologia real, embora as entidades a que elas se referem sejam muito mais simples do que as biológicas reais. Confira no Quadro 1 as nomenclaturas utilizadas ao longo do trabalho.

Quadro 1 - Nomenclatura que distingue a área de AG da área da genética

Linguagem da ciência natural	**Algoritmos Genéticos**
Cromossomo	Indivíduo, *String*, Cromossomo, Árvore
Gene	Características
Alelo	Valor
Locus	Posição
Genótipo	Estrutura
Fenótipo	Conjunto de Parâmetros

Fonte: Linden (2012, p.51)

Devido ao fato de os AGs serem altamente inspirados na genética e na teoria da evolução, existe uma analogia muito forte entre os termos da biologia e termos usados no campo dos AGs. Mitchell (1996) explica de forma geral que na genética todos os organismos vivos consistem em células, e cada célula contém o mesmo conjunto de um ou mais cromossomos - cadeias de DNA - que servem como um "modelo" para o organismo. Um cromossomo pode ser conceitualmente dividido em genes - cada um dos quais codifica uma determinada proteína. De maneira geral, pode-se pensar em um gene como codificador de uma característica, como a cor dos olhos. Os diferentes “valores” possíveis para uma característica são chamados de alelos. Cada gene está localizado em um determinado *locus* (posição) no cromossomo.

Segundo Linden (2012) nos AGs as populações de indivíduos são criadas e submetidas aos operadores genéticos de: seleção, *crossover* e mutação. A avaliação é uma caracterização da qualidade de cada indivíduo e vão ajudar em um processo de evolução natural destes indivíduos. A seguir é mostrada uma visão de alto nível de um Algoritmo Genético de forma bem simplificada.

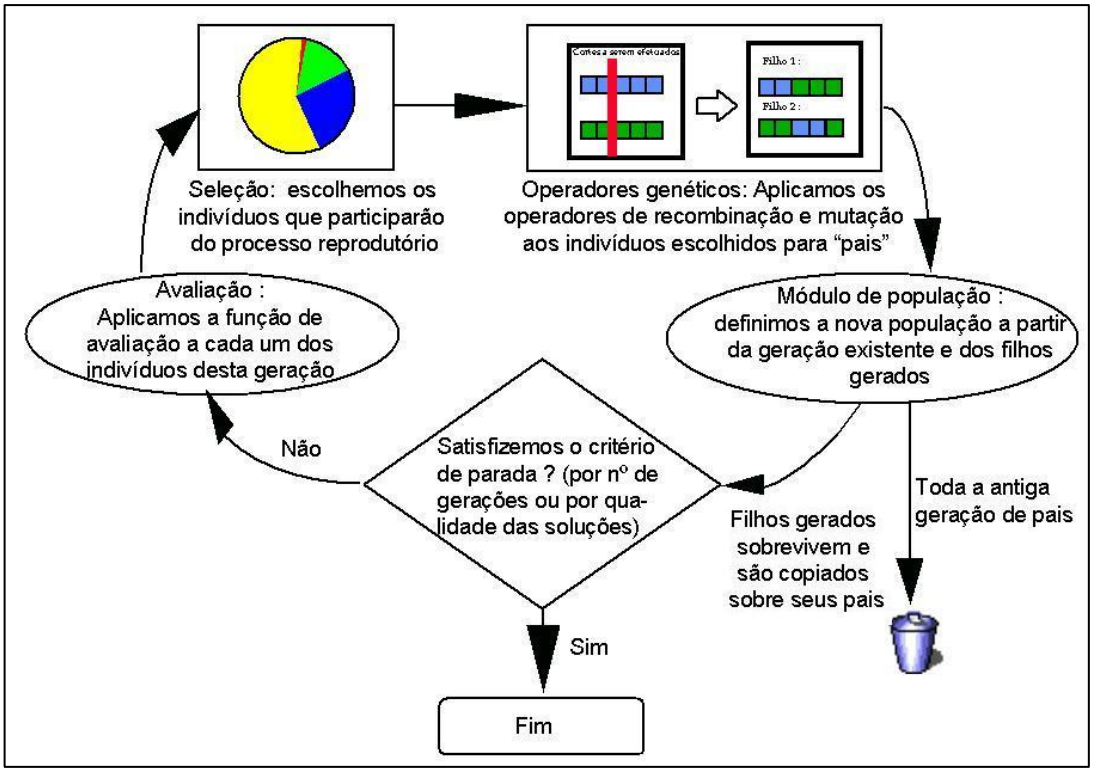
Quadro 2 - Algoritmo Genético simples

```
1:  Inicializar a população
2:  Avaliar cada cromossomo da população
3:  Repita
4:      Selecionar os pais para gerar novos cromossomos
5:      Aplicar o operador de crossover para gerar os indivíduos da nova geração
6:      Aplicar o operador de mutação
7:      Apagar os velhos membros da população
8:      Avaliar todos os novos cromossomos e insira-os na população
9:  até acabar o tempo OU cromossomo satisfaz os requerimentos e desempenho
```

Fonte: Adaptado de Linden (2012, p.63)

O Quadro 2 mostra o pseudocódigo de um Algoritmo Genético básico, que pode servir como modelo. No início, um conjunto de soluções, que é denominado população, é inicializado. Esta inicialização serve para cobrir aleatoriamente a população com indivíduos com características mais diversificadas possíveis. Depois as soluções são avaliadas, para que os indivíduos com os maiores *fitness* tenham mais chances de serem selecionados para a reprodução. O principal laço repetição do AG gera novas soluções candidatas, através dos operadores de seleção, *crossover*, mutação até que o tempo acabe ou encontre uma solução adequada. O esquema da Figura 2 apresenta o ciclo de forma mais detalhada.

Figura 2 - Esquema de um Algoritmo Genético



Fonte: Linden (2012, p.64)

O esquema da Figura 2 esconde a complexidade do processo de cada fase do AG. Nas próximas seções são discutidas mais detalhadamente cada uma destas fases.

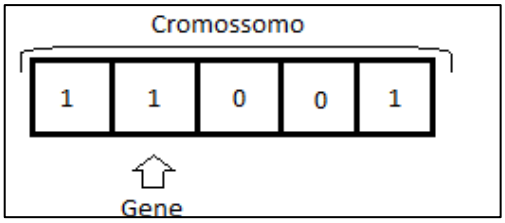
2.2.2.1. Representação cromossomial

A representação cromossomial consiste em traduzir a informação do problema em uma maneira que possa ser tratada computacionalmente. A representação cromossomial é completamente arbitrária, mas é interessante que algumas regras sejam seguidas, Linden (2012) destaca as regras gerais na representação:

a) A representação deve ser a mais simples possível.
b) Se houver soluções proibidas ao problema, então é preferível que elas não tenham uma representação.
c) Se o problema impuser condições de algum tipo, estas devem estar implícitas dentro da representação.

A representação mais usada pela comunidade da área de AGs é a representação binária, devido a sua simplicidade, onde um cromossomo nada mais é do que uma sequência de *bits* e cada gene é somente um *bit*, conforme é mostrado na Figura 3.

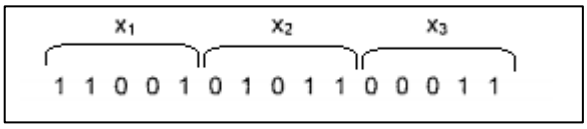
Figura 3 - Representação binária



Fonte: Acervo do autor (2018)

Em problemas com múltiplas variáveis, deve-se avaliar a ação conjunta do grupo de cromossomos, ou seja, o seu genótipo. No caso de um problema multimodal, cuja solução é um vetor $X = [x_1, x_2, x_3]$, cada x_i é uma variável e seu agrupamento é um indivíduo (SOARES, 1997). A Figura 4 ilustra a representação de 3 variáveis dentro de um cromossomo binário, onde cada variável usa 5 *bits* de representação.

Figura 4 - Exemplo da representação binária com 3 variáveis



Fonte: Adaptado de Soares (1997, p.37)

Dependendo do problema, podem ser necessários outros tipos de representação menos comuns, mas, igualmente interessantes e eventualmente mais poderosas. Linden (2012) destaca as representações de:

a) Números reais: (10,1; 4,5; 3,2; 5,6)
b) Permutações de elementos: (E4; E3; E1; E2; E5)
c) Listas de regras: (R2; R1; R5; R4; R3)
d) Representações híbridas: (1; 0; 0; 1; R2; R1; 10,5; 5,3; 8,9; 4,7)

A representação cromossomial não se limita a somente as representações apresentadas no trabalho, mas sim qualquer estrutura de dados que pudermos imaginar.

2.2.2.2. População inicial

Segundo Russell e Norvig (2013) os Algoritmos Genéticos começam com um conjunto de *k* estados gerados aleatoriamente. Linden (2012) destaca que a escolha da inicialização aleatória, de forma geral, gera uma boa distribuição das soluções no espaço de

busca, combinado com o uso do operador de mutação de forma eficaz garante uma boa exploração de todo o espaço de busca.

Outra forma de gerar a população inicial é dividindo o espaço de busca em *k* espaços igual e gerar *n/k* indivíduos em cada um destes espaços, e depois combinar com uma estratégia de otimização local, obtendo assim as melhores características de todo o espaço de busca em nossa população inicial (LINDEN, 2012).

2.2.2.3. Função de avaliação

Os AGs requerem uma função de avaliação ou *fitness* que atribui uma pontuação a cada cromossomo na população atual. A adequação de um cromossomo depende de quão bem esse cromossomo resolve o problema em questão (MITCHELL, 1996). A função de avaliação deve retornar valores mais altos para estados melhores, devido ao fato de a probabilidade de um indivíduo ser escolhido para reprodução é diretamente proporcional à sua pontuação *fitness* (RUSSELL; NORVIG, 2013).

Segundo Linden (2012) podemos entender a função de avaliação como uma nota dada ao indivíduo, sendo uma forma de diferenciar entre as boas e as más soluções para um problema. Ela deve embutir todas as restrições do problema, através de punições apropriadas para os cromossomos que as desrespeitarem.

Linden (2012) lembra que em alguns casos o desempenho do AG pode se degenerar. O primeiro problema é a questão do superindivíduo, que ocorre quando existe um ou mais membros com o fitness muito maior do que os outros indivíduos da população. O segundo problema ocorre quando há uma pequena diferença entre as avaliações, ou seja, todos os indivíduos têm funções de avaliação que diferem muito pouco percentualmente. As técnicas a seguir são utilizadas para resolver estes dois problemas.

2.2.2.3.1. Normalização linear

Segundo Linden (2012) a técnica consiste em colocar os cromossomos em ordem decrescente de valor e criam-se novas funções de avaliação, de forma que o melhor de todos receba um valor fixo (k) e os outros recebam valores iguais ao do cromossomo anterior menos um valor de decremento constante (t).

De acordo com Linden (2012) a técnica de normalização linear pode ser resumida com a seguinte fórmula recursiva:

$$aval_0 = k \quad (1)$$

$$aval_i = aval_{i-1} - t, \forall \ i = 1,2,..., n-1 \quad (2)$$

A técnica de normalização linear resolve o problema do superindivíduo. Veja a aplicação da técnica na Tabela 1, onde *n* é o tamanho da população, foi definido *k = n* e *t = 1*.

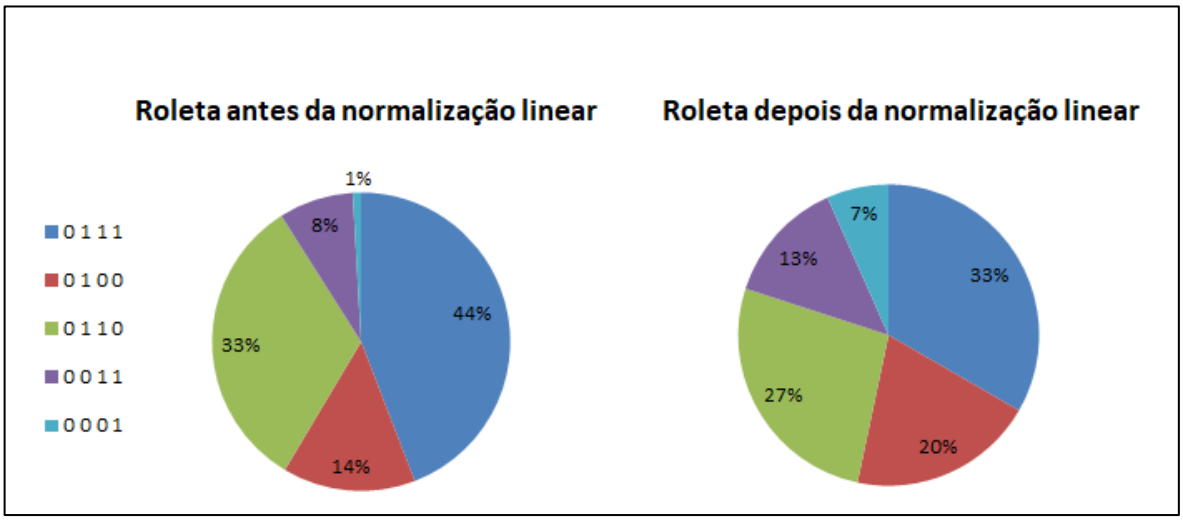
Tabela 1 - Exemplo da aplicação da técnica normalização linear

Indivíduo	Avaliação antes da Normalização	Avaliação depois da Normalização
0111	49	5
0100	16	3
0110	36	4
0011	9	2
0001	1	1

Fonte: Acervo do autor (2018)

A Figura 5 mostra que o indivíduo “0001” tinha cerca de 1% de chance antes da normalização, depois suas chances de seleção aumentaram para aproximadamente 7%.

Figura 5 – Roleta antes e depois da normalização linear



Fonte: Acervo do autor (2018)

Já o indivíduo “0111” tinha aproximadamente 44% de chance de seleção antes da normalização linear, e depois que foi aplicado a técnica suas chances caíram para cerca de 33%.

2.2.2.3.2. Windowing

Segundo Linden (2012) a técnica consiste em achar um valor mínimo dentre as funções de avaliação da população e diminua um valor arbitrário, de forma que o indivíduo de menor avaliação não passe a ter um fitness igual à zero. Depois designe para cada um dos cromossomos uma avaliação que seja igual à quantidade que excede o valor mínimo escolhido.

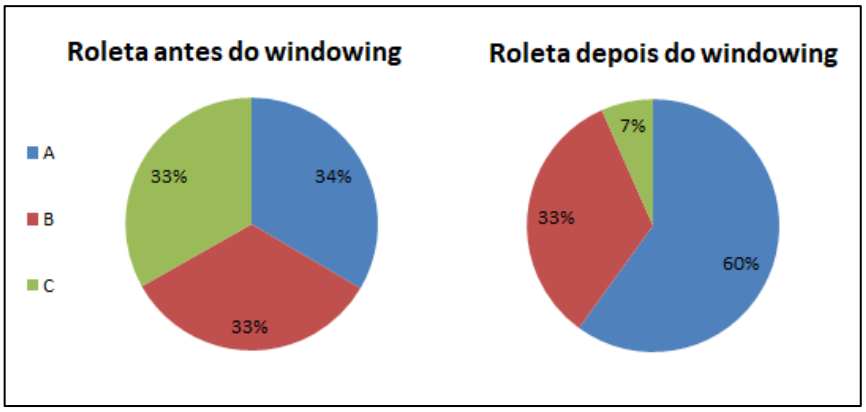
Tabela 2 - Exemplo da aplicação da técnica de windowing

Indivíduo	Fitness antes do Windowing	Fitness depois do Windowing
A	999	9
B	995	5
C	991	1

Fonte: Acervo do autor

No exemplo da Tabela 2, o indivíduo com menor fitness é “C” com 991, então foi subtraído o valor antecessor (neste caso 990) de todos os indivíduos da população. Desse modo, as chances de seleção do indivíduo “C” que é de aproximadamente 33% antes do windowing, caíram para aproximadamente 7% depois do windowing, conforme pode ser observado na Figura 6.

Figura 6 - Roleta antes e depois do windowing



Fonte: Acervo do autor

Já as chances de seleção do indivíduo “A” que era de aproximadamente 34% antes do windowing, subiram para cerca de 60% depois da aplicação do windowing. Linden (2012) destaca que esta técnica aumenta a pressão seletiva, portanto, deve-se haver consciência do tipo de problema que esta enfrentando antes de optar pela aplicação da técnica.

2.2.2.3.3. Escalonamento Sigma

De acordo com Mitchell (1996) a técnica Escalonamento Sigma consiste em modificar o fitness por uma fórmula, que considere: a avaliação de cada indivíduo, avaliação média populacional e o desvio padrão da população. Mitchell (1996) propôs a seguinte fórmula:

$$ExpVal\ (i,t) = \begin{cases} 1 + \dfrac{f(i) - \bar{f}(t)}{2\sigma(t)}, & \sigma \neq 0 \\ 1, & \sigma = 0 \end{cases} \quad (3)$$

Onde:

- *ExpVal (i, t)*: é o valor esperado do indivíduo *i* no tempo *t*.
- *f (i)*: é o fitness de *i*.
- *f(t)*: é o fitness médio da população no tempo *t*.
- *σ(t)*: o desvio padrão do fitness da população no tempo *t*.

No caso de ExpVal (i, t) for menor que 0, Mitchell (1996) e Linden (2012) destacam que pode ser atribuído um valor arbitrário baixo (por exemplo, 0,1), de modo que todos indivíduos tenham pelo menos alguma pequena chance de se reproduzir.

No início o desvio padrão é tipicamente alto, os indivíduos mais aptos não terão muitos desvios-padrão acima da média e, portanto, não receberão a maior parte da descendência. Da mesma forma, mais tarde, quando a população é tipicamente mais convergente e o desvio padrão é tipicamente menor, os indivíduos mais aptos se destacam mais, permitindo que a evolução continue (MITCHELL, 1996).

2.2.2.4. Seleção de pais

Este operador seleciona cromossomos na população para reprodução. Quanto mais apto o cromossomo, mais vezes é provável que ele seja selecionado para se reproduzir (MITCHELL, 1996). Linden (2012, p.75) destaca que “o método de seleção deve simular o

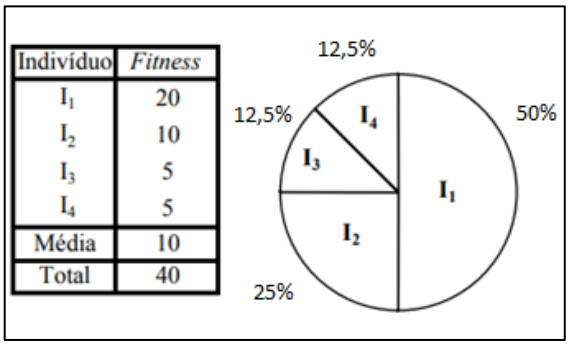
mecanismo de seleção natural que atua sobre as espécies biológicas, em que os pais mais capazes geram mais filhos, ao mesmo tempo em que permite que os pais menos aptos também gerem descendentes".

O método de seleção de pais pode influenciar bastante no resultado final, pois dependendo da técnica escolhida, podemos acelerar ou retardar a ocorrência da convergência genética (LINDEN, 2012).

2.2.2.4.1. Roleta Viciada

Existem diversas formas de seleção dos pais, entre elas destaca-se o método da Roleta Viciada – este método privilegia os indivíduos com *fitness* mais alto, sem desprezar os indivíduos com função de avaliação extremamente baixa (LINDEN, 2012). A Figura 7 ilustra uma população formada por 4 indivíduos, com seus respectivos *fitness*, e uma roleta com as chances de seleção de cada indivíduo.

Figura 7 - Representação do método de seleção Roleta Viciada



Fonte: Adaptado de Machado (2005, p.32)

No método da Roleta Viciada a probabilidade de um indivíduo ser escolhido para reprodução é diretamente proporcional à sua pontuação de fitness (RUSSELL; NORVIG, 2013).

2.2.2.4.2. Método de Torneio

Segundo Goldberg e Deb (1991) a ideia do método do torneio é escolher um certo número de indivíduos aleatoriamente de uma população, e selecionar o melhor indivíduo deste grupo. Torneios são frequentemente realizados entre pares de indivíduos (tamanho de torneio s = 2), embora torneios maiores possam ser realizados. O vencedor do torneio é o

indivíduo com o maior fitness no torneio. Observe na Figura 8 um exemplo da aplicação do método do torneio.

Figura 8 - Exemplo de aplicação do método do torneio

Indivíduo	Fitness
A1	40
A2	10
A3	20
A4	100
A5	60
A6	50

Torneios
A1 x A3
A2 x **A6**
A1 x **A4**
A4 x A6
A3 x **A4**
A2 x **A3**

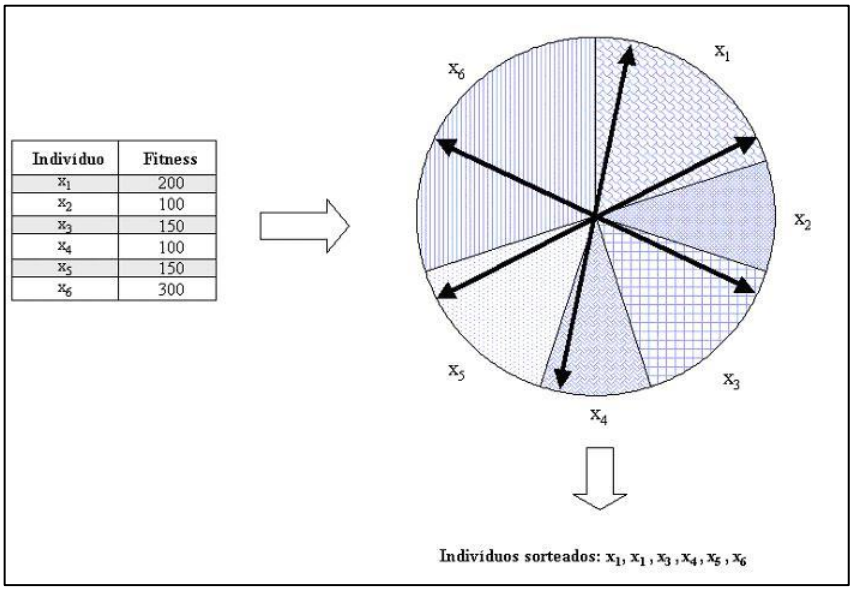
Fonte: Acervo do autor (2018)

O cromossomo com maior fitness da população, neste caso o A4, sempre que for sorteado para participar de um torneio será o vencedor.

2.2.2.4.3. Método de Amostragem Estocástica Uniforme

No Método de Amostragem Estocástica Uniforme para selecionarmos os *n* indivíduos que serão pais nesta geração, é escolhido um ponto de partida no círculo, em graus, entre 1 e 360°/n. Todos os indivíduos são separados um do outro por uma distância de 360°/n, de forma que o círculo seja completamente percorrido (LINDEN, 2012). Observe o exemplo da Figura 9.

Figura 9 - Exemplo do método de amostragem estocástica uniforme



Fonte: Linden (2012, p.209)

Depois que a lista de cromossomos está formada, ela deve ser “embaralhada” antes de selecionarmos os pais, senão podemos ter pais iguais no crossover, o que leva sempre a filhos iguais (LINDEN, 2012).

2.2.2.5. Operador Crossover

O operador de *crossover* simula o processo de reprodução sexuada como ocorre na natureza, onde os filhos são gerados a partir da combinação dos genes dos pais (PARREIRAS, 2006). Na natureza, a maioria das espécies tem dois pais. Algumas exceções não conhecem sexos diferentes e, portanto, possuem apenas um dos pais. Em Algoritmos Genéticos, podemos até estender os operadores de crossover para mais de dois pais (Kramer, 2017).

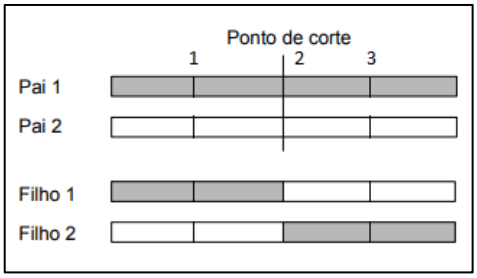
A motivação para tal operador é que ambas as sequências de caracteres podem representar partes bem-sucedidas de soluções que, quando combinadas, até superam seus pais (Kramer, 2017). Segundo Machado (2005) nem todos os cromossomos “filhos” gerados serão melhores que os “pais”, mas, devido à pressão de busca seletiva aplicada ao longo das gerações, a tendência é que os cromossomos que venham a ser gerados se tornem melhores.

2.2.2.5.1. Operador de um ponto

O operador de *crossover* de um ponto é o mais simples, nesse operador um único ponto de corte é escolhido aleatoriamente. De acordo com Linden (2012, p.84) “o ponto de corte constitui uma posição entre dois genes de um cromossomo. Cada indivíduo de *n* genes contém *n-1* pontos de corte, e este ponto de corte é o ponto de separação entre cada um dos genes que compõem o material genético de cada pai”.

Depois que o ponto de corte foi escolhido, as partes dos pais são combinadas para formar dois descendentes, conforme mostra a Figura 10.

Figura 10 - Exemplo de crossover de um ponto



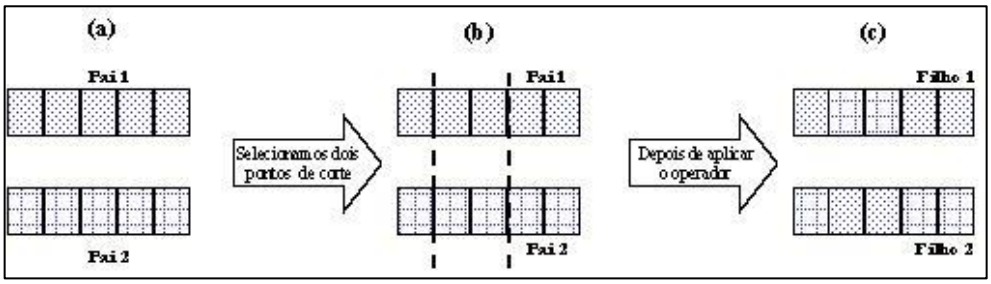
Fonte: Adaptado de Machado (2005, p.33)

A Figura 10 ilustra dois cromossomos “pais” compostos por 4 genes cada, o ponto de corte escolhido foi número 2. O primeiro filho é composto através da concatenação da parte do primeiro pai à esquerda do ponto de corte, com a parte do segundo pai à direita do ponto de corte. O segundo filho é composto pelas partes que sobraram.

2.2.2.5.2. Crossover de dois pontos

Neste operador, dois pontos de corte são selecionados aleatoriamente e a porção do cromossomo entre os dois cortes é trocada (COLEY, 1999). Veja o exemplo da Figura 11.

Figura 11 - Exemplo de crossover de dois pontos



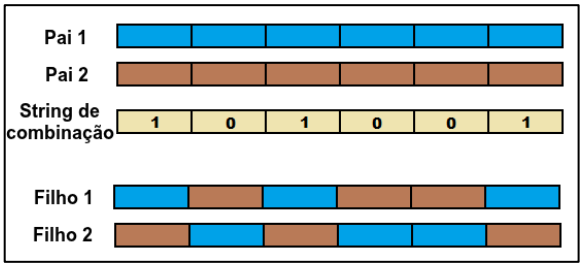
Fonte: Linden (2012)

Como pode ser visto na Figura 11, o primeiro filho é formado pelo material genético do primeiro pai que está fora dos pontos de corte mais o material genético do segundo pai entre os pontos de corte. O segundo filho é formado com os genes restantes.

2.2.2.5.3. Crossover uniforme

O crossover uniforme leva a ideia do crossover multiponto ao seu limite, sendo capaz de combinar qualquer esquema existente (COLEY, 1999).

Figura 12 - Exemplo de crossover uniforme



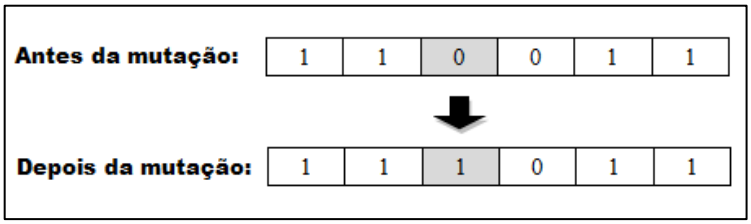
Fonte: Acervo do autor

De acordo com Linden (2012) esse operador sorteia um numero: “zero” ou “um” para cada gene. Se o valor sorteado for igual a “um”, o filho número um recebe o gene na posição corrente do primeiro pai e o segundo filho o gene corrente do segundo pai. Se o valor for zero, as atribuições são invertidas.

2.2.2.6. Operador de Mutação

Os operadores de mutação introduzem uma alteração aleatória no indivíduo. Para cada indivíduo recentemente criado, existe uma probabilidade de sofrer uma alteração, isto é, ele irá mudar suas características hereditárias de forma aleatória (MALAQUIAS, 2006). Na Figura 13 ocorreu uma mutação no terceiro gene de um cromossomo binário.

Figura 13- Exemplo de mutação em um cromossomo binário



Fonte: Acervo do autor (2018)

O operador de mutação tem como vantagem introduzir características que não estão na população original. No caso da representação binária, uma única mutação de um ponto altera um 1 para um 0 e vice-versa. Os pontos de mutação são selecionados aleatoriamente a partir do número total de bits do cromossomo. Uma taxa de mutação mais elevada aumenta a liberdade do algoritmo de pesquisar fora da região atual, mas também tende a dificultar que o algoritmo consiga convergir para uma solução (HAUPT; HAUPT, 2004).

De acordo com Linden (2012) não existe valor único de taxa de mutação para todos os casos, a taxa depende fortemente das características do problema que está sendo resolvido em cada instante. A maioria dos trabalhos na área dos AGs usa um valor predeterminado de 0,5% ou 1%.

2.2.3. Aplicações

Linden (2012) destaca que os AGs tem ganhado notoriedade, porque podem resolver uma gama infinita de problemas – sempre que houver uma necessidade de busca ou

otimização, um AG pode ser considerado como uma ferramenta de solução. Segundo Kramer (2017) os problemas de otimização podem ser encontrados nas mais diversas áreas, desde as ciências naturais até a matemática e a ciência da computação, da engenharia à vida social e cotidiana. Sempre que a tarefa é minimizar um erro, minimizar a energia, o peso, o desperdício, maximizar o lucro, enfrentamos problemas de otimização.

Frequentemente procuramos a solução ideal global, que é a melhor solução em todo o espaço da solução. Isso pode ser uma tarefa tediosa, já que o espaço da solução pode sofrer com restrições, ruído, instabilidade e um grande número de ótimos locais. Se modelados de maneira apropriada, os AGs são capazes de resolver a maioria dos problemas de otimização que ocorrem na prática (KRAMER, 2017).

O Quadro 3 apresenta algumas das tarefas e aplicações onde os AG já foram utilizados.

Quadro 3 - Aplicações dos Algoritmos Genéticos

Tarefas	**Aplicações**
Otimização	Agendamento de produção Alocação de Espaço Físico Estratégias de produção de combustíveis Funções Matemáticas Planejamento de tarefas Projeto de circuitos eletrônicos Tráfego Urbano
Busca	Classificação de Clientes Escalonamento de horários

Fonte: Linden (2012, p.387-429); Pacheco (1999, p.2)

Os AGs são muito utilizados para revolver problemas de busca e de otimização, porque são muito úteis para efetuar buscas em espaços intratavelmente grandes, que não podem ser resolvidos por técnicas tradicionais. Como já foi dito anteriormente, os AGs são baseados na evolução natural, onde se prevalece à sobrevivência dos mais aptos, devido a este fato, os AGs são uma importante ferramenta de otimização de soluções.

3. TRABALHOS CORRELATOS

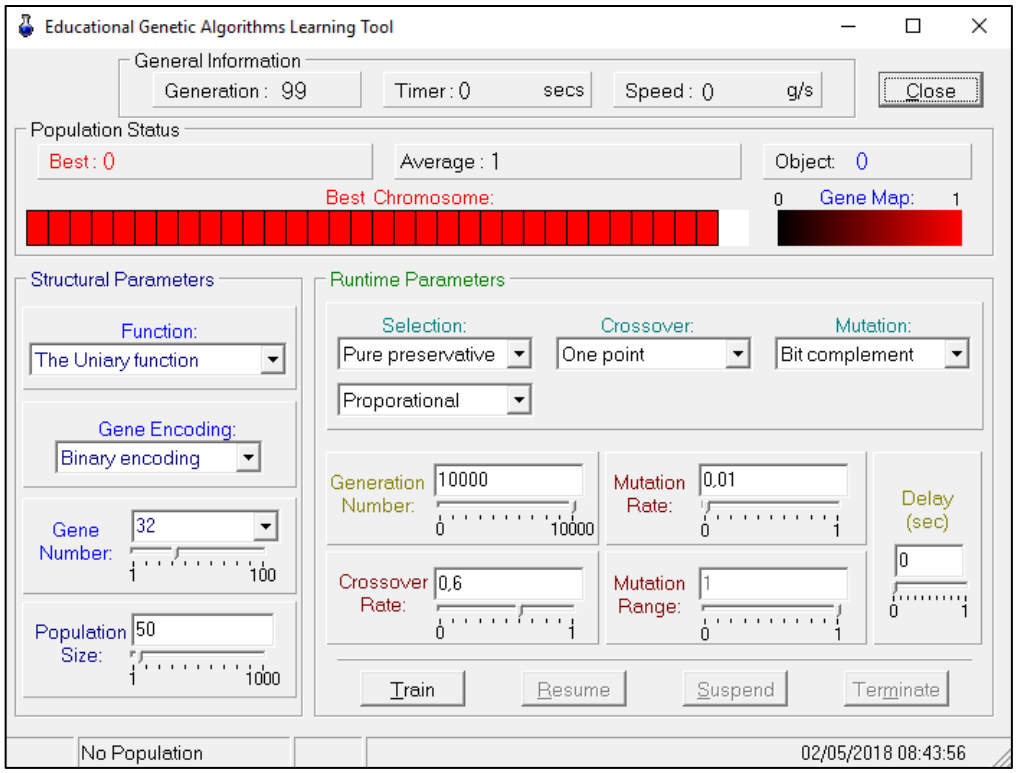
Nesta seção, são abordados trabalhos correlacionados ao objeto de estudo desta monografia. Desta forma, são apresentados diferentes softwares educacionais voltados ao ensino de Algoritmos Genéticos.

Atualmente existem poucos softwares educacionais na área de Algoritmos Genéticos, dos que são mostrados a seguir, somente o GraphEA possui um módulo de explicação. Os softwares apresentados nesta seção podem ser definidos como educacionais, porque se enquadram na definição de Valente (1999) como softwares de simulação.

3.1. EGALT

O *Educational Genetic Algorithms Learning Tool* (EGALT) foi desenvolvido por Ying-Hong Liao e Chuen-Tsai Sun, utilizando a linguagem de programação C++. O EGALT é uma ferramenta educativa, que tem como proposito ajudar os alunos na escolha dos parâmetros dos AGs e mostrar o resultado desta escolha. A tela principal do software é apresentada na Figura 14.

Figura 14 - Tela do EGALT



Fonte: Acervo do autor (2018)

Segundo Liao e Sun (2001) o usuário pode escolher parâmetros como:

- Escolha da função de avaliação.
- Operadores de seleção, *crossover* e mutação.
- Função de teste.
- Tamanho da população.
- Método de codificação.
- Quantidade de genes.
- Taxa de mutação e *crossover*.
- Faixa de mutação.

De acordo com Liao e Sun (2001) o EGALT mostra informações como:

- Quantidade de gerações.
- Tempo de execução (em segundos).
- Velocidade em (quantidade gerações/segundo).
- Quantidade de genes diferentes da solução desejada.
- Mapa genético do melhor cromossomo.
- Média da população.

Liao e Sun (2001) destacam que usando o EGALT, os estudantes podem concentrar-se apenas nos princípios do AG, podendo assim dar mais atenção em compreender o significado dos parâmetros do AG, e diminuir o tempo dedicado no ensino de métodos para resolver determinado problema.

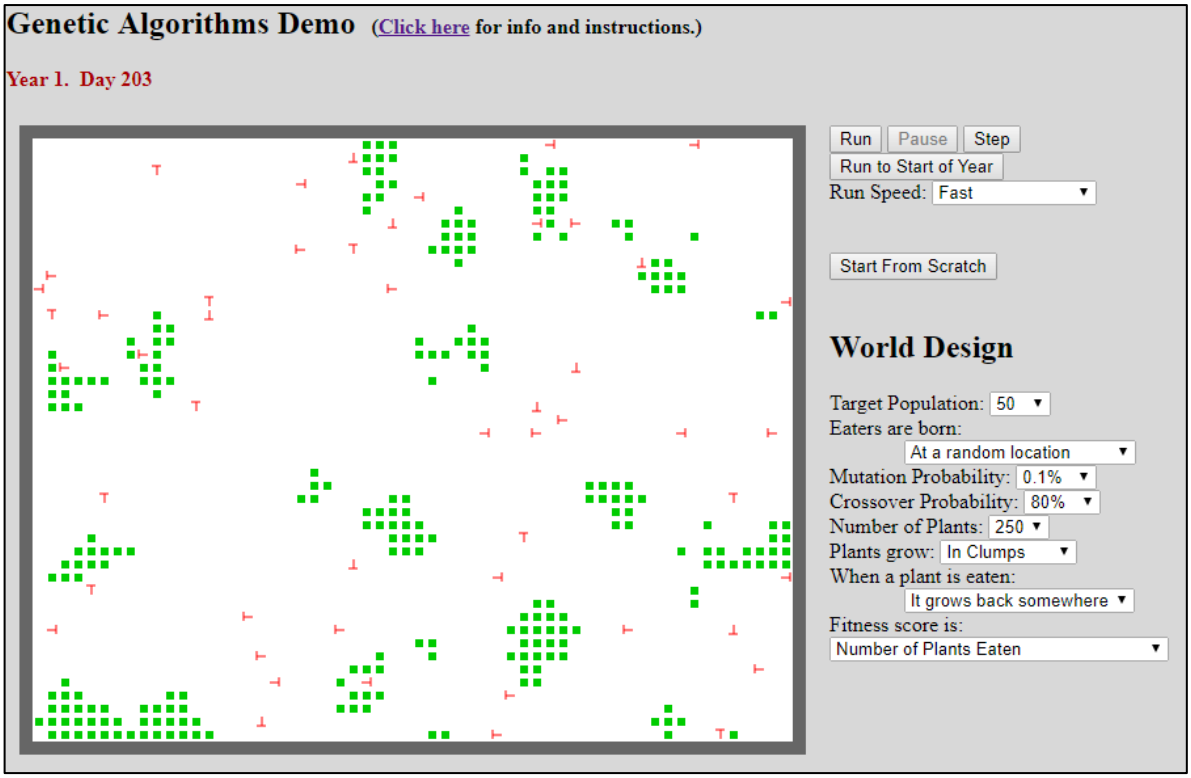
3.2. GENETIC ALGORITHMS DEMO

O Genetic Algorithms Demo foi desenvolvido em JavaScript e é um programa que faz uma analogia a evolução natural no mundo real. No software o usuário pode escolher parâmetros tradicionais dos AGs como tamanho da população, taxa de mutação e de crossover, e pontuação do fitness. Existem os parâmetros que são próprios do problema, que neste caso são o número de plantas, lugar que as plantas crescem, e o que acontece quando uma planta é comida.

Eck (2018) explica que no software existem os organismos chamados de “comedores” em um mundo simulado que contém “plantas” para os comedores comerem. As plantas são os pontos verdes e os comedores são as figuras vermelhas, conforme pode ser observado na Figura 15. As plantas tendem a crescer em aglomerados. Os comedores exibirão um

inicialmente comportamento não direcionado, de aparência aleatória. No final de cada ano, um novo mundo é produzido, contendo uma nova geração de comedores. Os novos comedores são produzidos como cópias, com modificação, de comedores da geração anterior. Quanto mais plantas um comedor come, maiores as chances de produzir "descendentes" na nova geração. A reprodução é baseada no fitness, que é determinado pelo número de plantas consumidas. O processo de mutação permite que novos comportamentos sejam testados e, se eles se mostrarem vantajosos, espalhar-se pela população nas gerações posteriores. Cada quadrado pode conter um comedor ou uma planta, ou pode estar em branco.

Figura 15 - Tela do Genetic Algorithms Demo



Fonte: Acervo do autor (2018)

O cromossomo do comedor é formado por regras, que diferem de um comedor para outro. O comportamento do comedor é completamente determinado por um conjunto de regras, que informam o que fazer para cada combinação possível de estado atual e o item que vê na frente. Tudo o que o comedor faz é seguir suas regras. (ECK, 2018).

De acordo com Eck (2018) conforme vão se passando as gerações, os comedores podem evoluir para se tornarem melhores consumidores. Depois de um tempo, o comportamento dos comedores muda e se adaptam no ambiente que foram inseridos e tornam-se mais eficientes, conseguindo assim consumir mais plantas.

3.3. GAV

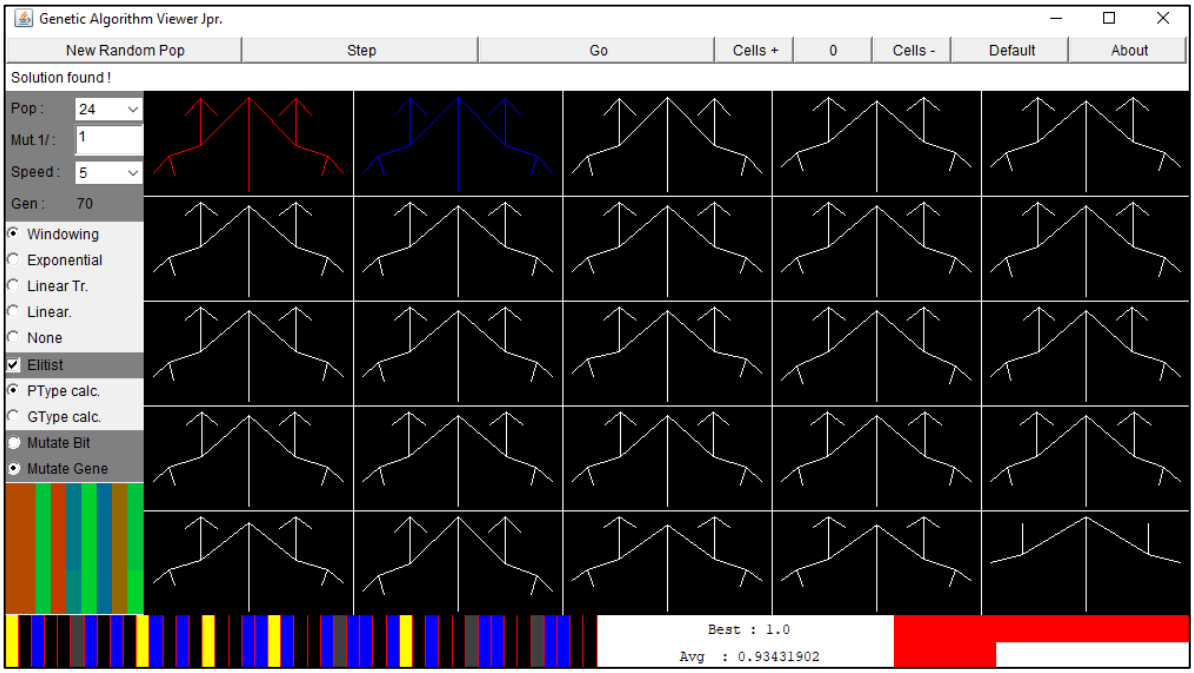
O *Genetic Algorithm Viewer* (GAV) foi desenvolvido por Jean Philippe Rennard. O GAV é uma aplicação de demonstração do funcionamento de um Algoritmo Genético. O software destina-se a mostrar a capacidade dos AGs, permitindo de certa forma uma visualização do processo de evolução até encontrar a solução.

Segundo Rennard (2000), essa ferramenta é composta pelas seguintes funções:

- Gerar uma população aleatória
- Escolha de parâmetros básicos como tamanho da população e taxa de mutação
- Escolha da função de avaliação (*Windowing*, Exponencial, Transformação Linear, Normalização Linear, nenhuma).
- Escolha do modo elitista
- Escolha do modo de mutação
- Visualização da evolução da população
- Visualização do genoma solução, do melhor e do pior genoma na população.
- Visualização do *fitness* do melhor indivíduo e o *fitness* médio.

A Figura 16 mostra que o software GAV encontrou a solução depois de 70 gerações, com uma população de 24 indivíduos, uma taxa de mutação de 1% e com a função de avaliação *Windowing*.

Figura 16 - Tela do GAV



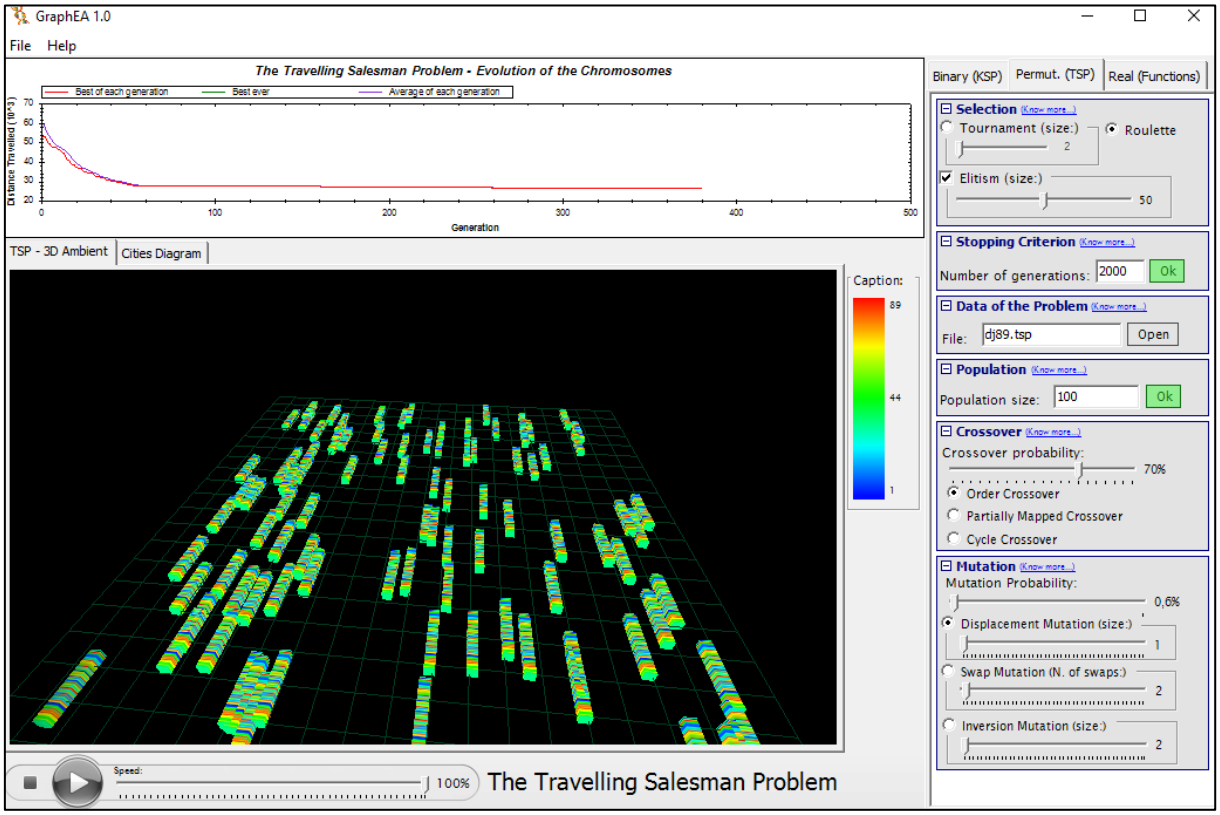
Fonte: Acervo do autor (2018)

O GAV permite visualizar o funcionamento de um AG e testar à influência de vários parâmetros, sendo assim possível observar a capacidade do algoritmo em encontrar a solução ou sua incapacidade de sair de um ótimo local (RENNARD, 2000).

3.4. GRAPHEA

O GraphEA é uma ferramenta que visa ajudar estudantes a aprender AGs, desenvolvida por Rafael Dinis, Anabela Simões e Jorge Bernardino. A ferramenta é exposta em um artigo científico, o qual expõe alguns detalhes sobre suas características e finalidades. Na Figura 17 é apresentado a tela inicial do software GraphEA.

Figura 17 - Tela inicial do GraphEA



Fonte: Acervo do autor (2018)

O GraphEA tem implementado três problemas de otimização bem conhecidos: o Problema da Mochila, o Problema do Caixeiro Viajante e o Problema de Otimização da Função.

De acordo com Dinis, Simões e Bernardino (2013) a ferramenta GraphEA oferece para cada AG implementado os seguintes recursos:

- O monitoramento contínuo da qualidade dos cromossomos: qualidade média da geração, melhor cromossomo da geração e melhor cromossomo de todos os tempos;
- Possibilidade de controlar a execução do algoritmo e parametrizar qualquer aspecto do algoritmo durante a execução: seleção, elitismo, tamanho da população, operador de *crossover* e respectiva probabilidade, operador de mutação e respectiva probabilidade, entre outras funcionalidades;
- Uma nova visualização 3D da população, permitindo ao usuário navegar por um ambiente 3D e selecionar um cromossomo arbitrário e visualizar seu conteúdo genético, bem como os efeitos da aplicação dos operadores de *crossover* e mutação sobre os genes.

Dinis, Simões e Bernardino (2013) destacam que o principal objetivo de sua ferramenta GraphEA é mostrar graficamente o poder dos AGs. Características como a parametrização do AG e a visualização 3D da constituição e formação de cromossomos com a intenção de proporcionar ao usuário uma experiência agradável, e não apenas para aprender os componentes dos algoritmos genéticos, mas também para permitir uma análise detalhada da eficácia dos operadores e outros mecanismos inerentes a esses algoritmos.

Os quatro softwares apresentados neste capítulo possuem características em comum, como a escolha de parâmetros do AG e a visualização dos resultados desta escolha. Percebe-se a ausência de características importantes, como a visualização de cada fase do AG e de uma explicação. O GeneticsA busca suprir estas ausências e será apresentado no capítulo a seguir.

4. DESENVOLVIMENTO

Neste capítulo, são abordadas as especificações, as técnicas e ferramentas utilizadas, além das telas do GeneticsA e os principais trechos de código do software.

4.1. ESPECIFICAÇÕES FORMAIS

Nesta seção são descritas as especificações formais do software educacional proposto. São descritos os Requisitos Funcionais (RF) e Requisitos não Funcionais (RNF) da aplicação, além das especificações do projeto, através de Diagramas de Caso de Uso, Diagramas de Classe, e Diagrama de Sequência.

4.1.1. Requisitos

Segundo Sommerville (2007, p.79) “os requisitos de um sistema são descrições dos serviços fornecidos pelo sistema e as suas restrições operacionais”. Os requisitos de sistema de software são, frequentemente, classificados em requisitos funcionais e não funcionais.

4.1.1.1. Requisitos Funcionais

Os requisitos funcionais de um sistema descrevem o que o sistema de fazer, como o sistema de reagir a entradas específicas e como o sistema deve se comportar em determinadas situações (SOMMERVILLE, 2007). O Quadro 4 mostra os RF do software proposto neste trabalho.

Quadro 4 - Requisitos funcionais

Requisitos funcionais
RF01: O software deve permitir o aluno escolher parâmetros genéticos como função de avaliação, método de seleção, operador de *crossover*, taxa de mutação e tamanho da população.
RF02: O software deve permitir o aluno escolher um exemplo didático preestabelecido.
RF03: O software deve permitir o aluno acompanhar o desenvolvimento da população.
RF04: O software deve permitir ao aluno a visualização do *fitness* dos cromossomos.
RF05: O software deve permitir a visualização dos cromossomos que foram selecionados.

RF06: O software deve permitir ao aluno a visualização do cruzamento do cromossomo.
RF07: O software deve permitir ao aluno a visualização do ponto onde ocorreu a mutação no cromossomo.

Fonte: Acervo do autor (2018)

4.1.1.2. Requisitos não funcionais

De acordo com Sommerville (2007) os requisitos não funcionais são aqueles não diretamente relacionados às funções específicas fornecidas pelo sistema. Os RNF aplicam-se frequentemente, ao sistema como um todo, eles não se aplicam às características ou serviços individuais do sistema. O Quadro 5 mostra os RNF do software proposto neste trabalho.

Quadro 5 - Requisitos não funcionais

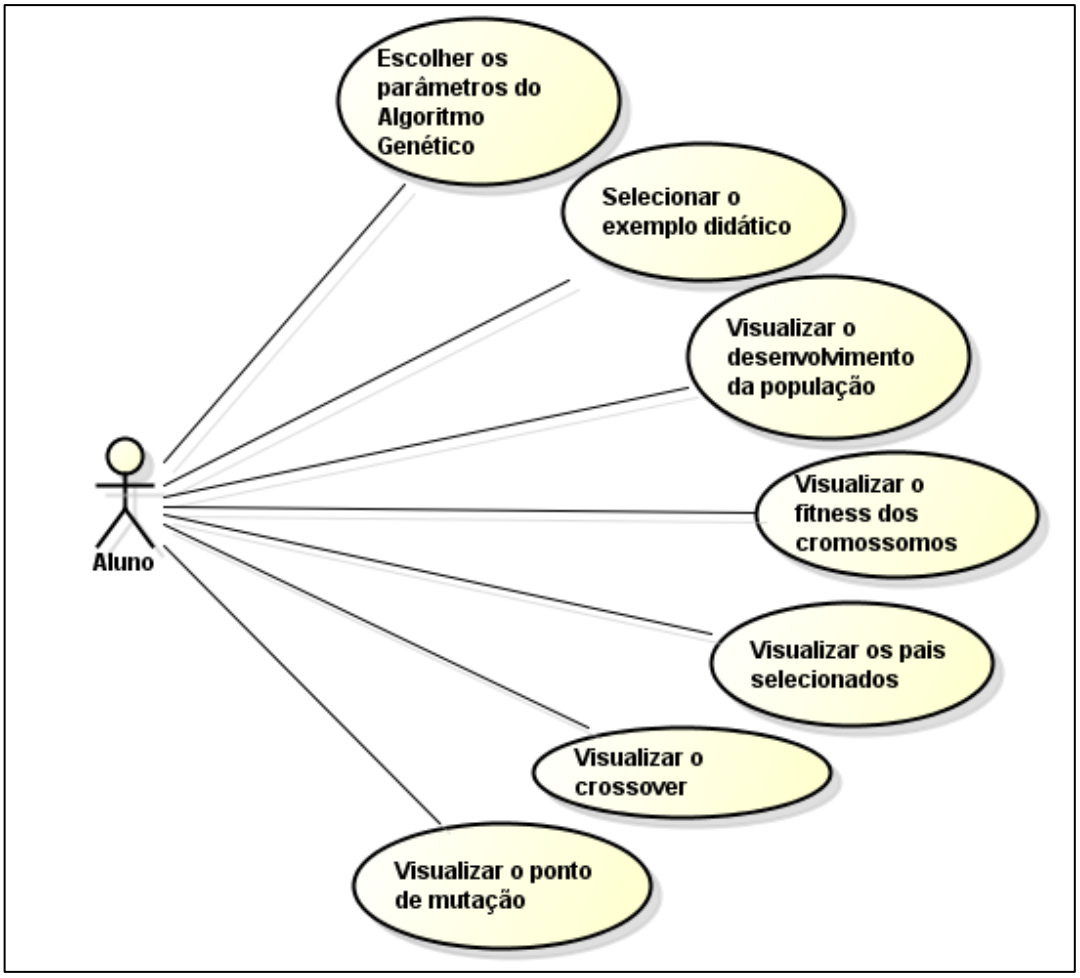
Requisitos não funcionais
RNF01: Utilizar a linguagem de programação Java para desenvolver o software educacional.
RNF02: Utilizar a API JavaFX na construção da interface.

Fonte: Acervo do autor (2018)

4.1.2. Diagrama de Casos de uso

De acordo com Booch, Rumbaugh e Jacobson (2012, p.263) “os diagramas de casos de uso são importantes para visualizar, especificar e documentar o comportamento de um elemento. Esses diagramas fazem com que sistemas, subsistemas e classes fiquem acessíveis e compreensíveis”. A Figura 18 ilustra o diagrama de caso de uso do software proposto neste trabalho.

Figura 18 - Diagrama de Caso de Uso

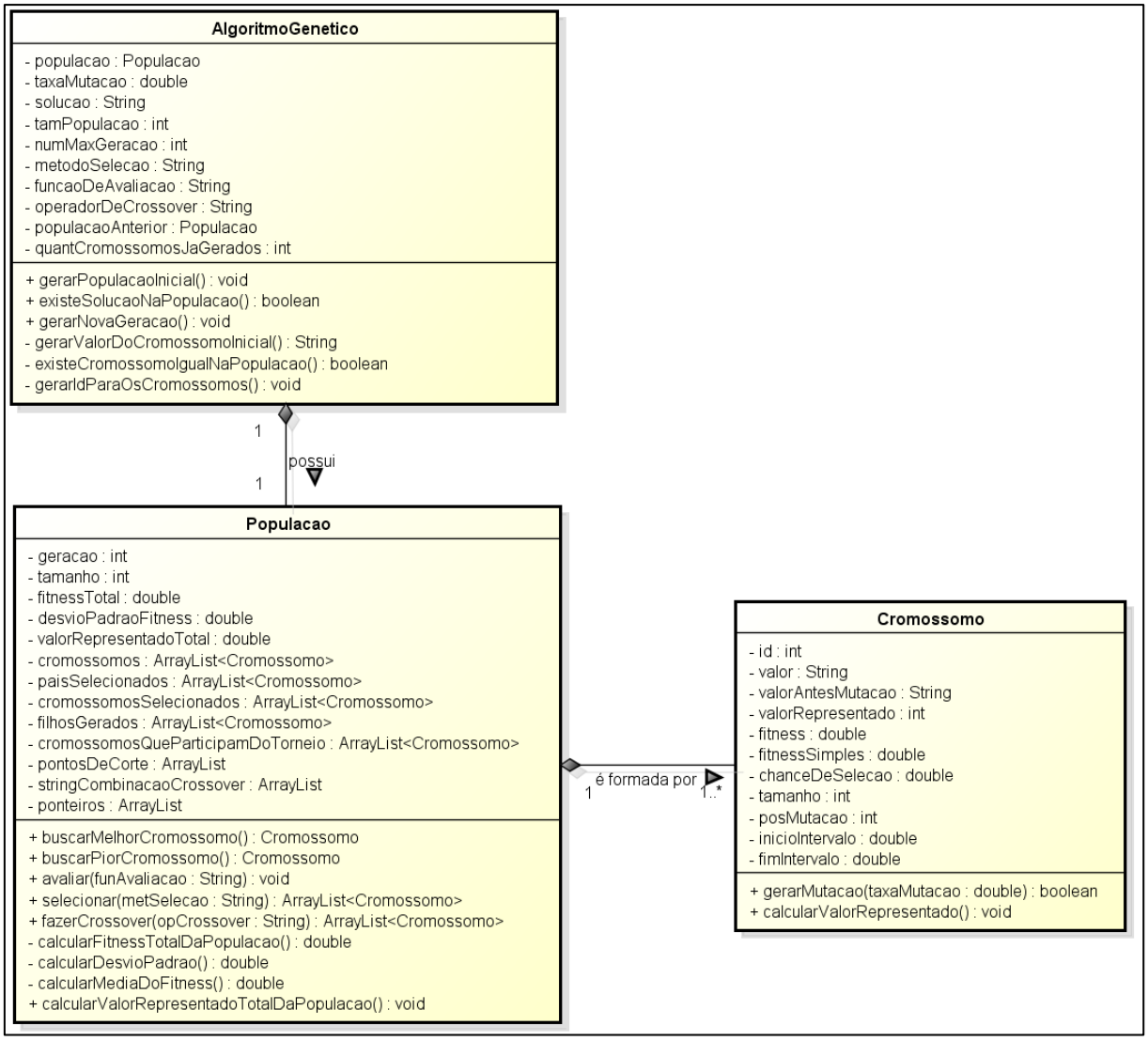


Fonte: Acervo do autor (2018)

4.1.3. Diagrama de Classes

Os diagramas de classes são utilizados para fazer a modelagem da visão estática de um sistema. O diagrama de classe mostra um conjunto de classes, interfaces e colaborações e seus relacionamentos (BOOCH; RUMBAUGH; JACOBSON, 2012). A Figura 19 mostra o diagrama de classes do software proposto neste projeto.

Figura 19 - Diagrama de Classes

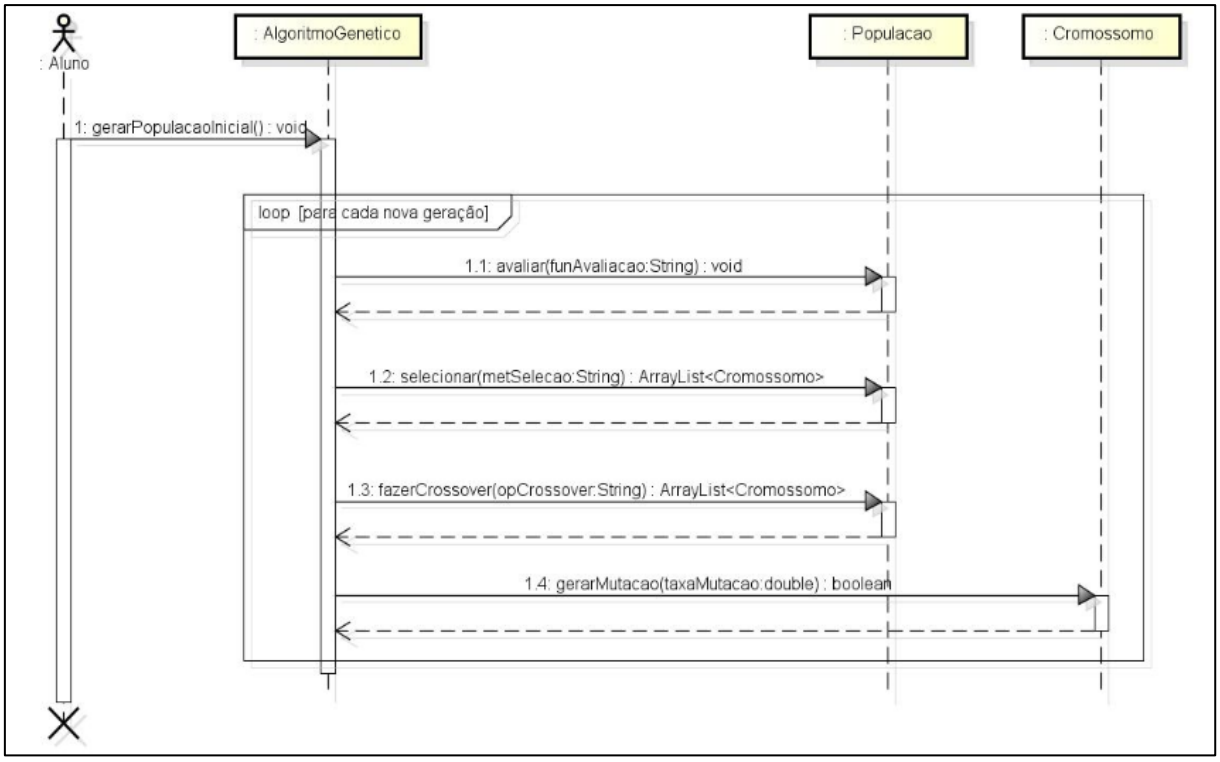


Fonte: Acervo do autor (2018)

4.1.4. Diagrama de sequência

O diagrama de sequência é utilizado para indicar as comunicações dinâmicas entre objetos durante a execução de uma tarefa, além de mostrar a ordem temporal na qual as mensagens são enviadas entre os objetos para executar aquela tarefa. O diagrama de sequência pode ser usado também para mostrar as interações em um caso de uso ou em um cenário de um sistema de software (PRESSMAN, 2011). A Figura 20 ilustra o diagrama de sequência.

Figura 20 - Diagrama de sequência



Fonte: Acervo do autor (2018)

O ator Aluno ao clicar no botão iniciar acaba chamando o método *gerarPopulacaoInicial* na classe *AlgoritmoGenetico*. Depois entra em um loop, onde são realizadas as chamadas dos métodos *avaliar*, *selecionar*, *fazerCrossover* e *gerarMutacao*. Quando o loop terminar o Algoritmo Genético chegou ao fim.

4.2. IMPLEMENTAÇÃO

Nesta seção será descrito como o software foi desenvolvido, suas principais funcionalidades, técnicas e ferramentas utilizadas no desenvolvimento, com o objetivo de detalhar a implementação do projeto.

4.2.1. Técnicas e ferramentas

O projeto foi desenvolvido na linguagem de programação Java (versão 1.8 do JDK) em conjunto com o JavaFX 8.0.20 e com a ferramenta de layout JavaFX Scene Builder 2.0. O ambiente de desenvolvimento utilizado foi o Netbeans 8.0.2 e o sistema operacional Windows 10.

4.2.2. JavaFX

Segundo Oracle (2018) o “JavaFX é uma tecnologia de software que, ao ser combinada com Java, permite a criação e implantação de aplicações de aparência moderna e conteúdo rico de áudio e vídeo”. A aparência dos aplicativos JavaFX pode ser personalizada com Cascading Style Sheets (CSS), que separa aparência da implementação para que os desenvolvedores possam se concentrar na codificação. Os designers gráficos podem personalizar facilmente a aparência e o estilo do aplicativo por meio do CSS (ORACLE, 2014).

O JavaFX permite que o desenvolvedor projete com o Model-View-Controller (MVC), através do uso de FXML e Java. O "Model" consiste em objetos de domínio específicos do aplicativo, a "View" consiste em FXML e o "Controller" é um código Java que define o comportamento da GUI ao interagir com o usuário (HOMMEL, 2014).

4.2.2.1. FXML e Scene Builder

O FXML é uma linguagem de marcação declarativa baseada em eXtensible Markup Language(XML) para a construção de uma interface de usuário do aplicativo JavaFX. O programador pode codificar em FXML ou usar o JavaFX Scene Builder para projetar interativamente a interface gráfica do usuário (ORACLE, 2014).

O Scene Builder é uma ferramenta da Oracle bastante intuitiva, que auxilia na criação e manutenção do FXML. De acordo com Oracle (2013) o Scene Builder permite projetar rapidamente as interfaces(GUI) do aplicativo JavaFX arrastando um componente de interface e soltando-o em uma área de exibição de conteúdo. O código FXML é gerado automaticamente em segundo plano.

4.2.3. Informações sobre o projeto

O nome dado ao software apresentado neste trabalho é GeneticsA, a palavra *genetics* significa genética em inglês e a letra “A” vem de Algoritmos. O GeneticsA conta com um assistente virtual chamado de “Darwinho”, em homenagem a Charles Darwin, considerado o pai da teoria da evolução. Darwinho apresenta as informações sobre cada etapa do AG ao discente, auxiliando assim o processo de ensino-aprendizagem.

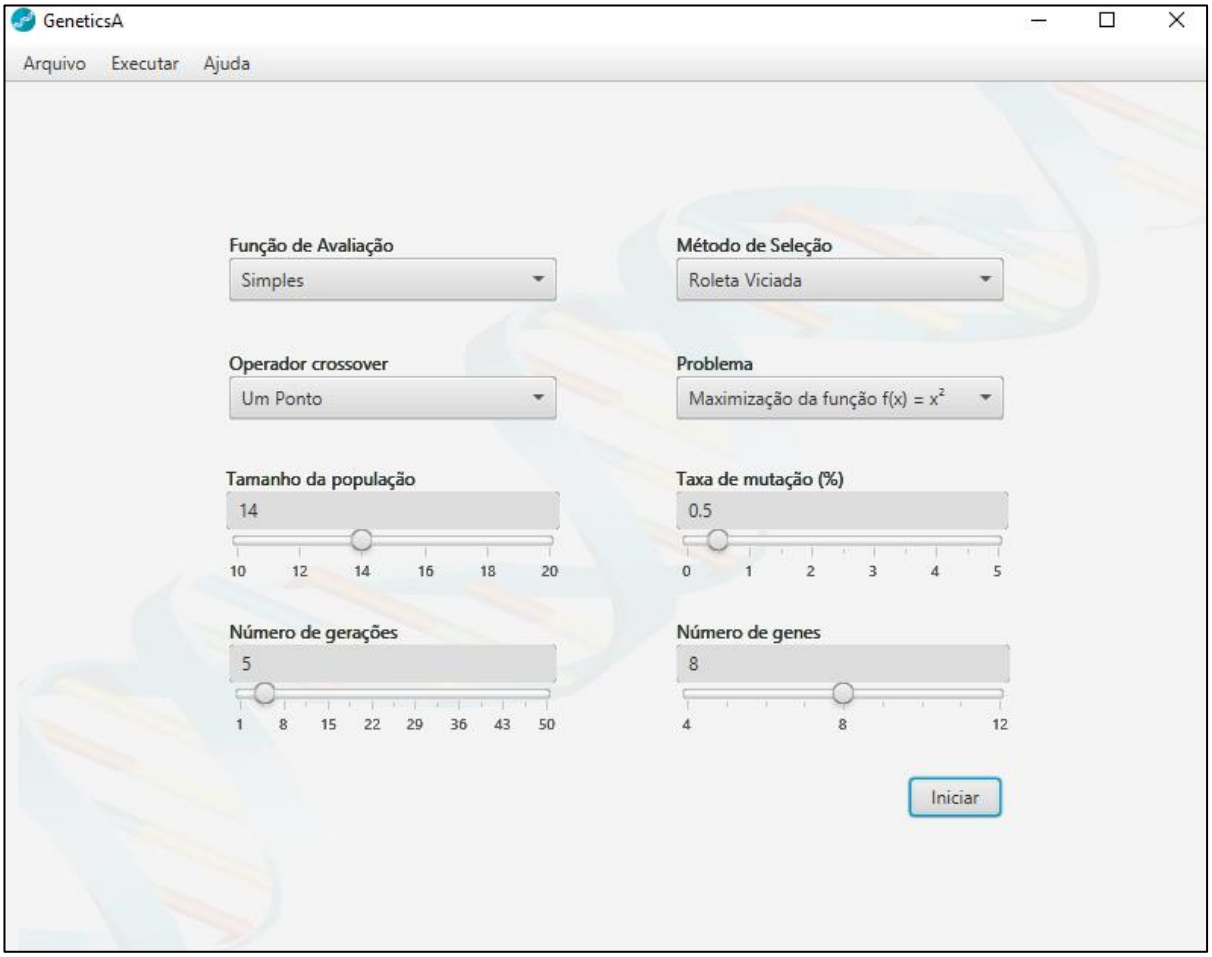
4.2.4. Interfaces do GeneticsA

Nesta subseção são apresentadas as principais telas do GeneticsA.

4.2.4.1. Tela inicial

O primeiro passo para começar a usar o GeneticsA é escolher os parâmetros do Algoritmo Genético. Na tela inicial podem ser escolhidos parâmetros como função de avaliação, método de seleção, operador de crossover, tamanho da população, taxa de mutação, número de gerações e número de genes. Podemos ver a tela inicial na Figura 21.

Figura 21 - Tela inicial do GeneticsA



Fonte: Acervo do autor (2018)

As escolhas disponíveis em cada parâmetro são:

- **Função de avaliação**: Simples; Normalização linear; Windowing; Escalonamento sigma.
- **Método de seleção**: Roleta Viciada; Método do torneio; Amostragem estocástica.
- **Operador crossover**: Um ponto; Dois pontos; Uniforme.
- **Problema**: Maximização da função $f(x) = x^2$; Maximização da função $f(x) = -x + 4096$.
- **Tamanho da população**: $A = \{10, 12, 14, 16, 18, 20\}$.
- **Taxa de mutação**: $B = \{x \in \mathbb{R}: 0 \leqslant x \leqslant 5\}$.
- **Número de gerações**: $C = \{x \in \mathbb{N}: 1 \leqslant x \leqslant 50\}$.
- **Número de genes**: $D = \{x \in \mathbb{N}: 4 \leqslant x \leqslant 12\}$.

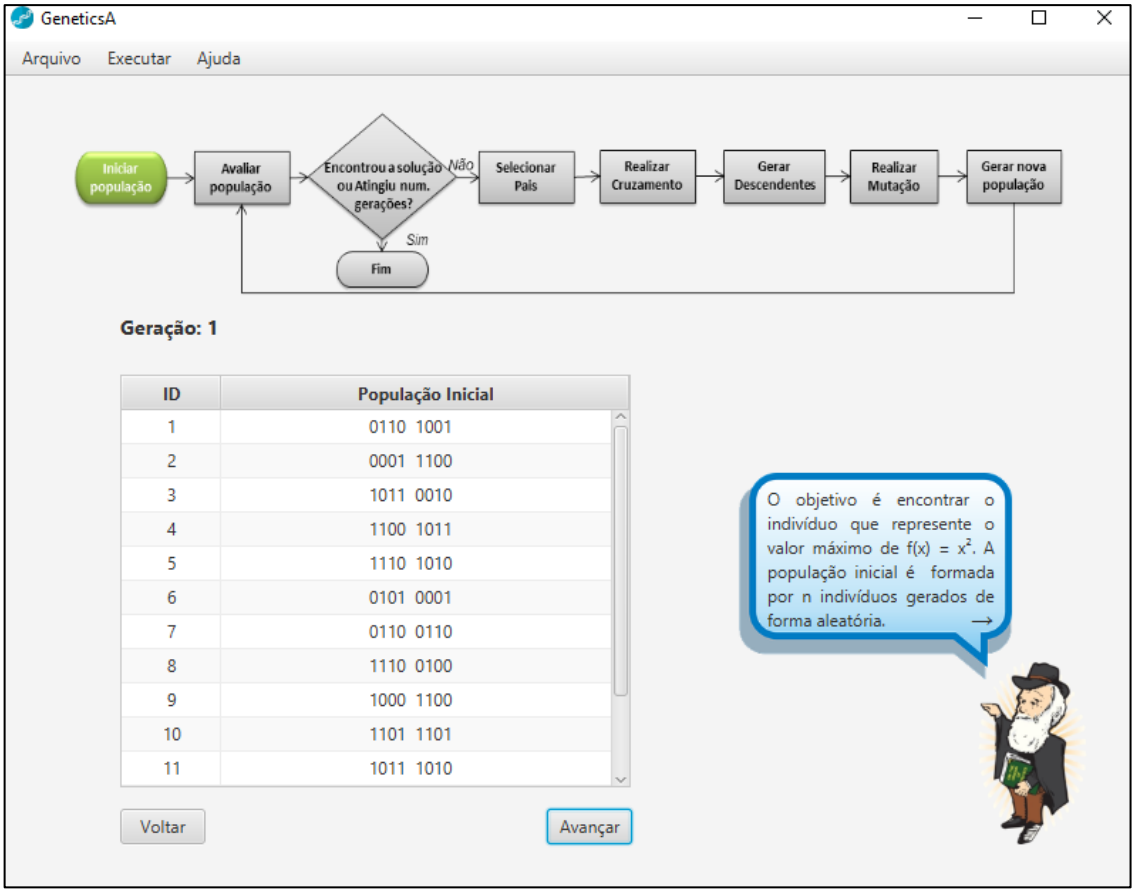
Na maioria das telas existe um menu que está posicionado no topo da janela e é formado por três itens: arquivo, executar e ajuda, conforme pode ser observado na Figura 21. O item “arquivo” permite ao usuário “reiniciar” e “sair” do software. O item “executar” é formado por um único subitem chamado de “Executar até o final automaticamente”, que realiza a execução até encontrar a solução ou atingir o limite de gerações, sem precisar acompanhar o desenvolvimento da população. Já o item “ajuda” é composto por informações sobre as funções de avaliação, métodos de seleção, operadores de crossover e informações sobre o projeto.

4.2.4.2. População inicial

A interface padrão é composta por um fluxograma localizado na parte superior da tela, o assistente Darwinho no lado direito, e as informações da população no lado esquerdo, conforme pode ser observado na Figura 22. O fluxograma serve para mostrar a fase atual do Algoritmo Genético, além de dar uma visão geral do seu funcionamento.

A tela população inicial permite visualizar os cromossomos que foram gerados inicialmente de forma aleatória. Cada cromossomo tem um número de identificação único chamado de “ID”, que é usado para diferenciar cada indivíduo na população.

Figura 22 - Tela população inicial



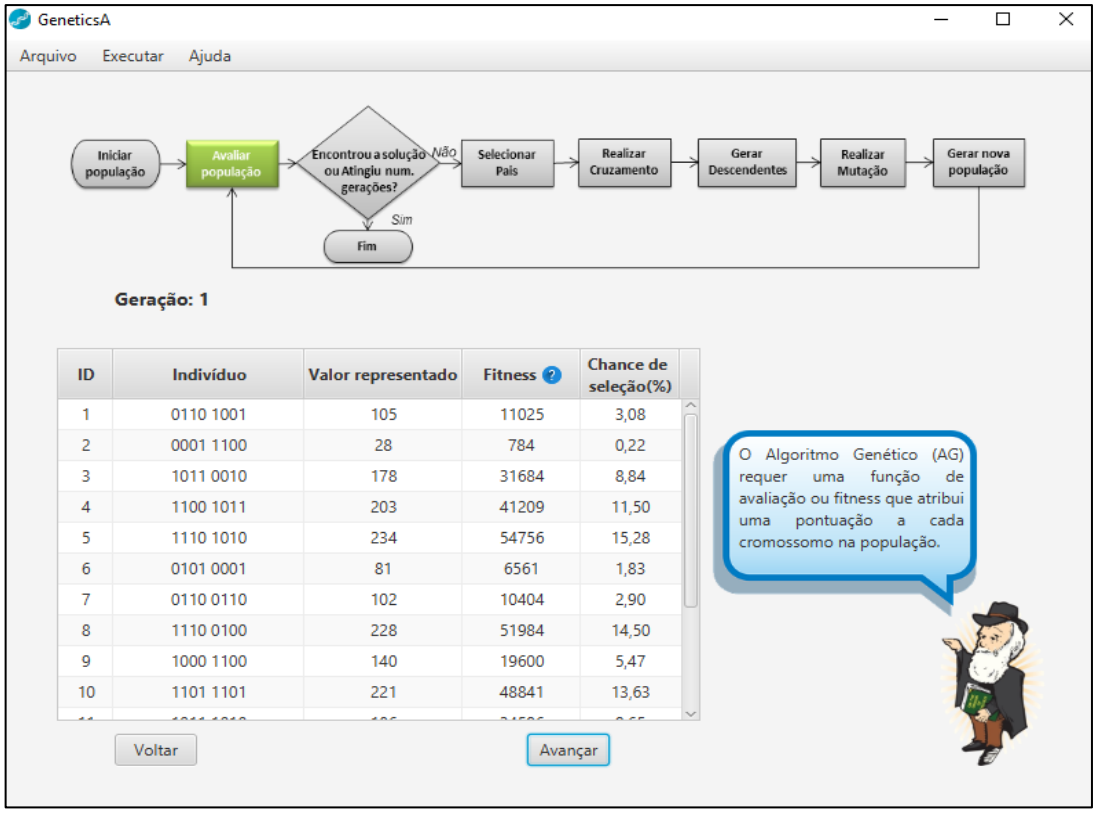
Fonte: Acervo do autor (2018)

4.2.4.3. Avaliar População

Nesta fase cada cromossomo é avaliado, a função de avaliação é uma maneira utilizada para determinar a qualidade de um indivíduo como solução do problema. Para o cálculo do fitness é necessário converter o conteúdo do cromossomo para seu valor decimal, a coluna “Valor representado” é o valor decimal de cada cromossomo.

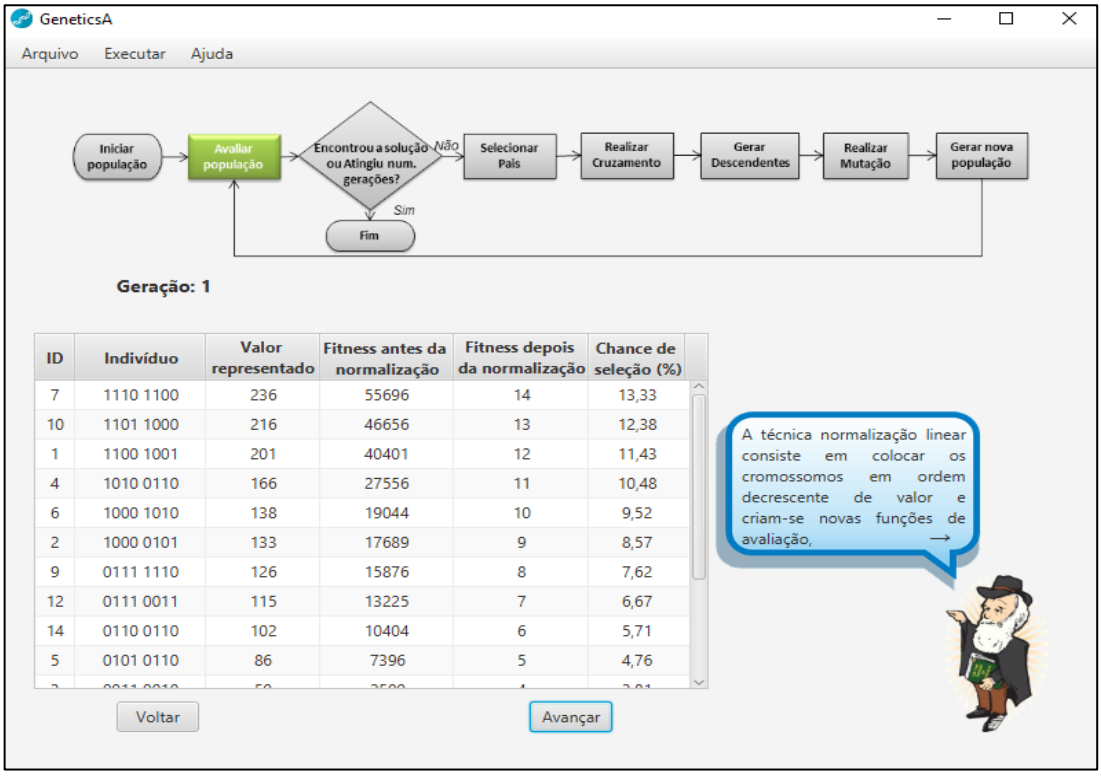
Na Figura 23 foi escolhida a função de avaliação simples, portanto, não é aplicada nenhuma técnica sobre o fitness dos cromossomos, ficando assim o valor de f(x). Já quando é escolhida outra função de avaliação, o fitness é alterado conforme a técnica. Na Figura 24 foi aplicada a técnica de normalização linear.

Figura 23 - Tela avaliar população (função de avaliação simples)



Fonte: Acervo do autor (2018)

Figura 24 - Tela avaliar população (Normalização linear)

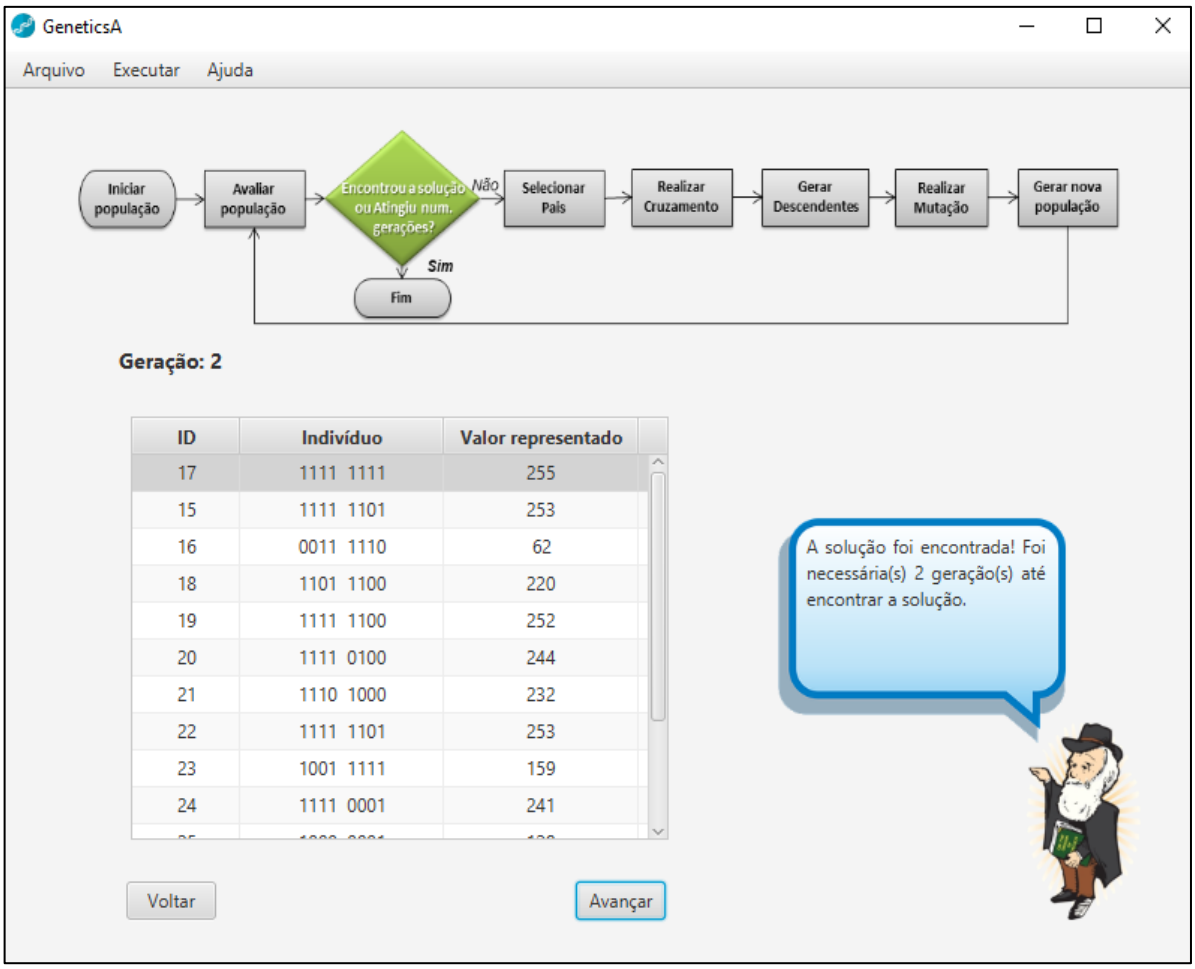


Fonte: Acervo do autor (2018)

4.2.4.4. Quando a condição de parada é satisfeita

Existem duas situações que podem levar a condição de parada, a primeira é quando atingir o número limite de gerações, e a segunda situação é quando encontra a solução. A Figura 25 ilustra quando é encontrada a solução do problema, o cromossomo solução ganha destaque em cinza e o balão de texto mostra quantas gerações foram necessárias até que a fosse solução encontrada.

Figura 25 - Tela quando a solução é encontrada



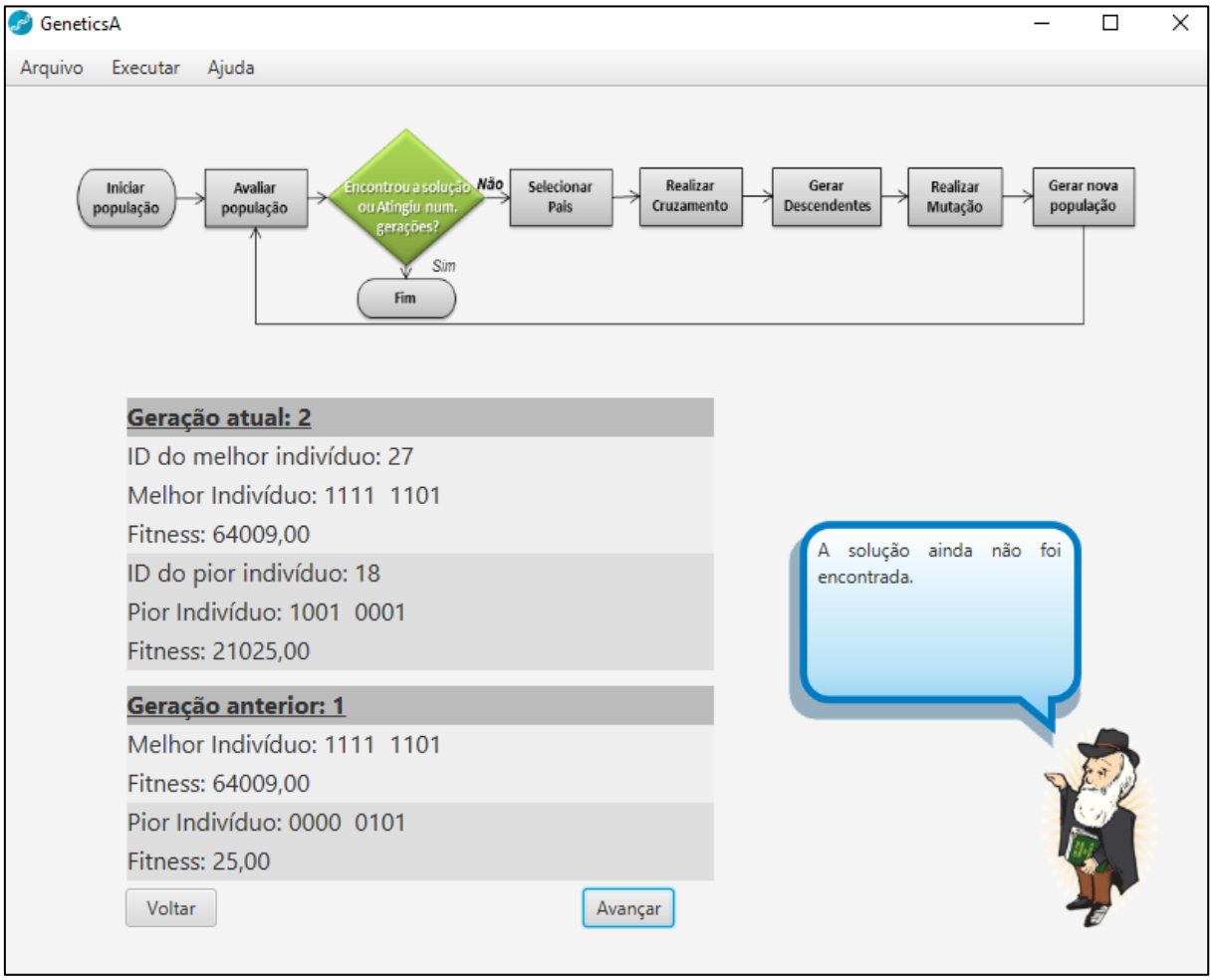
Fonte: Acervo do autor (2018)

No caso de atingir o limite de gerações sem ter encontrado a solução do problema, a mensagem no balão de informações será “O número máximo de gerações foi atingido! Tente trocar os parâmetros inicias ou aumente o número de gerações”.

4.2.4.5. Quando a condição de parada não é satisfeita

No caso da condição de parada não ser atendida, a tela da Figura 26 é apresentada com informações do cromossomo mais apto e menos apto da geração atual, e da anterior para efeitos de comparação.

Figura 26 - Tela quando a solução não é encontrada



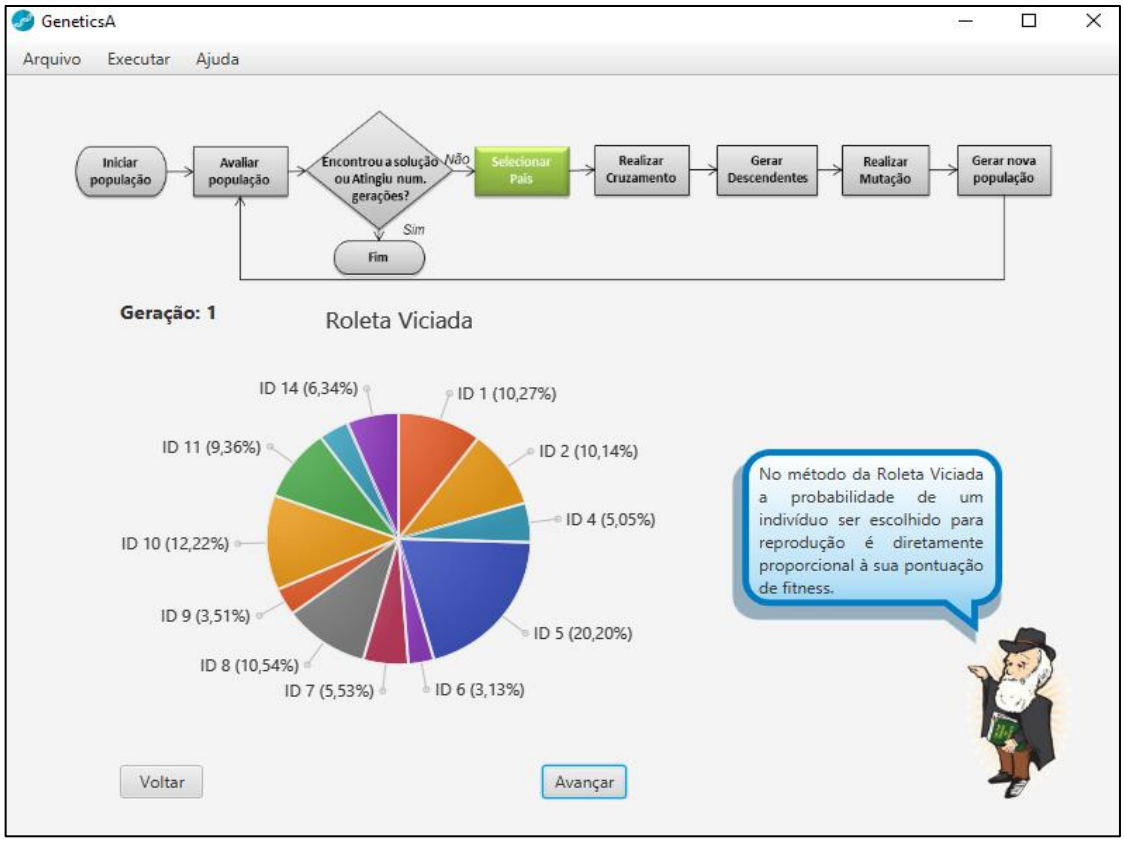
Fonte: Acervo do autor (2018)

No balão de informações do assistente Darwinho aparece uma mensagem dizendo que “A solução ainda não foi encontrada”.

4.2.4.6. Seleção dos pais

Na fase de seleção dos pais, existe uma tela distinta para cada método de seleção escolhido. A Figura 27 mostra a tela do método da Roleta Viciada. O conteúdo do balão de informações também é alterado conforme o método de seleção escolhido.

Figura 27 - Tela do método de seleção (Roleta Viciada)

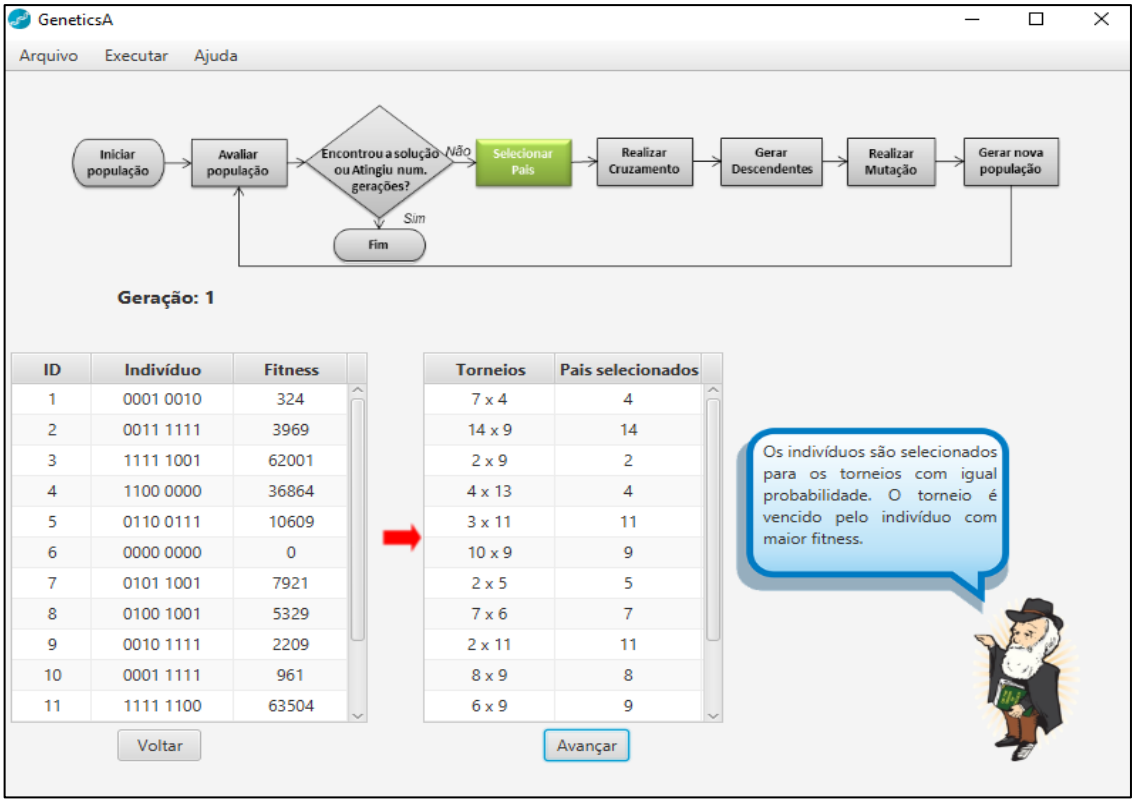


Fonte: Acervo do autor (2018)

Já na tela do método do torneio existem duas tabelas (veja a Figura 28), a do lado esquerdo contém informações como id, conteúdo do indivíduo e fitness. Já na tabela do lado direito é composta por uma coluna com os cromossomos que foram sorteados para participar dos torneios, e outra coluna com o vencedor de cada torneio.

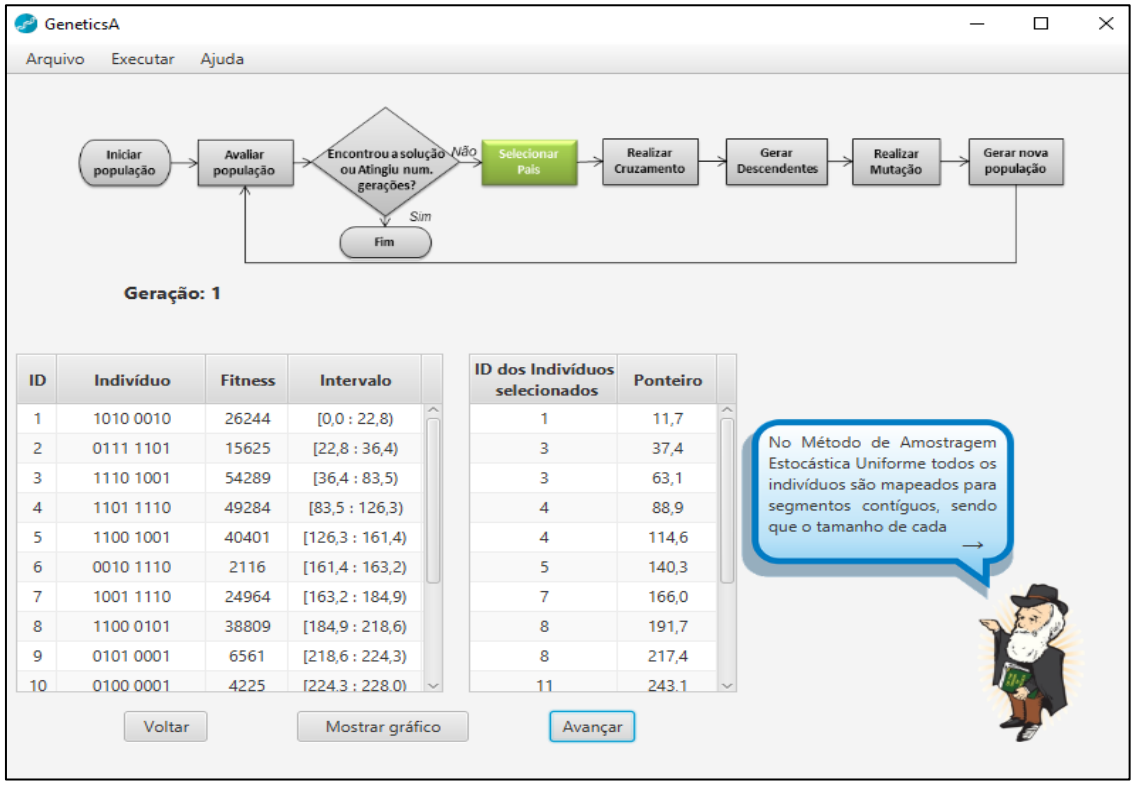
A tela do método de Amostragem Estocástica Uniforme é composta por duas opções de visualização: a primeira é através de tabelas e a segunda é graficamente. Na primeira opção existem duas tabelas, conforme pode ser visualizado na Figura 29. A tabela do lado esquerdo contém informações como id, conteúdo do indivíduo, fitness e o intervalo que cada cromossomo ocupa. A tabela do lado direito apresenta os cromossomos que foram selecionados conforme o ponteiro.

Figura 28 - Tela do método seleção (Método do torneio)



Fonte: Acervo do autor (2018)

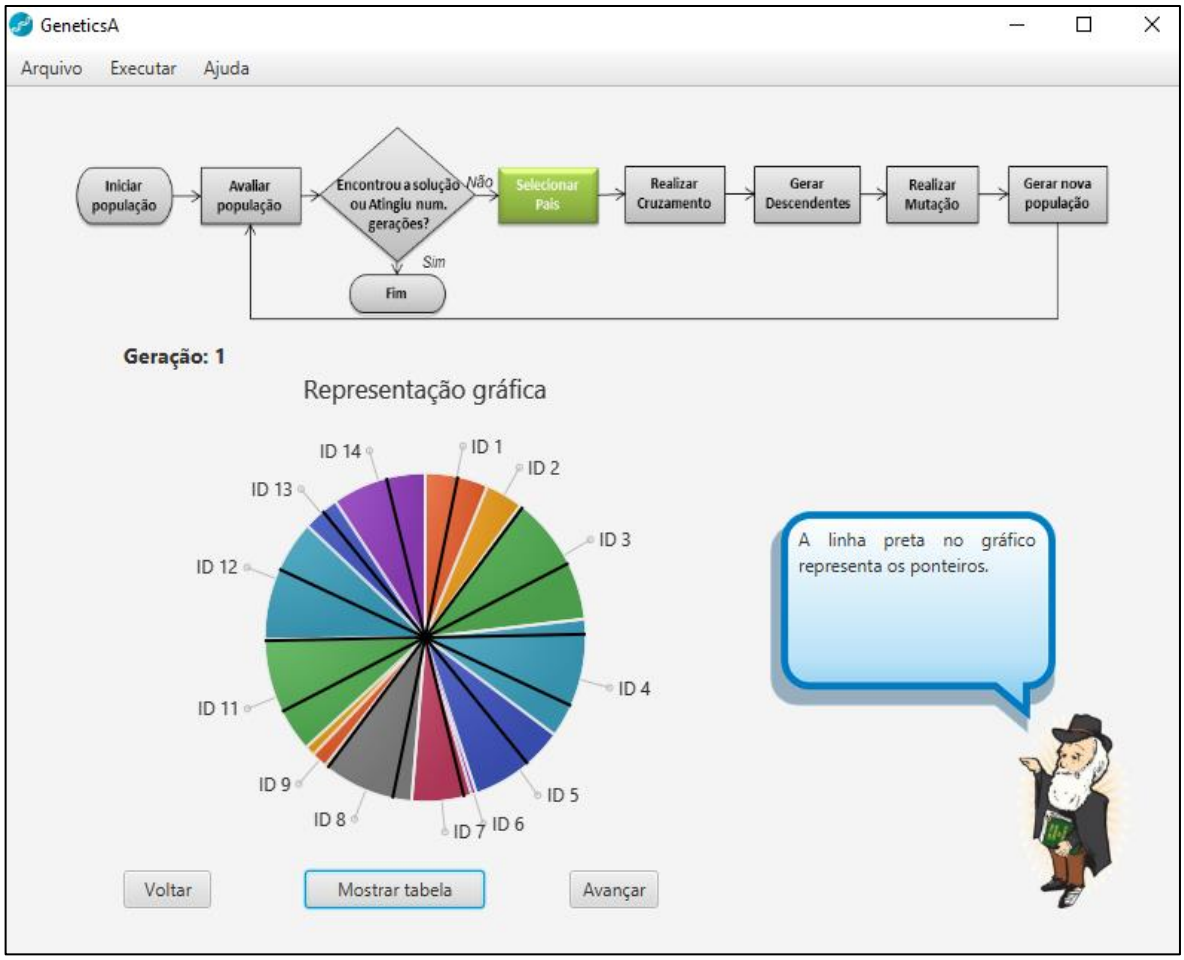
Figura 29 - Tela do método de seleção (Amostragem Estocástica Uniforme)



Fonte: Acervo do autor (2018)

A segunda opção de visualização é por representação gráfica, conforme pode ser visto na Figura 30. O intervalo é representado em graus no círculo, ficando assim cada cromossomo com sua fatia correspondente, os ponteiros são as retas de cor preta que cortam o círculo.

Figura 30 – Tela da representação gráfica do método de Amostragem Estocástica Uniforme

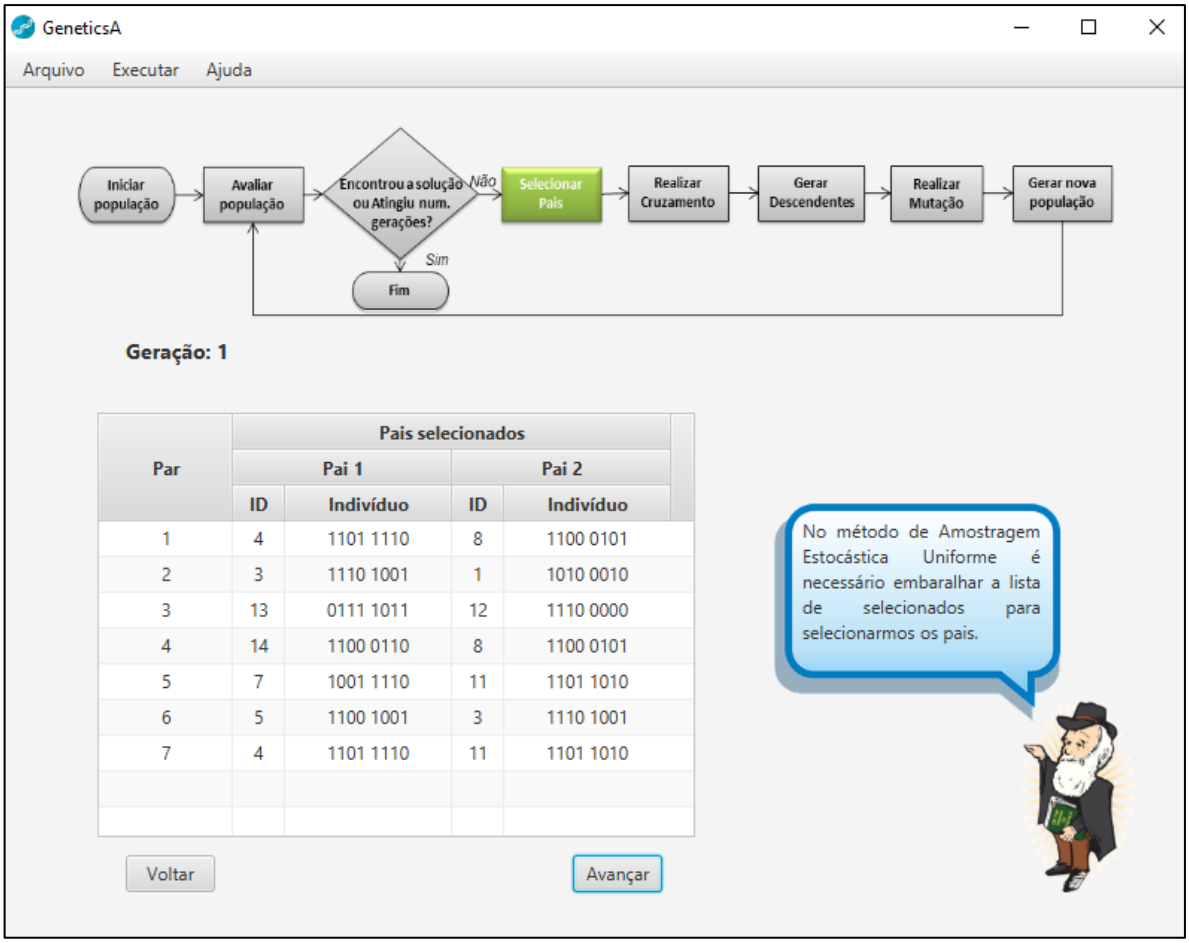


Fonte: Acervo do autor (2018)

Como pode ser observado na Figura 30, o indivíduo com “id 2” acabou não sendo selecionamento por ocupar uma parte menor do círculo, enquanto o pai com “id 4” vai ser pai duas vezes.

Depois que os pais são selecionados é apresentada uma tela com os pares que foram formados, conforme a Figura 31.

Figura 31 - Tela com os pares de pais selecionados



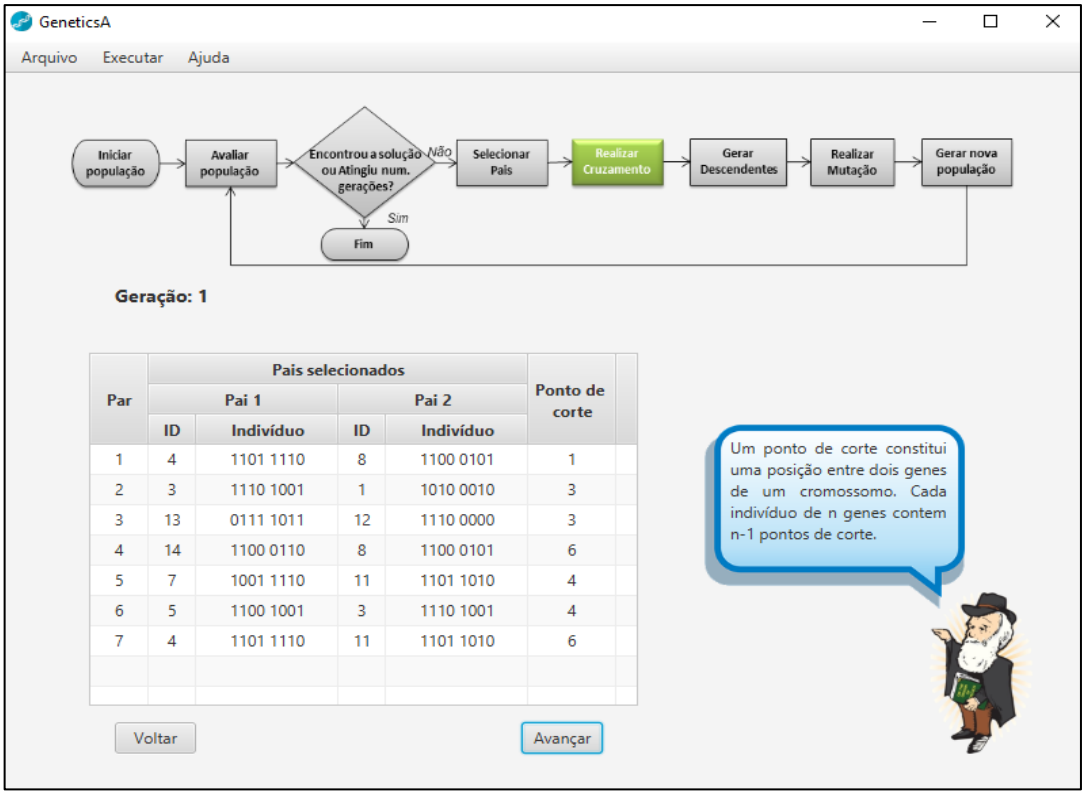
Fonte: Acervo do autor (2018)

4.2.4.7. Crossover

Agora que os pais estão organizados em pares, o próximo passo é realizar o crossover, se o operador de crossover for de “um ponto” ou de “dois pontos” é apresentado ao usuário o(s) ponto(s) de corte de cada par, conforme a Figura 32. Caso o operador de crossover seja o “uniforme” haverá um campo chamado “string de combinação”.

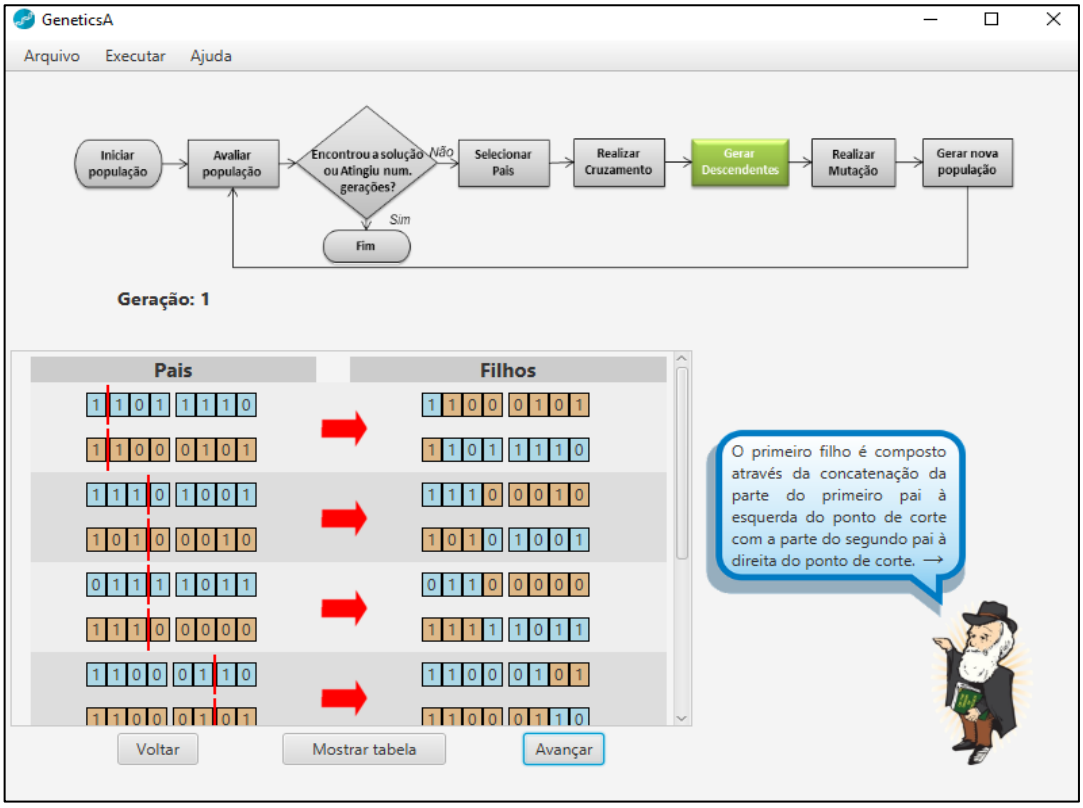
Na próxima fase são gerados os cromossomos filhos conforme o “ponto de corte” ou a “string de combinação”. O ponto de corte é representado por uma reta vertical na cor vermelha, a Figura 33 mostra como ocorre o crossover de “um ponto” de corte no GeneticsA.

Figura 32 - Tela com os pares de pais e o ponto de corte



Fonte: Acervo do autor (2018)

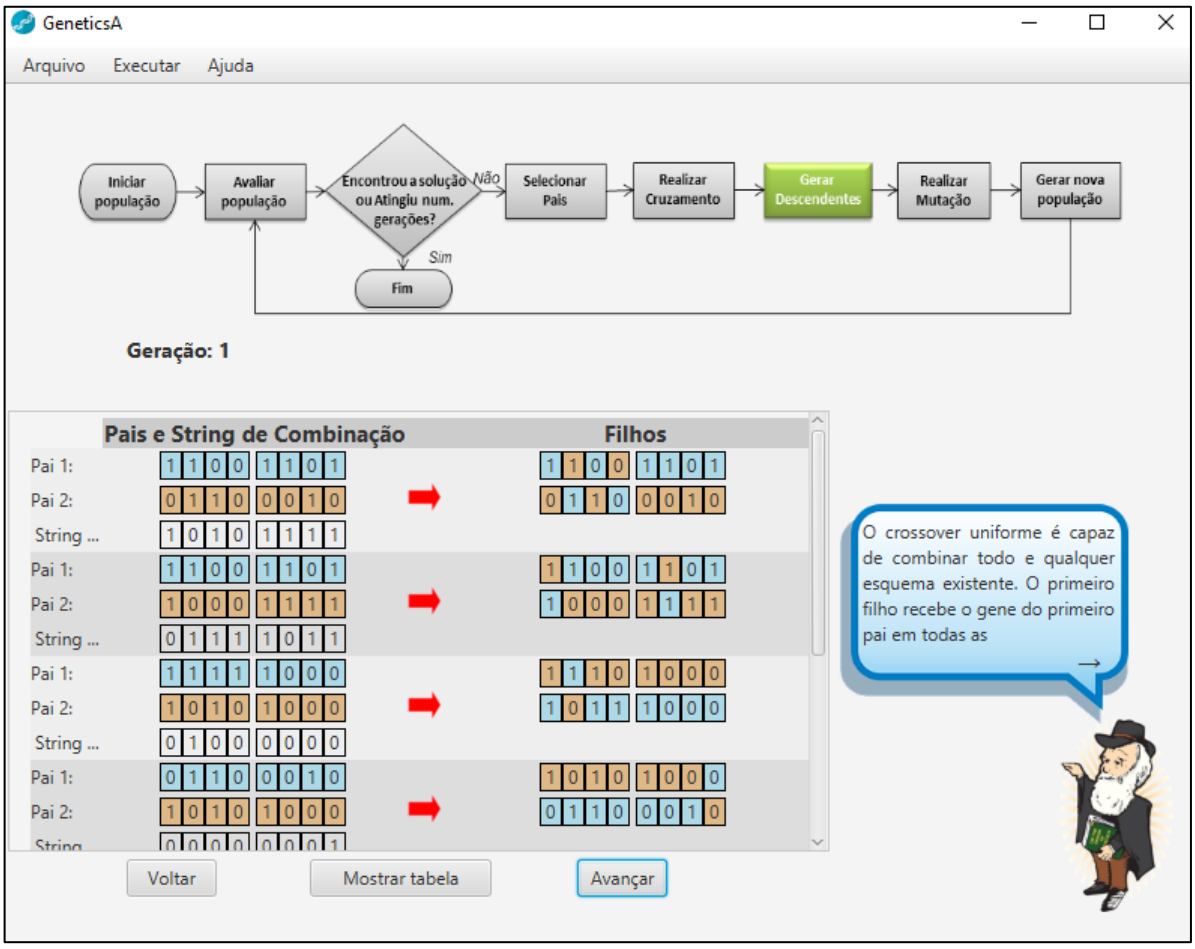
Figura 33 - Tela gerar descendentes com operador de crossover de um ponto de corte



Fonte: Acervo do autor (2018)

Caso o operador de crossover escolhido seja o “uniforme” é apresentado ao usuário à tela da Figura 34. A “string de combinação” está representada na cor cinza.

Figura 34 - Tela gerar descendentes com operador de crossover uniforme

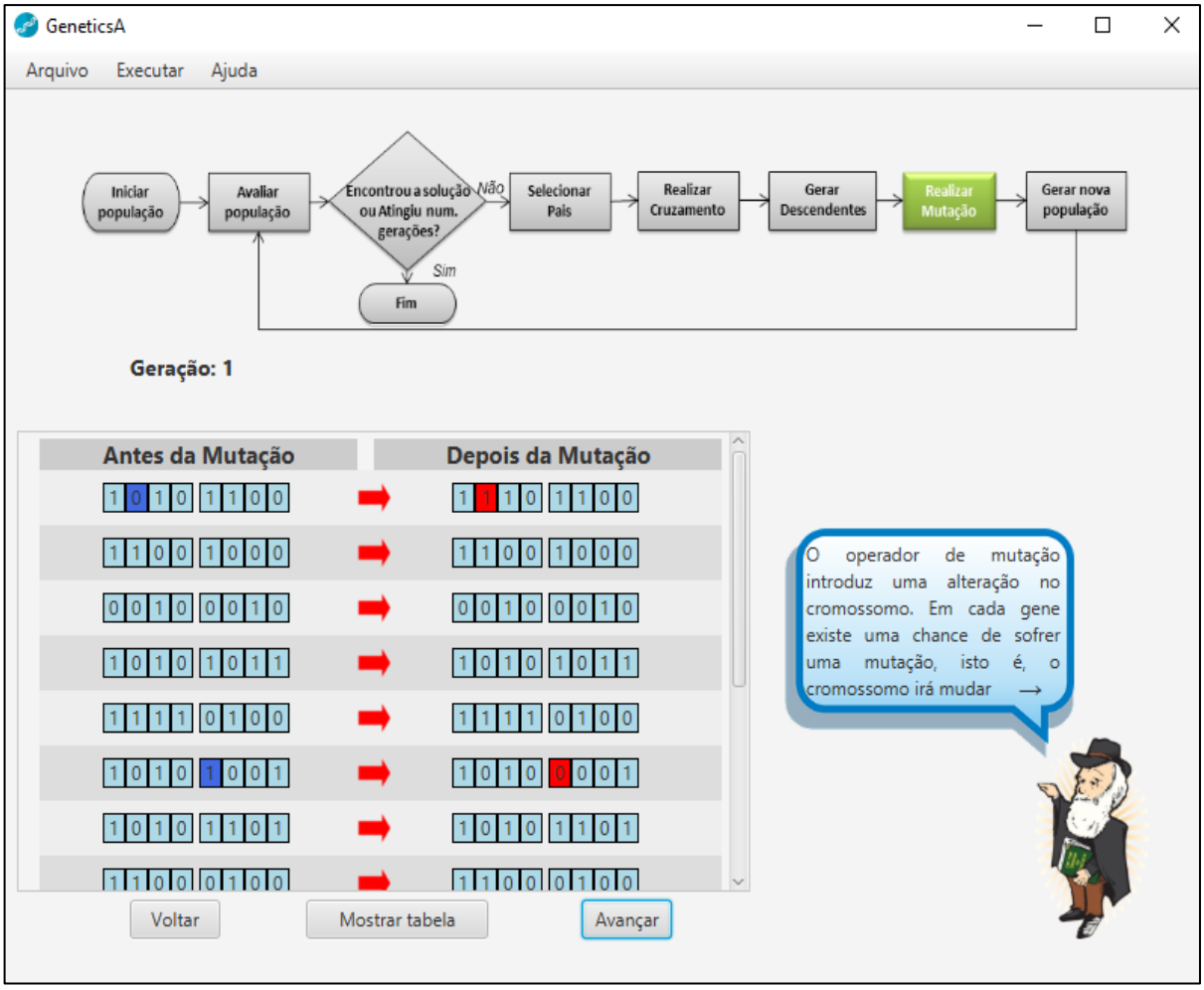


Fonte: Acervo do autor (2018)

4.2.4.8. Mutação

Depois que os cromossomos filhos foram gerados, ocorre o processo de mutação. O GeneticsA apresenta duas formas de visualização desta fase: visão gráfica ou através de uma tabela. A Figura 35 mostra a representação gráfica.

Figura 35 - Tela da representação gráfica da mutação



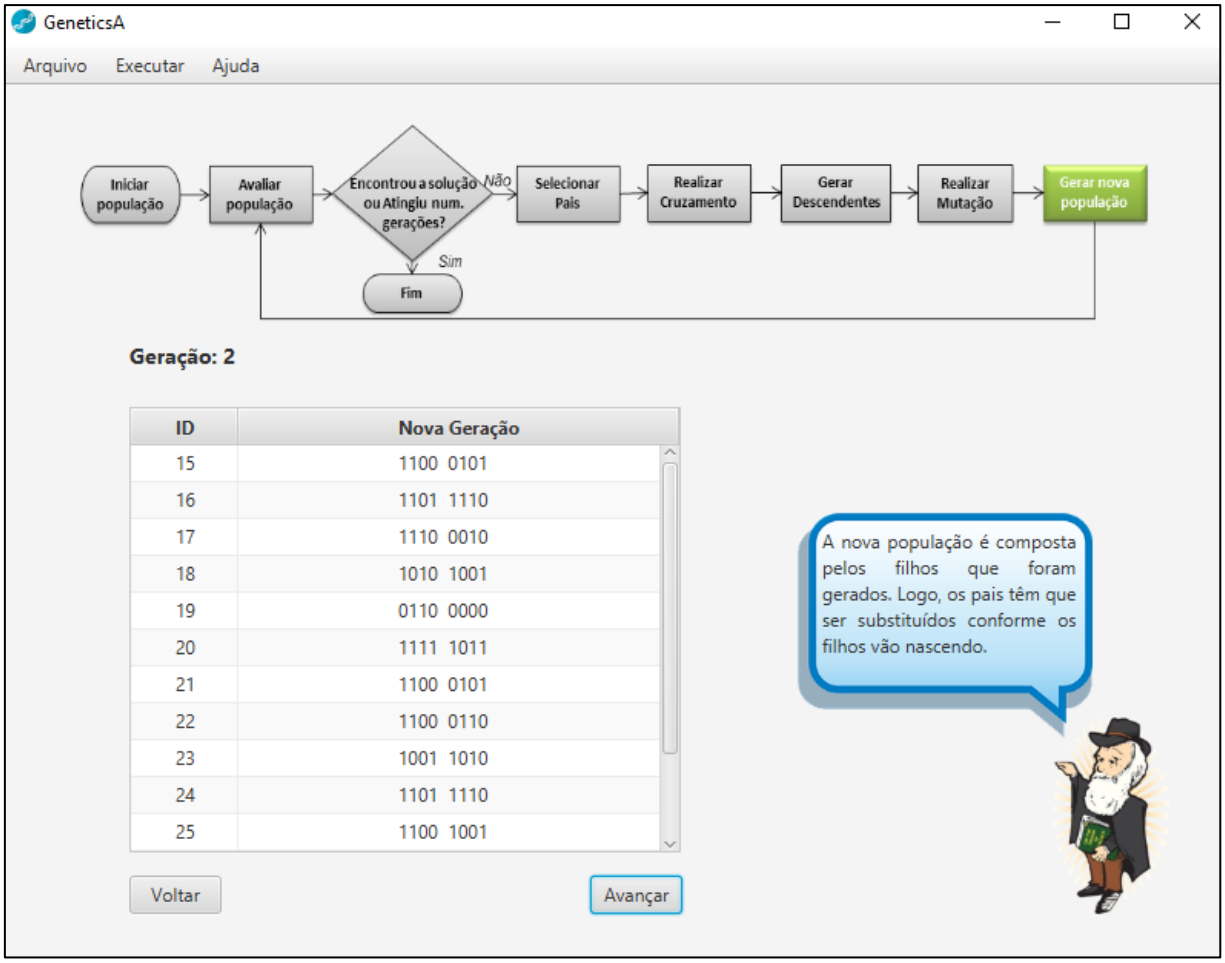
Fonte: Acervo do autor (2018)

O gene mutável é representado na cor “azul escuro” antes da mutação, e depois da mutação na cor “vermelha”.

4.2.4.9. Nova população

A nova população é formada pelos filhos que foram gerados, já a antiga geração de pais é descartada. Os novos indivíduos desta nova geração recebem seu id, a Figura 36 mostra como ficou a nova população.

Figura 36 - Tela nova população



Fonte: Acervo do autor (2018)

Agora que a nova população está formada, a próxima tela vai ser a da função de avaliação e todo o ciclo vai continuar até atingir o critério de parada.

4.2.5. Implementação do GeneticsA

Nesta seção serão apresentados os códigos de alguns dos principais métodos do GeneticsA, que estão presentes no diagrama de classes da Figura 19.

4.2.5.1. Gerar população inicial

Depois que o usuário escolheu os parâmetros do GeneticsA da Figura 21, a população inicial deve ser gerada. O método *gerarPopulacaoInicial* do Quadro 6 é o ponto de partida para preencher a população inicial.

Quadro 6 - Código do método *gerarPopulacaoInicial*

```
1   public void gerarPopulacaoInicial(int tamanho) {
2      populacao = new Populacao();
3      populacao.setTamanho(tamanho);
4      populacao.setGeracao(1);
5      ArrayList cromossomos = new ArrayList<Cromossomo>();
6
7      for (int i = 0; i < populacao.getTamanho(); i++) {
8         Cromossomo c = new Cromossomo();
9         c.setId(i + 1);
10         c.setTamanho(solucao.length());
11         c.setValor(gerarValorDoCromossomoInicial());
12         cromossomos.add(c);
13         populacao.setCromossomos(cromossomos);
14      }
15      setQuantCromossomosJaGerados(tamanho);
16   }
```

Fonte: Acervo do autor (2018)

O laço de repetição da linha 7 do Quadro 6 cria os cromossomos e já inseri alguns atributos importantes como id, tamanho e valor. A linha 11 chama o método *gerarValorDoCromossomoInicial* do Quadro 7, este método é responsável por gerar um cromossomo de valor aleatório, desde que não tenha outro cromossomo com o mesmo valor na população.

Quadro 7 - Código do método *gerarValorDoCromossomoInicial*

```
1   private String gerarValorDoCromossomoInicial() {
2      String valor;
3      do {
4         Random gerarNum = new Random();
5         StringBuilder stringBuilder = new StringBuilder();
6         for (int i = 0; i < solucao.length(); i++) {
7            stringBuilder.append(gerarNum.nextInt(2));
8         }
9         valor = stringBuilder.toString();
10      } while ((existeCromossomoIgualNaPopulacao(valor)) &&
11            (populacao.getCromossomos() != null));
12      return valor;
13   }
```

Fonte: Acervo do autor (2018)

O laço de repetição da linha 3 do Quadro 7 é responsável por gerar um valor aleatório e só termina se este valor não for igual ao de outro cromossomo gerado. O método *existeCromossomoIgualNaPopulacao* como o próprio nome diz faz uma varredura na população se já existe um cromossomo igual, caso exista é necessário gerar outro valor.

4.2.5.2. Funções de avaliação

O próximo passo é avaliar os cromossomos conforme a função de avaliação escolhida. O método *avaliar* do Quadro 8 é responsável por determinar o fitness para cada cromossomo, de acordo com a função de avaliação. Primeiramente é necessário calcular o valor representado (valor decimal) de cada cromossomo, este valor é necessário para o cálculo do fitness. O trecho de código 7-18 do Quadro 8 realiza cálculo do fitness simples conforme o problema que foi escolhido no GeneticsA, é necessário ser realizado este cálculo porque as outras funções de avaliação necessitam desta informação. Caso a função de avaliação escolhida seja a Normalização Linear, os cromossomos são ordenados em ordem decrescente (24-33) e depois é calculado seu fitness (35-38).

Quadro 8 - Código do método *avaliar*

```
1  public void avaliar(String funAvaliacao, String problema) {
2     for (int i = 0; i < getTamanho(); i++) {
3        c.calcularValorRepresentado();
4     }
5
6     //Realiza o calculo do fitness simples
7     for (int i = 0; i < getTamanho(); i++) {
8
9        //Calcula o fitness simples conforme o problema
10       if (problema.equalsIgnoreCase("Maximização da função f(x) = x²")) {
11          c.setFitness(c.getValorRepresentado() * c.getValorRepresentado());
12          c.setFitnessSimples(c.getValorRepresentado() * c.getValorRepresentado());
13       } else if (problema.equalsIgnoreCase("Maximização da função f(x) = -x+4096")) {
14          c.setFitness(-1 * c.getValorRepresentado() + 4096);
15          c.setFitnessSimples(-1 * c.getValorRepresentado() + 4096);
16       }
17       getCromossomos().set(i, c);
18    }
19
20    //Função de avaliação: Normalização linear
21    if (funAvaliacao.equalsIgnoreCase("Normalização linear")) {
22       double k = getTamanho() + 1;
23       double t = 1;
24       //Ordena os cromossomos
25       for (int i = 0; i < getTamanho(); i++) {
26          for (int j = 0; j < getTamanho() - 1; j++) {
27             if (c1.getFitness() < c2.getFitness()) {
28                aux = c1;
29                c1 = c2;
30                c2 = aux;
31             }
32          }
33       }
34
35       for (int i = 0; i < getTamanho(); i++) {
36          c.setFitness(k - t);
37          k = k - t;
38       }
39    //Função de avaliação: Windowing
40    } else if (funAvaliacao.equals("Windowing")) {
41       //Procura o menor valor
```

```
42          for (int i = 0; i < getTamanho(); i++) {
43              Cromossomo c = (Cromossomo) getCromossomos().get(i);
44              if (c.getFitness() < menor) {
45                  menor = c.getFitness();
46              }
47          }
48
49          if (menor != 0) {
50              for (int i = 0; i < getTamanho(); i++) {
51                  fitness = cromossomos.get(i).getFitness();
52                  //Acrecenta +1 para evitar que um cromossomo tenha fitness = 0
53                  cromossomos.get(i).setFitness(fitness - menor + 1);
54              }
55          }
56      //Função de avaliação: Escalonamento sigma
57      } else if (funAvaliacao.equals("Escalonamento sigma")) {
58          double desvioPadrao = calcularDesvioPadrao();
59          double fitness;
60
61          //Se o desvio padrão é zero, todos cromossomos recebem fitness = 1
62          if (desvioPadrao == 0) {
63              fitness = 1;
64              for (int i = 0; i < getTamanho(); i++) {
65                  cromossomos.get(i).setFitness(fitness);
66              }
67          } else {
68              double media = calcularMediaDoFitness();
69              for (int i = 0; i < getTamanho(); i++) {
70                  fitness = 1 + ((cromossomos.get(i).getFitness() - media) / (2 * desvioPadrao));
71
72                  //Se a função se torna negativa para algum indivíduo
73                  if (fitness < 0.00) {
74                      fitness = 0.1;//Para ter alguma chance de seleção
75                  }
76                  cromossomos.get(i).setFitness(fitness);
77              }
78          }
79      }
80  }
```

Fonte: Acervo do autor (2018)

O trecho de código 40-55 do Quadro 8 é executado caso a função de avaliação escolhida seja Windowing, nesta função é necessário encontrar o cromossomo com menor fitness (42-47), e depois é subtraído este *menor valor + 1* de todos os cromossomos. Já o trecho de código 58-72 é executado somente quando a função de avaliação Escalonamento Sigma for escolhida, fazendo o calculo conforme a fórmula mostrada anteriormente.

4.2.5.3. Métodos de seleção

Agora que os indivíduos já foram avaliados, a próxima fase é selecionar os cromossomos que vão fazer parte do crossover. O método *selecionar* como o próprio nome sugere, realiza a seleção conforme o método escolhido.

Ao escolher o método da Roleta Viciada, o trecho de código 3-16 do Quadro 9 é executado. Neste método cada cromossomo ocupa uma “parte” proporcional a sua chance de seleção. Já no método do torneio são “sorteados” indivíduos para disputar cada torneio (linha 19-20) e o vencedor de cada torneio esta selecionado para a etapa de crossover. Outro método disponível no GeneticsA é o Amostragem Estocástica, sendo executado no trecho 37-65 do Quadro 9.

Quadro 9 - Código do método *selecionar*

```
1  public void selecionar(String metSelecao) {
2      if (metSelecao.equals("Roleta Viciada")) {
3          //Seleciona os pais
4          for (int i = 0; i < getTamanho(); i++) { 11
5              double numSorteado = gerarNum.nextDouble();
6              fimIntervaloRoleta = 0;
7              //Analisa qual cromossomo foi selecionado
8              for (int j = 0; j < getTamanho(); j++) {
9                  inicioIntervaloRoleta = fimIntervaloRoleta;
10                 fimIntervaloRoleta += cromossomos.get(j).getChanceDeSelecao();
11                 if (inicioIntervaloRoleta <= numSorteado && fimIntervaloRoleta > numSorteado) {
12                     cromoSelecionados.add(cromossomos.get(j));
13                     break;
14                 }
15             }
16         }
17     } else if (metSelecao.equalsIgnoreCase("Método do torneio")) {
18         for (int i = 0; i < getTamanho(); i++) {
19             int membro1 = gerarNum.nextInt(getTamanho());
20             int membro2 = gerarNum.nextInt(getTamanho());
21
22             /*Verificação para evitar cromossomo igual no mesmo torneio e para evitar cromossomo igual no casal*/
23             while (membro1 == membro2) {
24                 membro1 = gerarNum.nextInt(getTamanho());
25                 membro2 = gerarNum.nextInt(getTamanho());
26             }
27         }
28         //Analisa o vencedor de cada torneio
29         for (int i = 0; i < cromossomosQueParticipamDoTorneio.size(); i += 2) {
30             if (cromossomosQueParticipamDoTorneio.get(i).getFitness()
31                     > cromossomosQueParticipamDoTorneio.get(i + 1).getFitness()) {
32                 cromoSelecionados.add(cromossomosQueParticipamDoTorneio.get(i));
33             } else {
34                 cromoSelecionados.add(cromossomosQueParticipamDoTorneio.get(i + 1));
35             }
36         }
37     } else if (metSelecao.equalsIgnoreCase("Amostragem Estocástica")) {
38         //O início deve estar entre 0 e 360/tamanho da população
39         double inicio = gerarNum.nextDouble() * 360 / getTamanho();
40         ponteirosAux.add(inicio);
41
42         /*Agora que foi sorteado o início será determinado os outros
43          ponteiros pela fórmula*/
44         double tamIntervalo = 360.0 / getTamanho();
45         for (int i = 1; i < getTamanho(); i++) {
46             ponteirosAux.add(inicio + i * tamIntervalo);
47         }
48         setPonteiros(ponteirosAux);
49         double inicioIntervalo = 0;
50         //Determina o intervalo de cada indivíduo
51         for (int i = 0; i < getTamanho(); i++) {
52             fimIntervalo = inicioIntervalo + 360 * (cromossomos.get(i).getChanceDeSelecao());
```

```
53              cromossomos.get(i).setInicioIntervalo(inicioIntervalo);
54              cromossomos.get(i).setFimIntervalo(fimIntervalo);
55              inicioIntervalo = fimIntervalo;
56          }
57          for (int i = 0; i < getTamanho(); i++) {
58              for (int j = 0; j < getTamanho(); j++) {
59                  if (getPonteiros().get(i) >= cromossomos.get(j).getInicioIntervalo()
60                          && getPonteiros().get(i) <= cromossomos.get(j).getFimIntervalo()) {
61                      cromoSelecionados.add(cromossomos.get(j));
62                  }
63              }
64          }
65      }
66      for (int i = 0; i < cromoSelecionados.size(); i++) {
67          cromoSelecionadosAux.add(cromoSelecionados.get(i));
68      }
69      int contCasal = 1;
70      //Forma os casais
71      while (contCasal <= getTamanho() / 2) {
72          numPai1 = gerarNumero1.nextInt(cromoSelecionadosAux.size());
73          numPai2 = gerarNumero2.nextInt(cromoSelecionadosAux.size());
74          //Os dois pais precisam ser diferentes na lista
75          while (numPai1 == numPai2) {
76              numPai1 = gerarNumero1.nextInt(cromoSelecionadosAux.size());
77              numPai2 = gerarNumero2.nextInt(cromoSelecionadosAux.size());
78          }
79          //Se os pais são diferentes está apto a reprodução
80          if (pai1.getId() != pai2.getId()) {
81              pais.add(pai1);
82              pais.add(pai2);
83              contCasal++;
84          }
85          /*Caso o último casal seja composto de pai1 e pai2 iguais é necessário reembalharar,
86           evitando assim que entre em loop infinito*/
87          if (contCasal == getTamanho() / 2 && pai1.getId() == pai2.getId()) {
88              for (int i = 0; i < pais.size(); i++) {
89                  cromoSelecionadosAux.add(pais.get(i));
90              }
91              contCasal = 1;
92              pais.clear();
93          }
94      }//Termina de formar os casais
95      setPaisSelecionados(pais);
96  }
```

Fonte: Acervo do autor (2018)

O trecho de código 71-94 do Quadro 9 é executado independentemente do método de seleção escolhido, sendo responsável por combinar os casais, desde que o pai1 seja diferente do pai2.

4.2.5.4. Operadores Crossover

No GeneticsA existem 3 operadores de crossover: um ponto de corte, dois pontos de corte e o uniforme. O trecho de código 2-18 do Quadro 10 realiza o cruzamento de um ponto de corte.

O crossover de dois pontos de corte inicia na linha 20 e termina na linha 53. Neste operador é necessário um cuidado a mais, existe um comando de repetição na linha 30, para garantir que os dois pontos de corte sejam distintos.

Quadro 10 - Código do método *fazerCrossover*

```
 1   public void fazerCrossover(String opCrossover) {
 2      if (opCrossover.equals("Um Ponto")) {
 3         int pontoDeCorte = -1;
 4         for (int i = 0; i < getTamanho(); i += 2) {
 5            c = (Cromossomo) getPaisSelecionados().get(i); 12
 6            //Gera um ponte de corte
 7            pontoDeCorte = gerarNum.nextInt(c.getTamanho() - 1) + 1;
 8            pontsCorte.add(pontoDeCorte);
 9            //Realiza o crossover de 1 ponto de corte
10             valorFilho1 = pai1.getValor().substring(0, pontoDeCorte) +  pai2.getValor().substring(pontoDeCorte);
11             valorFilho2 = pai2.getValor().substring(0, pontoDeCorte) + pai1.getValor().substring(pontoDeCorte);
12             //Termina o crossover
13
14             filho1.setValor(valorFilho1);
15             filho2.setValor(valorFilho2);
16          }
17          setPontosDeCorte(pontsCorte);
18          setFilhosGerados(filhos);
19       } else if (opCrossover.equalsIgnoreCase("Dois pontos")) {
20          int pontoDeCorte1 = -1;
21          int pontoDeCorte2 = -1;
22          for (int i = 0; i < getTamanho(); i += 2) {
23             c = (Cromossomo) getPaisSelecionados().get(i);
24
25             //Gera os dois pontos de corte
26             pontoDeCorte1 = gerarNum.nextInt(c.getTamanho() - 1) + 1;
27             pontoDeCorte2 = gerarNum.nextInt(c.getTamanho() - 1) + 1;
28
29             //Garanti que os pontos de cortes serão diferentes
30             while (pontoDeCorte1 == pontoDeCorte2) {
31                pontoDeCorte2 = gerarNum.nextInt(c.getTamanho() - 1) + 1;
32             }
33
34             /*O Primeiro ponto de corte deve ser menor que o segundo ponto de corte*/
35             if (pontoDeCorte2 < pontoDeCorte1) {
36                int aux = pontoDeCorte1;
37                pontoDeCorte1 = pontoDeCorte2;
38                pontoDeCorte2 = aux;
39             }
40
41             pontsCorte.add(pontoDeCorte1);
42             pontsCorte.add(pontoDeCorte2);
43             //Realiza o crossover de 2 pontos de corte
44             valorFilho1 = pai1.getValor().substring(0, pontoDeCorte1) + pai2.getValor().substring(pontoDeCorte1,
pontoDeCorte2) + pai1.getValor().substring(pontoDeCorte2);
45             valorFilho2 = pai2.getValor().substring(0, pontoDeCorte1) + pai1.getValor().substring(pontoDeCorte1,
pontoDeCorte2) + pai2.getValor().substring(pontoDeCorte2);
46             //Termina o crossover
47
48             filho1.setValor(valorFilho1);
49             filho2.setValor(valorFilho2);
50          }
51          setPontosDeCorte(pontsCorte);
52          setFilhosGerados(filhos);
53       } else if (opCrossover.equals("Uniforme")) {
54          int tamCromossomo = cromossomos.get(0).getTamanho();
55          //Gera todas as strings de combinação
```

```
56          for (int i = 0; i < getTamanho(); i++) {
57             StringBuilder stringBuilder = new StringBuilder();
58             for (int j = 0; j < tamCromossomo; j++) {
59                stringBuilder.append(gerarNum.nextInt(2));
60             }
61             valor = stringBuilder.toString();
62             stringCombCrossover.add(valor);
63          }
64
65          //Início do crossover uniforme
66          for (int i = 0, k = 0; i < paisSelecionados.size(); i += 2, k++) {
67             /*Passa por todos os genes do cromossomo e gera os filhos conforme a string de combinação */
68             for (int j = 0; j < tamCromossomo; j++) {
69                String stringComb = (String) stringCombCrossover.get(k);
70
71                //Realiza o crossover uniforme
72                if (stringComb.charAt(j) == '1') {
73                   stringBuilderFilho1.append(paisSelecionados.get(i).getValor().charAt(j));
74                   stringBuilderFilho2.append(paisSelecionados.get(i + 1).getValor().charAt(j));
75                } else {
76                   stringBuilderFilho2.append(paisSelecionados.get(i).getValor().charAt(j));
77                   stringBuilderFilho1.append(paisSelecionados.get(i + 1).getValor().charAt(j));
78                }
79             }
80             filho1.setValor(stringBuilderFilho1.toString());
81             filho2.setValor(stringBuilderFilho2.toString());
82          }//Fim do crossver
83          setFilhosGerados(filhos);
84          setStringCombinacaoCrossover(stringCombCrossover);
85       }
86    }
```

Fonte: Acervo do autor (2018)

Caso o operador de crossover uniforme seja escolhido o trecho de código 53-85 será executado. O operador de crossover uniforme necessita de uma string de combinação, para depois realizar o processo de cruzamento.

4.2.5.5. Mutação

Depois que o crossover foi realizado, o próximo passo é a mutação, nesta fase cada gene do cromossomo vai ter uma chance *x* de sofrer mutação. Veja o código fonte do método *gerarMutacao* no Quadro 11.

Quadro 11 - Código do método *gerarMutação*

```
1   public boolean gerarMutacao(double taxaMutacao) {
2      double numSorteado;
3      String valorAux;
4
5      //Passa por todos os genes do cromossomo
6      for (int k = 0; k < getTamanho(); k++) {
7         Random gerarNum = new Random();
8
9         //Gera um número entre (0,1)
10         numSorteado = gerarNum.nextDouble();
```

```
11
12          //Faz a mutação
13          if (numSorteado < taxaMutacao) {
14              valorAux = null;
15              char chars[] = getValor().toCharArray();
16
17              //Realiza a mutação
18              if (chars[k] == '1') {
19                  chars[k] = '0';
20              } else {
21                  chars[k] = '1';
22              }
23
24              valorAux = new String(chars);
25              setValorAntesMutacao(getValor());
26              setValor(valorAux);
27              setPosMutacao(k + 1);//Salva a posição que ocorreu a mutação
28              return true; //Termina de fazer a mutação
29          } else {
30              setPosMutacao(-1);
31              setValorAntesMutacao(getValor());
32          }
33      }
34      return false; //Caso não tenha sido realizada a mutação
35  }
```

Fonte: Acervo do autor (2018)

Depois de ser realizada a mutação, a antiga geração é descartada e a nova geração passa pela função de avaliação, e todo o ciclo do Algoritmo Genético se repete até que a condição de parada seja satisfeita.

Essa seção abordou algumas linhas de código fundamentais para o funcionamento do GeneticsA. Essas linhas receberam ligeiros comentários e, obviamente, não contemplaram todas as instruções por trás das chamadas dos métodos utilizados para o completo desenvolvimento do software.

Neste capítulo buscou-se abordar as especificações formais do GeneticsA, através da engenharia de requisitos e modelagem UML. Posteriormente, foram apresentadas as ferramentas e técnicas utilizadas no desenvolvimento, e depois as principais telas do software. Por fim, foram abordados os trechos de códigos mais fundamentais das funções, métodos e operadores implementados no GeneticsA.

5. RESULTADOS

5.1. PESQUISA DE SATISFAÇÃO DE USO

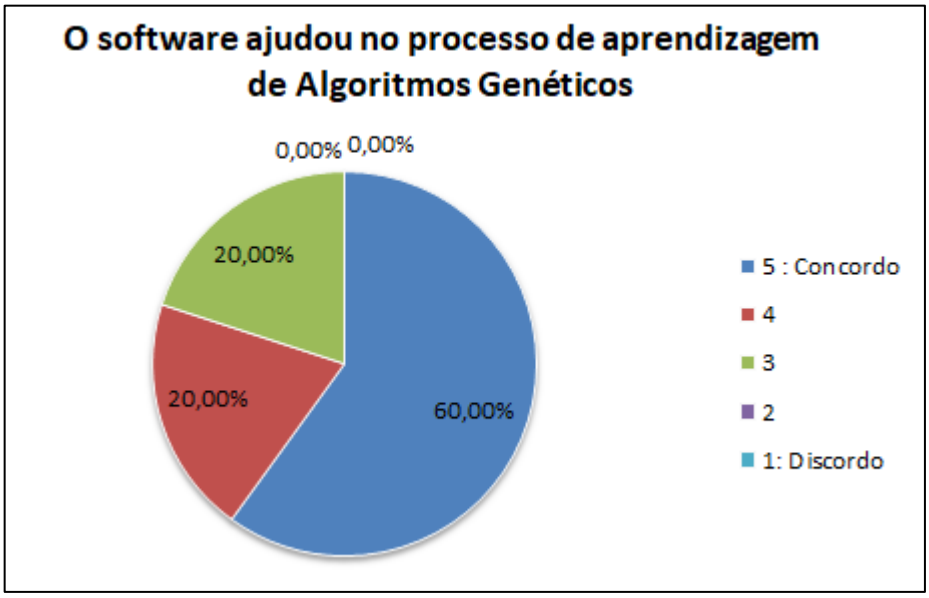
Foi aplicada uma pesquisa quantitativa com os alunos da disciplina de Inteligência Artificial do curso de Ciência da Computação do Instituto Federal Catarinense - Campus Rio do Sul. A amostra de participantes envolveu cinco alunos, no primeiro momento foi apresentado o software aos alunos e depois entregue o questionário do APÊNDICE A. O questionário é composto por 17 perguntas de escala, 1 pergunta dicotômica e 1 campo de comentários.

Os pontos das perguntas de escala revelam a força e a direção da atitude dos participantes da pesquisa em relação às extremidades da escala, que possuem categorizações antagônicas e com a alternativa neutro ao centro (REZENDE, 2013). No tópico a seguir serão apresentados os gráficos das perguntas mais relevantes e os resultados da pesquisa.

5.1.1. Resultados da pesquisa de satisfação de uso

A maioria dos entrevistados (60%) disseram que o software ajudou no processo de aprendizagem de Algoritmos Genéticos, 20% deram a nota 4 e 20% foram neutros, como pode ser observado no Gráfico 1.

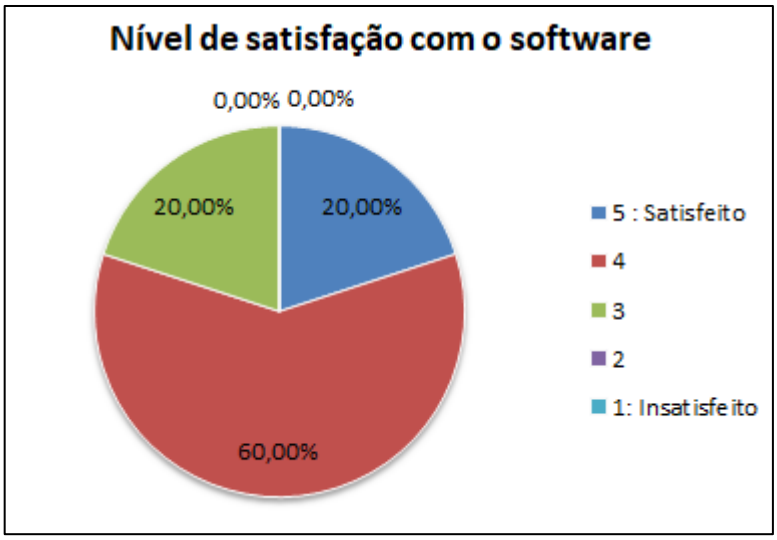
Gráfico 1 - Ajuda no processo de aprendizagem de Algoritmos Genéticos



Fonte: Acervo do autor (2018)

O Gráfico 2 mostra 20% deram a nota máxima no quesito nível de satisfação com o software, 60% avaliavam com nota 4 e outros 20% com a nota 3.

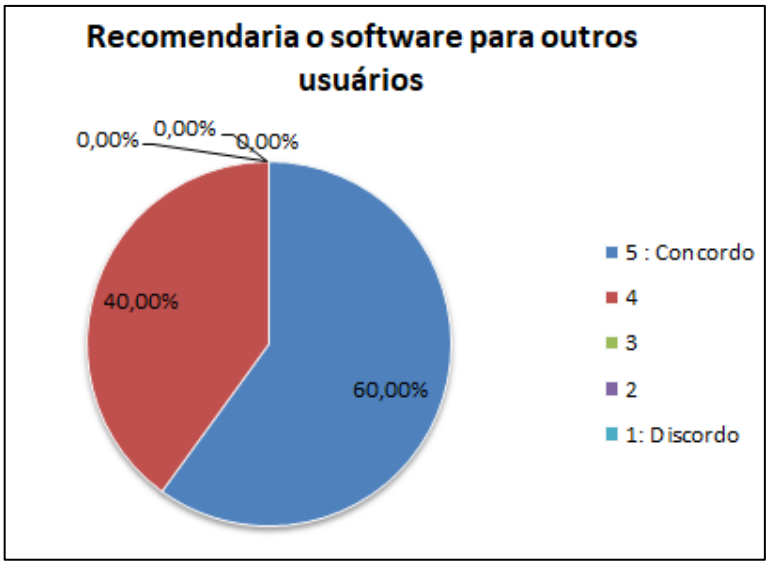
Gráfico 2 - Nível de satisfação com o GeneticsA



Fonte: Acervo do autor (2018)

Entre os entrevistados, 60% deram a nota máxima e os outros 40% deram nota 4 na questão “Recomendaria o software para outros usuários”, conforme pode ser visto no Gráfico 3 abaixo.

Gráfico 3 - Recomendação dos usuários



Fonte: Acervo do autor (2018)

A Tabela 3 mostra uma visão mais detalhada da pesquisa, com a avaliação dada por cada participante na pesquisa, sendo cada participante é representado pela letra “P” e um número correspondente.

Tabela 3 - Respostas dos participantes da pesquisa

	P1	**P2**	**P3**	**P4**	**P5**	**Média**
1) Leitura de caracteres na tela (Difícil - Fácil)	5	4	4	4	4	**4,2**
2) Disposição dos objetos na tela (Mal distribuídos – Bem distribuídos)	4	5	3	4	4	**4,0**
3) O total de informações na tela (Inadequado - adequado)	4	2	3	3	3	**3,0**
4) Ordem das informações na tela (Ilógica - Lógica)	4	3	4	4	4	**3,8**
5) A interface (tela de apresentação e de ações) do software (Desagradável - Agradável)	4	4	4	4	4	**4**
6) A quantidade de funções de avaliação (Insuficiente - Suficiente)	4	5	4	3	3	**3,8**
7) A quantidade de métodos de seleção (Insuficiente - Suficiente)	4	5	4	4	3	**4,0**
8) A quantidade de operadores de crossover (Insuficiente - Suficiente)	4	5	4	5	3	**4,2**
9) O software ajudou no processo de aprendizagem de Algoritmos Genéticos (Discordo - Concordo)	5	5	3	4	5	**4,4**
10) O software facilita a assimilação de conceitos dos Algoritmos Genéticos, tais como, avaliação, seleção, crossover e mutação (Discordo - Concordo)	4	5	3	5	5	**4,4**
11) É fácil de perceber quando o software terminou a simulação (Discordo - Concordo)	4	1	4	4	5	**3,6**
12) Nível de dificuldade para entender qual o caminho que deve ser seguido para ir de uma etapa para outra (Difícil - Fácil)	4	3	4	5	3	**3,8**
13) Os meios utilizados para apresentar as informações no software aumentaram a compreensão do conteúdo (Discordo - Concordo)	4	5	4	5	3	**4,2**
14) Depois de usar o software, o seu interesse	4	3	4	5	3	**3,8**

por Algoritmos Genéticos (Diminuiu - Aumentou)						
15) Recomendaria o software para outros usuários (Discordo - Concordo)	5	5	4	5	4	**4,6**
16) Nível de dificuldade de instalar/usar o software (Difícil - Fácil)	3	5	3	3	3	**3,4**
17) Nível de satisfação com o software (Insatisfeito - Satisfeito)	4	5	4	4	3	**4,0**
18) Você já estudou Algoritmos Genéticos antes de usar o software?	Não	Não	Sim	Não	Não	

Fonte: Acervo do autor (2018)

No geral, o GeneticsA recebeu pouca avaliações negativas, as 3 questões que obtiveram as menores médias foram: "Total de informações na tela" com 3,0, "Nível de dificuldade de instalar/usar o software" com nota 3,2 e "É fácil de perceber quando o software terminou a simulação" com avaliação de 3,4.

Já as 3 questões que obtiveram as maiores médias foram: "Recomendaria o software para outros usuários" com nota 4,6, "O software ajudou no processo de aprendizagem de Algoritmos Genéticos" com avaliação de 4,4 e "O software facilita a assimilação de conceitos dos Algoritmos Genéticos, tais como, avaliação, seleção, crossover e mutação" com nota 4,4. Os alunos que participaram da pesquisa tiveram uma semana para usar o GeneticsA, para depois responder o questionário.

5.2. TESTES

Os testes apresentados a seguir contemplam somente os resultados depois da execução do GeneticsA com diferentes funções de avaliação, métodos de seleção, operadores crossover e taxas de mutação. Neste trabalho não serão apresentadas todas as combinações possíveis dos parâmetros, os testes realizados servem apenas como exemplo de como a escolha dos parâmetros influenciam no desempenho do Algoritmo Genético em determinado problema. Em cada teste foi feita 5 execuções, com o limite de 50 gerações cada, cromossomos com 12 genes, população de 14 cromossomos e o problema de maximização da função $f(x) = x^2$.

5.2.1. Teste 1

O primeiro teste foi realizado com a função de avaliação simples, método de seleção Roleta viciada, operador de crossover de um ponto e taxa de mutação de 0,5%. Os resultados podem ser observados na Tabela 4 abaixo.

Tabela 4 - Resultados do teste 1

Execução nº	Encontrou a melhor solução?	Quantidade de gerações
1	não	50
2	não	50
3	não	50
4	não	50
5	não	50

Fonte: Acervo do autor (2018)

A melhor solução não foi encontrada em nenhuma das 5 execuções, ou seja, esta combinação de parâmetros não obteve um bom desempenho para o problema escolhido.

5.2.2. Teste 2

O segundo teste foi realizado com a função de avaliação Simples, método do torneio, operador de crossover de dois pontos e taxa de mutação de 1%. Os resultados podem ser observados na Tabela 5.

Tabela 5 - Resultados do teste 2

Execução nº	Encontrou a solução?	Quantidade de gerações
1	sim	10
2	sim	16
3	sim	12
4	sim	15
5	sim	25

Fonte: Acervo do autor (2018)

A solução foi encontrada em todas as 5 execuções, a primeira execução foi a que obteve o melhor desempenho, encontrando a solução depois de 10 gerações.

5.2.3. Teste 3

O terceiro teste foi realizado com a função de avaliação Normalização linear, método de seleção Roleta viciada, operador de crossover uniforme e taxa de mutação de 1,5%. Os resultados podem ser visto na Tabela 6 abaixo.

Tabela 6 - Resultados do teste 3

Execução nº	Encontrou a solução?	Quantidade de gerações
1	sim	16
2	sim	15
3	sim	6
4	sim	9
5	sim	22

Fonte: Acervo do autor (2018)

A solução foi encontrada em todos os casos, a terceira execução foi a que obteve o melhor desempenho, encontrando a solução depois de 6 gerações.

5.2.4. Teste 4

O quarto teste foi realizado com a função de avaliação Windowing, método de seleção Amostragem estocástica, operador de crossover uniforme e taxa de mutação de 3,0%. Os resultados podem ser analisados na Tabela 7.

Tabela 7 - Resultados do teste 4

Execução nº	Encontrou a solução?	Quantidade de gerações
1	sim	6
2	sim	26
3	não	50

4	sim	10
5	sim	22

Fonte: Acervo do autor (2018)

A solução foi encontrada em 4 dos 5 casos, a primeira execução foi a que obteve o melhor desempenho, encontrando a solução depois de 6 gerações.

5.2.5. Teste 5

O último teste foi realizado com a função de avaliação Escalonamento Sigma, método de seleção Roleta viciada, operador de crossover de um ponto e taxa de mutação de 0,1%. Os resultados podem ser observados na Tabela 8.

Tabela 8 - Resultados do teste 5

Execução nº	**Encontrou a solução?**	**Quantidade de gerações**
1	sim	8
2	não	50
3	não	50
4	não	50
5	sim	48

Fonte: Acervo do autor (2018)

A solução foi encontrada em 2 das 5 execuções, a primeira execução foi a que obteve o melhor desempenho, encontrando a solução depois de 8 gerações.

6. CONCLUSÃO

Os Algoritmos Genéticos são técnicas importantes, tanto que, são ensinadas em muitos cursos de Computação. Percebeu-se que existiam poucas ferramentas para auxiliar no ensino de AG, havendo uma escassez deste tipo de software. Os softwares educacionais são importantes instrumentos que podem ajudar o discente na construção do conhecimento. Portanto, foi proposto o desenvolvimento do GeneticsA, que é um software educacional que permite o discente acompanhar e entender o funcionamento das fases de um Algoritmo Genético.

O GeneticsA foi desenvolvido utilizando o JavaFX 8, que possui uma vasta biblioteca de componentes gráfico e possibilita a criação de interfaces (GUI) utilizando o padrão MVC, no qual a parte visual (View) é feita utilizando notação XML. Foi utilizado também o JavaFX Scene Builder, que é uma ferramenta bastante intuitiva, e auxilia na criação e manutenção do FXML.

A maior dificuldade do trabalho foi à criação da interface, houve uma grande preocupação em criar uma interface simples, que apresente somente às informações necessárias, para que o discente consiga entender cada fase do Algoritmo Genético da forma mais simples possível. O projeto foi executado dentro do cronograma e o objetivo geral do projeto foi atingido com sucesso, que era desenvolver um software de apoio ao ensino de Algoritmos Genéticos.

O GeneticsA é composto por 4 funções de avaliação (Simples, Normalização linear, Windowing e Escalonamento Sigma), 3 métodos de seleção (Roleta viciada, Método do torneio e Amostragem Estocástica), 3 operadores de crossover (Um ponto, Dois pontos e o operador Uniforme) e 2 problemas de maximização de funções. O discente também pode configurar parâmetros como taxa de mutação, número de genes, quantidade máxima de gerações e tamanho da população.

Com o intuito de testar a eficiência do GeneticsA no ensino de Algoritmos Genéticos, foi apresentado o software há um grupo de 7 alunos que fazem a disciplina de Inteligência Artificial no Instituto Federal Catarinense – Campus Rio do Sul. Deste grupo de 7 alunos, 5 deles responderam o questionário do Apêndice A. Os resultados obtidos na pesquisa foram positivos, em todas as questões a nota média foi maior ou igual 3, entre uma escala de 1 à 5. Outro aspecto positivo é que a questão que teve a melhor média na pesquisa foi “Recomendaria o software para outros usuários” com 4,6 de média, mostrando assim que o software teve uma certa aceitação. O GeneticsA, como qualquer software, tem aspectos que

podem serem melhorados, na subseção a seguir é apresentado algumas das melhorias possíveis no software.

6.1. TRABALHOS FUTUROS

Para trabalhos futuros, sugere-se:

- Aumentar a quantidade de problemas;
- Adicionar outras funções de avaliação;
- Adicionar mais métodos de seleção;
- Adicionar outros operadores crossover;
- Testar o software com uma amostra maior de alunos;
- Permitir que o usuário salve determinada execução, possibilitando assim a sua recuperação.

REFERÊNCIAS

BACK, Thomas; HOFFMEISTER, Frank; SCHWEFEL, Hans-Paul. A survey of evolution strategies. In: **Proceedings of the fourth international conference on genetic algorithms**. Morgan Kaufmann Publishers San Mateo, CA, 1991.

BARCELLOS, João Carlos Holland de. **Algoritmos Genéticos Adaptativos**: Um estudo comparativo. 2000. 143 f. Dissertação (Mestrado) - Curso de Engenharia, Escola Politécnica da Universidade de São Paulo, São Paulo, 2000.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML**: guia do usuário. 2. ed. Rio de Janeiro: Elsevier, 2012.

CENCI, Danielle; BONELLI, Sônia Maria de Souza. **Critérios para avaliação de softwares educacionais**. IX Anped Sul, Porto Alegre, 2012.

COLEY, David. **An Introduction to Genetic Algorithms for Scientists and Engineers**. S. L: World Scientific Publishing Company, 1999.

DARWIN, Charles. **On the Origin of Species**. John Murray, London, 1859.

DINIS, Rafael; SIMÕES, Anabela; BERNARDINO, Jorge. GraphEA: A 3D Educational Tool for Genetic Algorithms. **Proceedings Of The 15th Annual Conference Companion On Genetic And Evolutionary Computation**, New York, p.1293-1300, jul. 2013.

ECK, David. **About the Genetic Algorithms Demo**. s.d.. Disponível em: <http://math.hws.edu/eck/js/genetic-algorithm/ga-info.html>. Acesso em: 20 abr. 2018.

FREIRE, Paulo. **Educação como prática de liberdade**. Rio de Janeiro: Paz e Terra, 1967. 157 p.

FREIRE, Paulo. **Educação e mudança**. [S.l.]: Paz e Terra, [19--]. 46 p.

FREIRE, Paulo. **Pedagogia do oprimido**. Rio de Janeiro: Paz e Terra, 1987. 107 p.

FERREIRA, Naidson Clayr Santos. A informática no processo de ensino aprendizagem do Instituto Federal Baiano – Campus Guanambi. **Informática na Educação**: teoria & prática, Porto Alegre, v. 13, n. 1, p.140-155, jun. 2010.

GIRAFFA, L. M. M. Uma odisseia no ciberespaço: O software educacional dos tutoriais aos mundos virtuais. **Revista Brasileira de Informática na Educação**, v. 17, n. 01, p. 20, 2009.

GOLDBERG, David E.; DEB, Kalyanmoy. A comparative analysis of selection schemes used in genetic algorithms. In: **Foundations of genetic algorithms**. Elsevier, 1991. p. 69-93.

HAUPT, Randy L.; HAUPT, Sue Ellen. **Practical Genetic Algorithms**. 2. ed. Hoboken: John Wiley & Sons, 2004. 253 p.

HOLLAND, John Henry. **Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence**. MIT press, 1992.

HOMMEL, Scott. **Implementing JavaFX Best Practices**. 2014. Disponível em: <https://docs.oracle.com/javafx/2/best_practices/jfxpub-best_practices.pdf>. Acesso em: 05 out. 2018.

JUCA, Sandro César Silveira. A relevância dos softwares educativos na educação profissional. **Ciência & Cognição**, Fortaleza, v. 08, n. 22, p.22-28, ago. 2006. Disponível em: <http://pepsic.bvsalud.org/pdf/cc/v8/v8a04.pdf>. Acesso em: 05 mar. 2018.

KITA, Eisuke. **Evolutionary Algorithms**. Rijeka: Intech, 2011. 595 p.

KONRATH , Mary Lúcia P. and FALKEMBACH, Gilse Antoninha M. and TAROUCO, Liane M. R. A utilização de jogos em sala de aula: aprendendo Através de atividades digitais. **Novas tecnologias na educação**. Universidade Federal do Rio Grande do Sul. Rio Grande do Sul. v.3, n.1, p. 1-11, 2005.

KRAMER, Oliver. **Genetic Algorithm Essentials**. Oldenburg: Springer, 2017.

LEMOS, Heverton de; FERNANDES, Anita Maria da Rocha. Pesquisa e mapeamento de Shells, Frameworks e jogos para auxiliar no ensino de Inteligência Artificial. **Reabtic**, São José, v. 1, n. 2, dez. 2014. Disponível em: <https://revistas.setrem.com.br/index.php/reabtic/article/view/32>. Acesso em: 18 mar. 2018.

LIAO, Ying-hong; SUN, Chuen-tsai. **An Educational Genetic Algorithms Learning Tool**. 2001. Disponível em: <https://www.ewh.ieee.org/soc/es/May2001/14/Begin.htm>. Acesso em: 21 abr. 2018.

LINDEN, Ricardo. **Algoritmos Genéticos**: Uma importante ferramenta de Inteligência Computacional. 2. ed. Rio de Janeiro: Brasport, 2008.

LINDEN, Ricardo. **Algoritmos Genéticos**. 3. ed. Rio de Janeiro: Ciência Moderna, 2012. 475 p.

MALAQUIAS, Neli Gomes Lisboa. **Uso dos Algoritmos Genéticos para a Otimização de Rotas de Distribuição**. 2006. 113 f. Dissertação (Mestrado) - Curso de Engenharia Elétrica, Universidade Federal de Uberlândia, Uberlândia, 2006.

MACHADO, Marcelo Dornellas. **Algoritmo Evolucionário PBIL Multi-objetivo aplicado ao problema da recarga de reatores nucleares**. 2005. 127 f. Tese (Doutorado) - Curso de Ciências em Engenharia Nuclear, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2005.

MERCADO, Luís Paulo Leopoldo. **Novas tecnologias na educação**: reflexões sobre a prática. Maceió: Edufal, 2002. 206 p.

MITCHELL, Melanie. **An Introduction to Genetic Algorithms**. London: Massachusetts Institute Of Technology, 1996.

MORATORI, Patrick Barbosa. **Por que utilizar jogos educativos no processo de ensino aprendizagem?** 2003. 33 f. Dissertação (Mestrado) - Curso de Informática Aplicada à Educação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2003. Disponível em: <http://www.nce.ufrj.br/GINAPE/publicacoes/trabalhos/t_2003/t_2003_patrick_barbosa_moratori.pdf>. Acesso em: 04 abr. 2018.

ORACLE. **Informações gerais sobre o JavaFX**. Disponível em: <https://www.java.com/pt_BR/download/faq/javafx.xml>. Acesso em: 05 out. 2018.

ORACLE. **JavaFX**: Introdução ao JavaFX. 2014. Disponível em: < https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#BABGBBDH>. Acesso em: 05 out. 2018.

ORACLE. **Using JavaFX Scene Builder with Java IDEs**. 2013. Disponível em: <https://docs.oracle.com/javafx/scenebuilder/1/use_java_ides/jsbpub-use_java_ides.htm>. Acesso em: 05 out. 2018.

OSTERMANN, Fernanda; CAVALCANTI, Cláudio José de Holanda. **Teorias de aprendizagem**. Porto Alegre: Evangraf, 2011. Disponível em: <http://www.ufrgs.br/sead/servicos-ead/publicacoes-1/pdf/Teorias_de_Aprendizagem.pdf>. Acesso em: 05 abr. 2018.

PACHECO, Marco Aurélio Cavalcanti. Algoritmos Genéticos: princípios e aplicações. **Laboratório de Inteligência Computacional Aplicada**. Rio de Janeiro, jul. 1999.

PARREIRAS, Roberta Oliveira. **Algoritmos Evolucionários e Técnicas de Tomada de Decisão em Análise Multicritério**. 2006. 166 f. Tese (Doutorado) - Curso de Engenharia Elétrica, Universidade Federal de Minas Gerais, Belo Horizonte, 2006.

PIAGET, Jean. **O nascimento da Inteligência na Criança**. mental, v. 258, p. 259, 1970.

PRESSMAN, Roger S. **Engenharia de software**: uma abordagem profissional. 7. ed. Porto Alegre : AMGH, 2011.

RENNARD, Jean Philippe. **Genetic Algorithm Viewer**: Demonstration of a Genetic Algorithm. 2000. Disponível em: <https://www.rennard.org/alife/english/gavgb.pdf>. Acesso em: 20 abr. 2018.

REZENDE, C. d. S. **Modelo de avaliação de qualidade de software educacional para o ensino de ciências**. Dissertação, 2013.

RUSSELL, Stuart; NORVIG, Peter. **Inteligência artificial**. 3. ed. Rio de Janeiro: Elsevier, 2013. 1016 p.

SOMMERVILLE, Ian. **Engenharia de Software**. 8. ed. São Paulo: Pearson, 2007.

SOARES, Gustavo Luis. **Algoritmo Genético**: Estudo, novas técnicas e aplicações. 1997. 148 f. Dissertação (Mestrado) - Curso de Engenharia Elétrica, Universidade Federal de Minas Gerais, Belo Horizonte, 1997.

VALENTE, José Armando. Diferentes usos do computador na educação. **Em Aberto**, v. 12, n. 57, 2008.

VALENTE, José Armando. **O uso inteligente do computador na educação**. Artes Médicas Sul, Porto Alegre, v. 1, n. 1, p.19-21, maio 1997.

VALENTE, José Armando et al. O computador na sociedade do conhecimento. **Campinas: Unicamp/NIED**, v. 6, 1999.

VIEIRA, Fábia Magali Santos. Avaliação de software educativo: reflexões para uma análise criteriosa. **Campinas: EDUTECNET**, 1999.

VYGOTSKI, Lev Semenovich. **A formação social da mente**. 4. ed. São Paulo: Martins Fontes, 1991. 90 p. Disponível em: <http://egov.ufsc.br/portal/sites/default/files/vygotsky-a-formac3a7c3a3o-social-da-mente.pdf>. Acesso em: 05 abr. 2018.

APÊNDICE A– QUESTIONÁRIO

Questionário de Satisfação de Uso

Avalie sua satisfação com o GeneticsA, demonstrando sua percepção em relação ao itens abaixo. Marque o número mais apropriado que reflete suas impressões no uso do GeneticsA.

1. **Leitura de caracteres na tela.**

	1	2	3	4	5	
Difícil	◯	◯	◯	◯	◯	Fácil

2. **Disposição dos objetos na tela.**

	1	2	3	4	5	
Mal distribuídos	◯	◯	◯	◯	◯	Bem distribuídos

3. **O total de informações na tela.**

	1	2	3	4	5	
Inadequado	◯	◯	◯	◯	◯	Adequado

4. **Ordem das informações na tela.**

	1	2	3	4	5	
Ilógica	◯	◯	◯	◯	◯	Lógica

5. **A interface (tela de apresentação e de ações) do software.**

	1	2	3	4	5	
Desagradável	◯	◯	◯	◯	◯	Agradável

6. **A quantidade de funções de avaliação.**

	1	2	3	4	5	
Insuficiente	◯	◯	◯	◯	◯	Suficiente

7. **A quantidade de método de seleção.**

	1	2	3	4	5	
Insuficiente	◯	◯	◯	◯	◯	Suficiente

8. **A quantidade de operadores de crossover.**

	1	2	3	4	5	
Insuficiente	◯	◯	◯	◯	◯	Suficiente

9. **O software ajudou no processo de aprendizagem de Algoritmos Genéticos.**

	1	2	3	4	5	
Discordo	◯	◯	◯	◯	◯	Concordo

10. **O software facilita a assimilação de conceitos dos Algoritmos Genéticos, tais como, avaliação, seleção, crossover e mutação.**

	1	2	3	4	5	
Discordo	◯	◯	◯	◯	◯	Concordo

11. **É fácil de perceber quando o software terminou a simulação.**

	1	2	3	4	5	
Discordo	◯	◯	◯	◯	◯	Concordo

12. **Nível de dificuldade para entender qual o caminho que deve ser seguido para ir de uma etapa para outra.**

	1	2	3	4	5	
Difícil	◯	◯	◯	◯	◯	Fácil

13. **Os meios utilizados para apresentar as informações no software aumentam a compreensão do conteúdo.**

	1	2	3	4	5	
Discordo	◯	◯	◯	◯	◯	Concordo

14. **Depois de usar o software, o seu interesse por Algoritmo Genéticos.**

	1	2	3	4	5	
Diminuiu	◯	◯	◯	◯	◯	Aumentou

15. **Recomendaria o software para outros usuários.**

	1	2	3	4	5	
Discordo	◯	◯	◯	◯	◯	Concordo

16. **Nível de dificuldade de instalar/usar o software.**

	1	2	3	4	5	
Difícil	◯	◯	◯	◯	◯	Fácil

17. **Nível de satisfação com o software.**

	1	2	3	4	5	
Insatisfeito	◯	◯	◯	◯	◯	Satisfeito

18. **Você já estudou Algoritmo Genéticos antes de usar o software?**

- ◯ Sim
- ◯ Não

19. **Comentários (indique o número do item que deseja comentar na frente do comentário)**

_____

_____

_____

_____

_____

_____