UNIVERSIDADE PAULISTA

BACHALERADO EM CIÊNCIA DA COMPUTAÇÃO

FELIPHE STIVAL V. GUILIANI

GABRIEL RIBEIRO

PROJETO CALCULADORA GENETICA UTILIZANDO ALGORITMOS GENÉTICOS

GOIÂNIA - GO

2021

FELIPHE STIVAL V. GUILIANI

GABRIEL RIBEIRO

PROJETO CALCULADORA GENETICA UTILIZANDO ALGORITMOS GENÉTICOS

Trabalho de conclusão de curso para obtenção do título de graduação em Ciência da computação apresentado à Universidade Paulista – UNIP.

Orientador(a): Prof. Me. Nelson Batista Leitão Neto

GOIÂNIA - GO

2021

**AGRADECIMENTOS**

Agradecemos aos familiares, aos professores e todos aqueles que deram suporte e ajuda direta e indiretamente para concluirmos este trabalho.

“Uma mente precisa de livros como uma espada precisa de uma pedra de amolar para mantê-la afiada. Por isso eu leio tanto.”

(Tyrion Lannister – Game of Thrones)

**Sumário**

[1. INTRODUÇÃO 3](#_Toc85990180)

[**1.1. Objetivo** 5](#_Toc85990181)

[2. ALGORITMOS 6](#_Toc85990182)

[**2.1 Linguagens de programação** 7](#_Toc85990183)

[**2.2 Paradigmas de programação** 7](#_Toc85990184)

[**2.3 Programação estruturada** 7](#_Toc85990185)

[**2.4 Programação orientada a objetos** 8](#_Toc85990186)

[**2.5 Vantagens da orientação a objetos** 9](#_Toc85990187)

[**2.6 Problemas em algoritmos** 9](#_Toc85990188)

[2.6.1. Redução 10](#_Toc85990189)

[2.6.2. Boas práticas de programação 10](#_Toc85990190)

[2.6.3. Conceito DRY (DON’T REPEAT YOURSELF) 11](#_Toc85990191)

[3. MACHINE LEARNING 12](#_Toc85990192)

[**3.1 Algoritmo genético** 12](#_Toc85990193)

[3.1.1 Representação cromossomial 13](#_Toc85990194)

[3.1.2. Codificação binária 13](#_Toc85990195)

[3.1.3 Função de avaliação 14](#_Toc85990196)

[3.1.4 Seleção dos pais 15](#_Toc85990197)

[3.1.5 Método da roleta 16](#_Toc85990198)

[3.1.6 Crossover 18](#_Toc85990199)

[3.1.7 Mutação 18](#_Toc85990200)

[4. BANCO DE DADOS 20](#_Toc85990201)

[**4.1. Modelo hierárquico** 20](#_Toc85990202)

[**4.2. Modelo em rede** 21](#_Toc85990203)

[**4.3. Modelo relacional** 21](#_Toc85990204)

[**4.4. Banco de dados não relacionais** 22](#_Toc85990205)

[**4.5. PostgreSQL** 22](#_Toc85990206)

[5. DESENVOLVIMENTO 23](#_Toc85990207)

[**5.1 Fluxo de usabilidade do sistema** 23](#_Toc85990208)

[**5.2 Tecnologias e ferramentas utilizadas** 24](#_Toc85990209)

[5.2.1 R 24](#_Toc85990210)

[5.2.2 Ionic Framework 25](#_Toc85990211)

[6. CONSIDERAÇÕES FINAIS 26](#_Toc85990212)

[7. REFERÊNCIAS BIBLIOGRÁFICAS 27](#_Toc85990213)

# INTRODUÇÃO

O melhoramento genético de plantas vem sendo feito pelo ser humano há milhares de anos para torná-las mais produtivas, nutritivas e adequadas ao cultivo, de acordo com a necessidade humana. A genética de uma planta modificada é tão distinta de seu progenitor selvagem, que uma espécie que é totalmente domesticada se torna quase que completamente dependente do homem para sua sobrevivência.

As pesquisas na área de melhoramento de plantas e animais não são novas, as primeiras pesquisas propostas foram feitas por Gregor Johann Mendel em 1866. Com sua pesquisa houve um grande avanço na área de melhoramento vegetal. A partir daí, nas primeiras décadas do século XX, se estabeleceu que a informação genética se encontra contida nos cromossomos.

No ano de 1953 James Watson e Francis Crick, com ajuda do trabalho de Maurice Wilkins e Rosalind Franklin, propuseram um modelo para a estrutura de dupla hélice do DNA. Em 1961, Francois Jacob e Jacques Monod desvendaram a síntese de proteínas do DNA, que passou, então, a ser o elemento central das pesquisas de engenharia genética. Porém com todos esses avanços a primeira planta melhorada com técnicas de engenharia genética atuais só foram surgir no ano de 1972.

No ano de 1972 na Universidade de Stanford (Estados Unidos), Paul Berg e seus colegas geraram in vitro (fora do organismo vivo) a primeira molécula de DNA recombinada pela ligação de duas cadeias de DNA. Com esse estudo se deu início a era da manipulação genética, já no ano seguinte foram criadas as primeiras plantas modificadas com técnicas de engenharia desenvolvidas até então.

O processo de geração de melhoramento de uma planta é longo e depende de vários fatores, tais como genótipo (composição genética de um indivíduo), características introduzidas, tecnologias de transformação genética disponíveis, dentre outros. Para realizar o melhoramento de uma planta é necessário ter conhecimento básico bioquímico e de biológica molecular dos fatores envolvidos com a característica. Por fim, para que qualquer nova variedade possa chegar ao processo produtivo, são necessárias a avaliação de biossegurança rigorosa. Com a evolução dos estudos na área em conjunto com os avanços na área de inteligência artificial, o processo de melhoramento vegetal se tornou mais rápido e barato.

Os primeiros estudos na área de inteligência artificial surgiram no ano de 1943, quando Warren McCulloch e Walter Pitts publicaram um artigo sobre redes neurais e um modelo matemático que tinha como objetivo simular o sistema nervoso humano.

Alguns anos após a publicação, Claude Shannon também publicou outro artigo com o objetivo de mostrar como ensinar uma máquina a jogar xadrez utilizando cálculos de posição. Ambos dos artigos foram bastantes importantes para se iniciar os estudos na área de inteligência artificial. Existem diversas áreas da inteligência artificial, a que vamos utilizar nesse trabalho, como a área de busca e otimização utilizando algoritmos genéticos.

Os algoritmos genéticos são técnicas heurísticas de otimização global. A questão de otimização global opõe os G.A. (Genetic Algorithm) aos métodos como gradiente (hill climbing) que seguem a derivada de uma função de forma a encontrar o máximo de uma função, ficando facilmente retidos em máximos locais.

Diferentes dos algoritmos de hill climbing, os algoritmos genéticos não ficam estagnados por ter encontrado um máximo local. Nesse quesito os algoritmos genéticos imitam a evolução natural que mesmo que encontre um bom indivíduo não para de procurar. Devemos lembrar que G.A. (Genetic Algorithms), não são algoritmos de busca de solução ótima (solução que apresenta melhor valor) de um problema, mas sim uma busca heurística que encontra boas soluções a cada execução.

Basicamente, o que um algoritmo genético faz é criar uma população inicial e depois submetê-la as seguintes etapas para simular uma evolução.

* A primeira etapa é criar um conjunto de indivíduos que representam uma possível solução para o nosso problema.
* Após a criação dos indivíduos é feita a avalição dos mesmos para avaliar toda a geração de cromossomos,.
* Em seguida é realizada a etapa de cruzamento que é dividida em três rotinas sendo elas acasalamento, recombinação e mutação.
* Por fim, na última etapa, temos a escolha de dois cromossomos para se reproduzirem.

O processo de recombinação é um operador para causar variação de um cromossomo, após esse processo é realizado o processo de mutação que tem como objetivo aumentar a variabilidade genética da população de cromossomos, impedindo que o algoritmo fique estagnado em um determinado ponto.

## **1.1. Objetivo**

O programa desenvolvido nesse trabalho tem como principal objetivo auxiliar profissionais melhoristas, no aumento da produção agrícola e na resistência a doenças e pragas utilizando melhoramento genético.

No programa, o usuário poderá cadastrar genótipos, informando a lista de fenótipos (manifestação visível ou detectável de um genótipo), para que com essas informações, um algoritmo genético possa otimizar as melhores combinações de cruzamento de acordo com características definidas e nomeadas pelo usuário.

O cruzamento será realizado utilizando 2º lei de Mendel, conhecida como lei de segregação independente, onde é possível cruzar várias características (fenótipos) ao mesmo tempo. Logo em seguida um algoritmo genético (AG), que é uma técnica de busca utilizada na ciência da computação para achar soluções aproximadas em problemas de otimização e busca, fará a escolha das melhores combinações a serem apresentadas para o usuário.

# ALGORITMOS

Um programa ou algoritmo de computador, pode ser entendido como uma sequência de instruções que determinam o fluxo de impulsos elétricos dentro de um sistema de computador. Tais impulsos afetam diretamente a memória do computador, interagindo com o hardware, ou seja, a tela do monitor, teclado, mouse, etc. No nível inferior, mais concreto, os impulsos elétricos alteram o estado interno do computador, enquanto no nível superior, mais abstrato, os usuários de computador realizam trabalho no mundo real ou obtêm prazer real. As possibilidades são infinitas. Um algoritmo pode assumir o papel de uma calculadora financeira, um simples gerenciamento de dados e até mesmo um forte oponente de xadrez.

Um programa de computador é um exemplo de software de computador. A lógica e a matemática compõem completamente os conceitos por trás dos softwares. Portanto, levando em consideração um lado mais teórico, é possível de fato desenvolver programas sem a utilização de um computador.

Os programadores podem elaborar toda estrutura de um programa e raciocinar eficiência examinando somente símbolos abstratos correspondentes a linguagens de programação reais. O software torna o computador uma máquina verdadeiramente universal, transformando-o na ferramenta adequada para a tarefa em questão. O software deve ser armazenado na memória do computador, normalmente, os programas de computador são carregados na memória a partir de um hardware, como o disco rígido do computador.

Um padrão eletromagnético que representa o programa é armazenado no disco rígido do computador. Este padrão de símbolos eletrônicos deve ser transferido para a memória do computador antes que o programa possa ser executado. A sequência de símbolos, ou seja, o software pode ser transferido para o computador de diversas formas, tendo como mais comum o download via internet.

“As linguagens de programação servem para escrever programas que permitem a comunicação entre usuário e máquina. Programas especiais chamados tradutores (compiladores ou interpretadores) convertem as instruções escritas em linguagens de programação em instruções escritas em linguagens de máquina (0 e 1, bits) que a máquina pode entender. (Algoritmos e Ferramentas de Programação, p. 19)”

## **2.1 Linguagens de programação**

Linguagens de programação, são meios de comunicação direta com o computador. Existem linguagens de programação de alto e baixo nível, sendo respectivamente, linguagens mais fáceis de serem assimiladas por seres humanos e linguagens mais abstratas, semelhantes ao código de máquinas e a linguagem binária, zero e um.

## **2.2 Paradigmas de programação**

Ao longo dos anos de evolução da engenharia de software, cada vez mais se tornava necessária a elaboração de paradigmas de programação, que pudessem suprir de forma eficiente e simples, as necessidades de programadores no desenvolvimento de soluções de alto nível, ou seja, programas mais complexos e que exigem uma arquitetura perfeitamente elaborada. Paradigma de programação consiste basicamente em padronizar o estilo de escrita do código, obtendo um controle total na estrutura e execução do projeto. De acordo com o livro, *Linguagens de Programação - Princípios e Paradigmas - 2ª Ed.*, um paradigma pode ser definido como "um padrão de pensamentos que guia um conjunto de atividades relacionadas". Portanto, é uma forma amigável e mais simples de resolver problemas diversos em programas e linguagens de programação.

A partir desse conceito, tornou-se comum em linguagens de programação, a utilização dos mais diversos paradigmas, tais como, Linguagem de Programação Orientada a Objetos, Linguagem de Programação Estruturada, dentre outros. Atualmente, várias linguagens de programação são desenvolvidas com o suporte a vários paradigmas diferentes. Um ótimo exemplo de linguagem multiparadigma, é o famoso C++, que é uma linguagem imperativa e orientado a objetos. Também podemos citar a linguagem de programação Leda, que suporta em seu escopo paradigmas de programação imperativa, orientada a objetos, funcional e lógica.

## **2.3 Programação estruturada**

Para se compreender a necessidade e surgimento da programação estruturada, é necessário primeiro compreender alguns conceitos da estrutura de controle um tanto quanto polêmica chamada, Comando GoTo. Em uma tradução literal, ‘Go To’ significa ‘Ir Para’, portando pode se definir como uma estrutura para saltar instruções. Seu funcionamento é simples, o comando determina um rótulo ou um número representando um endereço qualquer. Em seguida, após a execução, todos as instruções são apontadas para o endereço de destino.

Os primeiros indícios de programação estruturada surgiram no final da década de 1960, impulsionados pelo desenvolvimento das linguagens de programação C e Pascal. Com uma abordagem completamente diferente de linguagens como Fortran 66 as linguagens novas tinham implementadas em seu escopo estruturas de laços e condicionais, como laços for e while, utilizados em estruturas de repetições, e também, condicionais if-else e switch-case, utilizados em tomadas de decisões.

Portanto, a programação estruturada vulgarizava qualquer utilização do comando GoTo, julgando desnecessário e incompreensível. A partir daí, algumas linguagens como Java e Ada depreciaram o comando, tornando notável que a evolução para programação estruturada foi um sucesso completo, sendo utilizada constantemente nos dias de hoje.

## 

## **2.4 Programação orientada a objetos**

A abordagem de Programação Orientada a Objetos, POO, consiste principalmente, em tratar o programa como o mundo real, ou seja, abstrair processos internos em objetos, representados por classes e que interagem entre si. Tal conceito, sugere uma criação de mecanismos modularizados, o que reduz a complexidade e aumenta a organização do sistema como um todo. Cada parte, encapsula a complexidade de seu funcionamento, permitindo que modificações internas não afetem outras partes do sistema.

Este paradigma tem como um de seus muitos benefícios, facilitar e acelerar tanto o desenvolvimento, quanto a manutenção futura do sistema, que quando mal implementado, se torna muito cara e muitas das vezes inviável.

Atualmente, as principais linguagens de programação do mercado possuem suporte ao paradigma orientado a objetos, e é notória a satisfação por parte de desenvolvedores. Podemos citar de exemplo linguagens como Java, que possui todo seu escopo e código fonte estruturado com os conceitos de orientação a objetos. Também podemos citar o C++, que mesmo sendo uma linguagem mais primitiva, tendo mais liberdade em relação ao Java, oferece suporte completo a orientação a objetos, compondo o pódio de representantes deste, que é um dos mais famosos paradigmas de programação.

## 

## **2.5 Vantagens da orientação a objetos**

A orientação a objetos tem inúmeras vantagens, algumas delas são, maior produtividade, as regras de orientação a objetos impostas ao programador, o ajudam a realizar um maior número de tarefas e um curto prazo. Programas que não seguem os padrões de POO, tendem a ser grandes e contínuos, o que torna seu código desorganizado e sua manutenção mais complexa. Também devemos lembrar sobre o baixo custo, principalmente em grandes projetos, os custos de mão de obra tendem a ser reduzidos, pois a equipe de programação pode desenvolver suas próprias bibliotecas de forma fácil, permitindo que outros programadores utilizem diversas vezes ao longo do projeto, reduzindo o tempo de desenvolvimento. Tal paradigma também tem suas desvantagens em vários aspectos, tais como curva de aprendizagem difícil, isso acontece pois os conceitos de aprendizagem são aplicados reutilizando objetos, ou seja, incrementando-o com novos recursos. Tais conceitos levam tempo e podem ser complexos demais para iniciantes durante a aprendizagem. Outra desvantagem da orientação a objetos é o tempo de resposta dos programas, que tendem a ser reduzidos, o que se torna um pesadelo para projetos que exigem velocidade de seus programas.

Qualquer linguagem, paradigma ou tecnologia em um geral, terá suas vantagens e desvantagens. Cabe ao gestor do projeto escolher a que melhor atende a cada caso. Esta é uma tarefa muito importante e indiscutivelmente necessária, fazendo parte de todo o processo de desenvolvimento.

## **2.6 Problemas em algoritmos**

Mesmo com toda evolução magnífica da computação, ainda nos dias de hoje existem inúmeras questões que não podem ser resolvidas no universo computacional. Desde os primeiros algoritmos, discussões são levantados pela comunidade tentando responder a famosa pergunta: “Um computador consegue resolver qualquer problema?”. A resposta é mais simples que a pergunta, não. Existem problemas de uma complexidade tão grande que se torna impossível a resolução por parte de um algoritmo computacional, mesmo com tempo e processamento ilimitado.

O primeiro problema computacional que foi comprovadamente considerado insolucionável foi o *Problema da Parada*, apresentado por Alan Turing em seus estudos. O problema sugere o seguinte. Dado um programa e uma entrada, se o programa irá eventualmente parar quando executado com aquela entrada ou se executará para sempre.

### 2.6.1. Redução

A redução tem seu papel considerado muito importante nos problemas em algoritmos. Por exemplo, ao desenvolver bibliotecas distintas para resolver determinados problemas, reduzimos novos problemas em apenas um, pois boa parte já foi solucionada e pode ser reutilizada, economizando muito tempo.

Considere situação em que é necessário desenvolver uma solução de estouro de memória durante a execução do programa. Caso essa solução seja desenvolvida de forma reduzida, será possível reutilizar essa mesma solução para todas as partes do programa que causam o problema citado.

### 2.6.2. Boas práticas de programação

No mundo da tecnologia de informação, não basta apenas aprender codificar. É preciso desenvolver um código de qualidade, utilizando todas as boas práticas sugeridas pelas especificações da linguagem de programação escolhida. Um código bem escrito, bem estruturado e planejado, tem como consequência um programa de fácil manutenção e com menos possibilidades de bugs.

Por tanto, é indiscutível que as boas práticas devem fazer parte do processo de desenvolvimento de qualquer programa, desde o mais simples website, até um sistema mais complexo. As boas práticas se estendem desde simples nomenclaturas de variáveis, até estruturação de classes. É preciso seguir padrões elegantes na sintaxe léxica de seu código. Essas padronizações fazem toda a diferença no bom crescimento de um software. Variáveis devem ser nomeadas seguindo um padrão “camelCase”, por exemplo, uma variável que armazena o nome de um usuário pode ser nomeada como, nomeUsuário. Constantes devem ser nomeadas seguindo um padrão UPPERCASE, com todas as letras em caixa alta. Funções devem ser nomeadas seguindo o padrão underscores, por exemplo, uma função soma de dois números poderia ser nomeado como, soma\_dois\_numeros. Classes devem ser nomeadas quase que da mesma forma que as variáveis, porém com primeira letra em caixa alta, por exemplo, uma classe de conta pessoal de banco poderia ser nomeada como, ContaPessoal.

Outra boa prática muito comum e indispensável no desenvolvimento de um bom algoritmo são os comentários. Escrever bons comentários mantém o equilíbrio e acaba se tornando uma arte, pois além de guiar e facilitar a jornada de outros programadores, documenta toda a lógica empregada no código fonte. Porém, não se pode exagerar, comentários precisam ser sucintos e diretos, devem ser utilizados com moderação para que o código se torne óbvio aos olhos de outros. Alguns dos lugares onde são indispensáveis os comentários são no início do código, detalhando o arquivo em questão sobre as funções as precedendo e também nos trechos de códigos não tão óbvios.

### 2.6.3. Conceito DRY (DON’T REPEAT YOURSELF)

Em uma tradução literal, “Não se repita!”. A frase diz tudo, é extremamente importante evitar ao máximo trechos de códigos repetidos. A reutilização correta de código é um dos principais tópicos das boas práticas da programação.

A reutilização de código deixa o algoritmo mais limpo e organizado, facilitando mais uma vez, a manutenção futura do código. Essa ideia surgia primeiro em outras áreas, como, gestão de dados e base de dados, mas ainda assim, se encaixa perfeitamente ao desenvolvimento de código.

Portanto, trechos de códigos que tendem a se repetir, mesmo que pouco, devem ser desenvolvidos de forma genérica, podendo ser chamada, inúmeras vezes ao longo do código. Como consequência, o código será reduzido juntamente com os erros.

# MACHINE LEARNING

Machine Learning (em português, Aprendizado de Máquina), é uma característica de um dispositivo tecnológico que possui Inteligência Artificial (IA). Inteligência Artificial pode ser categorizada como geral ou estreita. IA geral apresenta características de inteligência humana e capacidades de planejamento, reconhecimento de objetos, sons, aprendizado e resolução de problemas.

IA estreita são os dispositivos que apresentam alguma inteligência na área de atuação para que foram desenvolvidos, porém em outras áreas apresentam deficiência. Nesse ponto entra o aprendizado de máquina, que tem como objetivo tornar a inteligência do dispositivo ou objeto tecnológico o mais humano possível, seja através da comunicação ou da tomada de decisão e resolução de problemas. Como exemplo, podemos considerar o aprendizado de máquina através da observação de imagens ou vídeos. O ser humano poderia analisar centenas de imagens que tem um cachorro ou não e saberia diferenciar cada uma delas afirmando qual imagem é de um cachorro e qual não é.

Nesse mesmo caso, um algoritmo pode construir um modelo que pode indicar com precisão uma imagem que contém um cachorro ou não. Se o modelo for bem preciso, a máquina aprende como é um cachorro.

## **3.1** **Algoritmo genético**

Algoritmos genéticos começaram a ser estudados no início da década de 40, quando os cientistas tentaram introduzir a evolução natural no ramo da inteligência artificial.

Os algoritmos genéticos são técnicas heurísticas de otimização global, ao contrário dos outros algoritmos de hill climbing, o algoritmo genético não fica estagnado simplesmente por ter encontrado um local máximo. Nesse quesito ele se parece com a evolução natural que mesmo encontrando o melhor individuo ele vai continuar procurando outros indivíduos melhores.

Nos algoritmos genéticos as populações de indivíduos são criadas e avaliadas pelos operadores genéticos: seleção, recombinação(crossover) e mutação. O algoritmo genético de acordo com LIDEN (2006) pode ser resumido da seguinte forma.

“Primeiro se inicia a população de cromossomos, após a inicialização dos indivíduos é feita a avaliação de cada cromossomo de população gerada, na próxima etapa é feita a seleção dos pais para gerar um novo cromossomo, na última etapa são aplicados os operados de recombinação e mutação aos pais para encontrar novos indivíduos para a nova geração. Após isso é feita uma nova avaliação de todos os indivíduos para colocá-los na nova geração.” (LIDEN.2006,p.26) .”

### 3.1.1 Representação cromossomial

A representação cromossomial é essencial na implementação de um algoritmo genético, ela consiste em traduzir o nosso problema em uma maneira que o computador possa entender, essa parte é de extrema importância na implementação de um algoritmo genético, pois quanto mais ela for adequada ao nosso problema, maior a qualidade dos resultados.

Cada pedaço de um cromossomo é chamado de gene, por analogia com as partes fundamentais que compõe um cromossomo biológico. É importante salientar que a representação de um cromossomo é arbitraria, ficando a sua definição de acordo com a implementação do programador. Porém de acordo com LIDEN (2006) seria interessante se a representação tivesse as seguintes práticas:

“A representação deve ser a mais simples possível, caso houver soluções proibidas ao problema, então elas não devem ter uma representação caso o problema impuser condições de algum tipo, estas devem estar implícitas dentro da nossa representação.” (LINDEN, 2006, p. 59).”

Mesmo existindo diversas formas que um cromossomo pode ser representado, a mais comum e utilizada é a forma binária, nessa forma o cromossomo nada mais é que uma sequência de bytes.

Essa representação foi criada em 1975 por Holland em seu livro (Holland,1975), devido a sua simplicidade de implementação e a sua facilidade de uso, ela é bastante utilizada por pesquisadores da área de algoritmos genéticos.

### 3.1.2. Codificação binária

“A codificação binaria é a forma mais comum de se representar um espaço de um cromossomo. Nesse tipo de codificação os dados são armazenados em cadeias de bits. Em grande parte dos problemas práticos tratados com Algoritmo genético é utilizada a codificação binária (LOPES, 2006).”

Essa representação foi usada a primeira vez por Holland em seu livro seminal (Holland,1975) e, hoje em dia, por estes motivos históricos e pelo fato de ser muito fácil a sua implementação e utilização, ela é amplamente adotada nas implementações de algoritmos genéticos.

De acordo com (TANOMARU,1995), um indivíduo qualquer, em uma codificação binária posicional, é representado por uma sequência s = [bn....b2b1], onde n é o número de bits necessários para representar x e cada bi ∈ 0,1.

Quando a codificação binaria é utilizada para representar valores reais, o tamanho do cromossomo dependerá da precisão requerida. Sendo *t* o tamanho do intervalo real da variável e p a precisão requerida. O tamanho do cromossomo necessário para representar essa variável pode ser obtido por meio da seguinte formula.

Depois que o tamanho do cromossomo é obtido, é necessário mapear o intervalo real da variável no intervalo binário. Este processo deve ser realizado em duas etapas: o primeiro passo é obter o valor de mapeamento sobre o valor real da variável e depois converter o resultado para a base binária. O mapeamento pode ser obtido por meio da seguinte formula.

### 3.1.3 Função de avaliação

A função de avaliação tem como objetivo determinar o quanto o indivíduo está próximo de ser a solução do problema proposto. Podemos olhar a função de avaliação como sendo a única ligação verdadeira do programa com o problema real.

Na implementação de uma função de avaliação, o programador deve se atentar ao fato de que ela deve conter todo o conhecimento que se possui sobre o problema a ser resolvido, tanto em suas restrições quantos aos objetivos de qualidade.

Um ponto que deve ser lembrado na hora da implementação de uma função de avaliação é que nenhum elemento deve ter a avaliação negativa ou zero, essa prática faria com que a soma das avaliações diminuísse fazendo com que a roleta possa selecionar mais de um valor em um determinado intervalo.

### 3.1.4 Seleção dos pais

Existem vários métodos para se implementar a seleção em um algoritmo genético, porém não há uma regra especifica definindo qual método deve ser utilizado em cada tipo de problema, isto ainda é uma questão aberta em A.G. (Algoritmo Genético), (MITCHELL, 1996).

O primeiro método criado de seleção foi desenvolvido por Holland em 1975. Esse método era baseado na proporção do valor de fitness do indivíduo em relação a soma do fitness de toda a população. A probabilidade que um indivíduo tinha de ser escolhido é dada pela seguinte fórmula

O método de seleção é uma das etapas do algoritmo genético, de acordo com (TANOMARU,1995) se não houvesse o processo de seleção, além do A.G. (Algoritmo Genético) perder grande parte do caráter evolutivo, o mesmo seria um processo ineficiente similar a uma busca aleatória.

O método da seleção dos pais, tenta replicar o método presente na natureza, em que os pais que são mais adaptados geram mais filhos, porém os que são menos adaptados também devem gerar descendentes.

Nesse método devemos privilegiar os indivíduos que possuem as melhoras notas da função de avaliação, mas também devemos garantir que os indivíduos com notas baixas também sejam selecionados.

O objetivo de selecionar os indivíduos com baixa nota é que mesmo eles tendo uma baixa avaliação, eles podem conter genes que possam favorecer à criação de um “super individuo”, que pode ser entendido como a melhor solução encontrada do problema. No exemplo abaixo temos uma tabela com as informações de uma simulação.

**Figura 2** – Informações de uma simulação

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Fonte: LINDEN, R. Algoritmos genéticos teoria e implementação.

No exemplo utilizado, a função de avaliação corresponde ao quadrado do número do valor do indivíduo binário, podemos notar que o indivíduo ‘0001’ tem uma péssima avaliação, já que seu valor ao quadrado é apenas 1. Mas ele possui uma boa característica, que é o na última posição, que não está presente nos dois melhores indivíduos. Se na simulação fosse utilizado somente os dois melhores indivíduos para produzir a nova população o algoritmo nunca iria avançar para um indivíduo melhor que o ‘0110’.

A maneira que a grande maioria dos pesquisadores de G.A. (Algoritmo Genético) utiliza para dar chance a todos os indivíduos é o método da roleta viciada.

### 3.1.5 Método da roleta

O método da roleta é o mais utilizado por ser o mais simples, neste método criamos uma roleta virtual onde cada indivíduo recebe um pedaço proporcional a sua avaliação. Depois rodamos a roleta e será selecionado o indivíduo sobre o qual ela parar.

A roleta viciada para o exemplo acima é mostrada na figura abaixo. Podemos notar que alocamos um espaço igual a avaliação de cada indivíduo, dividida pela soma das avaliações de todos os indivíduos.

**Figura 3** – Roleta viciada para população

Diagrama

Descrição gerada automaticamente

Fonte: LINDEN, R. Algoritmos genéticos teoria e implementação.

Depois que cada indivíduo possui o seu espaço alocado na roleta, devemos fazer a implementação do algoritmo que irá simular uma roleta, segundo LIDEN (2006) o algoritmo da roleta possui o seguinte pseudocódigo.

**Figura 4** – Simulação

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Fonte: LINDEN, R. Algoritmos genéticos teoria e implementação.

O índice do indivíduo escolhido é dado pela variável *i*, e o algoritmo tem parada garantida, visto que *aux* tende ao somatório das avaliações dos indivíduos e *s* é menor que esta soma.

Apesar desse método ser o mais simples e utilizado, vários pesquisadores são contra o seu uso, segundo (LOPES,2006) o método da roleta é muito agressivo, pois discrimina indivíduos de melhor fitness de indivíduos de menor valor, e isso acaba ocasionando uma divergência em poucas gerações e possui grandes chances de ficar estagnado em um máximo local. Para solucionar o problema é recomendado utilizar métodos de ordenamento, em especial o linear que reduzem este tipo de problema, o método baseado em rank é um exemplo desta classe.

### 3.1.6 Crossover

Depois que os pais são selecionados é selecionado um ponto de corte. Um ponto de corte é uma posição entre os dois genes de um cromossomo. A metade à esquerda vai para um filho e outra metade vai para outro.

**Figura 5** - Representação cromossomo

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Fonte: Princípios de Genética, Robert H. Tamarin, 2011

Depois que um ponto do corte é selecionado, os pais são separados em duas partes: uma à esquerda do ponto de corte e outra à direta. Os dois pontos de cortes podem ter tamanhos variados e não precisam ter o mesmo tamanho.

O primeiro filho é composto pela concatenação da parte esquerda do primeiro pai com a parte direita do segundo pai. Já o segundo filho é gerado com as partes que sobraram.

### 3.1.7 Mutação

Depois que é aplicado o operador de crossover, o operador de mutação é aplicado. Ele tem associado a ele uma probabilidade extremamente baixa e nós sorteamos um número entre 0 e 1. Se ele for menor que a probabilidade então o operador atua sobre gene em questão, alterando o valor aletoriamente.

**Figura 6** - Representação mutação cromossomo

Texto

Descrição gerada automaticamente com confiança baixa

Fonte: Princípios de Genética, Robert H. Tamarin, 2011

O módulo de população é responsável pelo controle da nossa população, que por questão de recursos não pode ser ilimitada.

O módulo de população que utilizaremos por enquanto é extremamente simples. Sabemos que a cada atuação do nosso operador genético estamos criando dois filhos. Estes vão sendo armazenados em um espaço auxiliar até que o número de filhos criado seja igual ao tamanho da nossa população.

# BANCO DE DADOS

Banco de dados é uma coleção organizada de informações que normalmente são armazenadas em um computador. Um banco de dados normalmente utilizado em conjunto com um sistema SGBD (Sistema de Gerenciamento de Banco de Dados)

Os primeiros projetos de banco de dados começaram a ser pesquisados em 1960, a necessidade surgiu após as empresas perceberem que o custo de uma grande equipe de pessoas para organizar e manter documentos era muito alto.

Nessa mesma década foram realizadas diversas pesquisas, e foram propostos vários modelos de banco de dados, sendo eles, hierárquicos, em redes e relacionais.

## **4.1. Modelo hierárquico**

**Figura 7** – Exemplo de banco de dados hierárquico.

Diagrama

Descrição gerada automaticamente

Fonte: Documentação MariaDB.

O modelo de banco de dados hierárquico é o mais antigo proposto, ele possui semelhança a uma árvore de cabeça para baixo. Os arquivos são relacionados de uma maneira pai para filho, os pais são capazes de se relacionar com mais de um filho, mas um filho sendo capaz de se relacionar com apenas um dos pais.

Este modelo apesar de ser um grande avanço no tratamento e armazenamento de informações de arquivos não relacionais, possui algumas desvantagens que deixam seu uso complicado. Este modelo representa muito bem os relacionamentos de um para muitos, mas tem problemas com relacionamentos de muitos para muitos.

## 

## **4.2. Modelo em rede**

**Figura 8** - Representação banco de dados em rede

Diagrama

Descrição gerada automaticamente

Fonte: Documentação MariaDB.

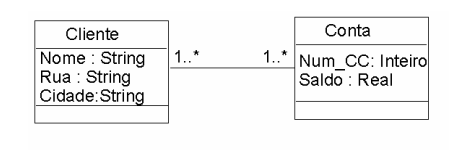
O modelo de banco de dados em rede foi uma progressão do modelo de banco de dados hierárquico e foi projeto para resolver alguns dos problemas existentes. A principal diferença desse modelo para o hierárquico é que nele é possui utilizar relacionamentos de muitos para muitos. Como você pode ver na figura abaixo, A1 tem dois membros, B1 e B2. B1 é o proprietário de C1, C2, C3 e C4. No entanto, neste modelo, C4 tem dois proprietários, B1 e B2.

Apesar do modelo ser uma grande evolução em relação ao modelo hierárquico, ele também possui os seus defeitos, é mais difícil de implementar e manter, e, embora seja mais flexível que o modelo hierárquico, ainda tem problemas de flexibilidade.

## 

## **4.3. Modelo relacional**

**Figura 9** - Representação banco de dados em rede



Fonte: Introdução a Banco de Dados, O.K.TAKAY, pg. 08.

O modelo relacional foi criado em 1990 por Ted Cold, devido a sua facilidade de uso ele atraiu uma grande atenção logo após ser lançado. Um banco de dados relacional que armazena e fornece acesso aos dados relacionados entre si. Em um banco de dados relacional, cada linha da tabela é um registro com um ID exclusiva chamada de chave. As colunas da tabela possuem atributos e cada registro possui uma chave para cada atributo.

Existem dois tipos de chaves, a chave primária, que é o valor ou conjunto de valores que identificam uma tupla dentro de uma tabela. E a chave estrangeira, que é o valor que representa uma chave primaria em outra tabela.

## 

## **4.4. Banco de dados não relacionais**

O modelo não relacional, permite que dados estruturados e semiestruturados sejam manipulados, esse modelo vai em contraste ao modelo relacional, onde os dados devem ser compostos.

## 

## **4.5. PostgreSQL**

PostgreSQL é um sistema de base de dados relacional avançado, de classe corporativa e de código aberto, suportando vários tipos de consultas, como SQL e JSON.

O PostgreSQL teve seu início no ano de 1986 no Departamento de Ciência da Computação de Berkeley, Universidade da Califórnia. Seu nome, POSTGRES, foi baseado no antigo banco de dados Ingres, que também foi desenvolvido em Berkeley.

Em 1996, o projeto POSTGRES teve seu nome alterado para PostgreSQL, ilustrando claramente seu suporte para total a linguagem SQL.

PostgreSQL foi projetado, inicialmente para rodar em plataformas do tipo UNIX. Ao passar dos anos, o PostgreSQL passou rodar em várias plataformas, como Windows, MacOS, dentre outras.

# DESENVOLVIMENTO

Podemos afirmar com toda certeza, que o primeiro passo, e talvez o mais importante para o início de um projeto é o planejamento. A fim de obter uma melhor visualização do sistema escolhido, surgiu a necessidade de utilização de algumas técnicas, abordagens e até mesmo, métodos de gerenciamento de projetos. O *Agile Project Management* foi a metodologia escolhida, pois permite que o sistema seja processado em pequenas fases, se tornando extremamente flexível ao projeto em questão, que exige mudanças e melhorias contínuas ao decorrer do desenvolvimento.

Devido as características dinâmicas do projeto, foram definidos sprints, com tempo de finalização entre 15 e 25 dias, e a cada conclusão, um novo sprint se iniciava contento novas tarefas, melhorias e mudanças em relação ao precedente. Esse planejamento, sem dúvidas, foi muito importante para que fosse possível garantir a finalização do sistema no tempo estabelecido.

## **5.1 Fluxo de usabilidade do sistema**

O aplicativo referente ao sistema desenvolvido, é composto por algumas páginas principais, oferecendo ao usuário uma manipulação completa de suas informações. O fluxo de usabilidade funciona da seguinte forma:

* **Cadastro de Plantas:** O usuário poderá cadastrar quantas plantas forem necessárias, informando os traços fenotípicos relevantes de sua escolha. Dessa forma, o modelo é definido e armazenado para a planta cadastrada.
* **Cadastro de Combinações:** Logo após, o usuário será induzido á uma tela de cadastro de combinações, onde de início, será necessário selecionar uma planta cadastrada para ser utilizada como modelo.
* **Cadastro de Parentais:** O preenchimento dos parentais será feito logo em seguida, informando os valores de cada fenótipo existente no modelo selecionado na página anterior.
* **Cálculo de Cruzamento:** Ao final, com todos os parentais cadastrados, será possível solicitar um cálculo de cruzamento, que quando finalizado, redireciona o usuário á uma tela de visualização das melhores combinações encontradas.

## **5.2 Tecnologias e ferramentas utilizadas**

Devido as diversas tecnologias já existentes e que surgem no mercado a cada dia, se torna uma tarefa muito difícil a escolha das ferramentas que serão utilizadas para o desenvolvimento do projeto. Cada projeto se encaixa melhor com uma determinada tecnologia, cabe aos idealizadores ou gestores do projeto definir a que melhor atende ao cenário em que se encontra.

Para o desenvolvimento do webservice, foi escolhida a linguagem de programação estatística R, pela velocidade e o ambiente propício para cálculos gigantes e complexos. Já para o desenvolvimento do aplicativo, o Ionic Framework foi a ferramenta que melhor atendeu, justamente por ser uma tecnologia multiplataforma, ou seja, para desenvolvimento de aplicativos híbridos (IOS, Android, WindowsPhone), e também, por ser uma ferramenta ótima para o desenvolvimento font-end.

### 5.2.1 R

R é uma linguagem e ambiente para computação estatística e gráficos. É um projeto GNU que é semelhante à linguagem e ambiente S que foi desenvolvido na Bell Laboratories (anteriormente AT&T, agora Lucent Technologies) por John Chambers e seus colegas.

R pode ser considerado como uma implementação diferente de S. Existem algumas diferenças importantes, mas muitos códigos escritos para S são executados inalterados em R.

R fornece uma ampla variedade de técnicas estatísticas (modelagem linear e não linear, testes estatísticos clássicos, análise de série temporal, classificação, agrupamento, etc.), técnicas gráficas, e é altamente extensível. A linguagem S é frequentemente o veículo de escolha para pesquisa em metodologia estatística, e R fornece uma rota de código aberto para a participação nessa atividade.

### 5.2.2 Ionic Framework

Ionic Framework é uma estrutura da web de código aberto para o desenvolvimento de aplicativos móveis. Torna possível usar diferentes dispositivos e com apenas uma base de código. Ionic é baseado em Angular e Apache Cordova. O framework Ionic é usado especialmente no desenvolvimento de aplicativos progressivos e híbridos baseados em HTML, mas também CSS e Sass. O Ionic Framework está sob licença do MIT, o que significa que pode ser usado para conceitos comerciais e não comerciais.

Como uma estrutura de front-end para o desenvolvimento de aplicativos móveis híbridos e progressivos, o Ionic usa principalmente HTML5. Devido às melhorias no mundo HTML5, os desenvolvedores projetaram o Ionic como um dos primeiros aplicativos nativos e móveis baseados em HTML, CSS (Cascading Style Sheets) e JavaScript.

# CONSIDERAÇÕES FINAIS

O objetivo desse estudo foi demonstrar como técnicas de aprendizado de máquina (Machine Learning) e aprendizado profundo de máquina (Deep Learning) podem ser empregadas no setor de agronegócio, foi possível notar como que as empresas que atuam no setor de agronegócio em sua grande maioria ainda não perceberam o potencial dessa tecnologia.

Também foi possível notar em como a inteligência artificial pode melhorar ou até mesmo mudar a forma em como o melhoramento genético pode ser feito, hoje para se realizar o melhoramento genético, demora muito ou tem um gasto muito alto, porém com o uso de dados já existentes e com conhecimento de técnicas de aprendizado de máquina é possível reduzir custos e o tempo para que o processo seja feito.

Também podemos notar que uma vez que os algoritmos de aprendizado de máquinas são adaptáveis, podemos então implementá-los em outras áreas do agronegócio, sendo necessário a utilização de uma grande base de dados.

Podemos notar que esse estudo é apenas uma demonstração em como a inteligência artificial pode ser utilizada no processo de melhoramento, ainda existem diversos avanços a serem feitos para realmente tornar possível e viável o uso de modelos de Machine Learning que são inteligentes o bastante para serem 100% efetivos em suas análises.

Analisando o objetivo do trabalho e o tema proposto podemos afirmar que a produção do mesmo se tonar importante e que atendeu o objetivo que foi proposto que foi de mostrar que é possível utilizar inteligência de dados combinada com dados históricos no melhoramento genético.

# REFERÊNCIAS BIBLIOGRÁFICAS

AGUILAR, L. J. **Fundamentos de Programação,Algoritmos, estruturas de dados e objetos**. 3. ed. Porto Alegre: AMGH Editora, 2008.

ALLEN B. TUCKER, R. E. N. **Linguagens de Programação:** Princípios e Paradigmas. 2ª. ed. [S.l.]: AMGH, 2009.

ILOMO, K. Understanding the Hierarchical Database Model. **mariaDB**, 2016. Disponivel em: <https://mariadb.com/kb/en/understanding-the-hierarchical-database-model/>. Acesso em: 23/03/2021 mar. 2021.

. 1º. ed. São Paulo: [s.n.], 2006.

LOPES, H. S. **Fundamentos da Computação Evolucionária e Aplica¸c˜oes. Bandeirantes**. Paraná: Bandeirantes, 2006.

OSVALDO KOTARO TAKAI. **Introdução A Banco De Dados**. São Paulo:São Paulo, 2005.

MOREIRA, F. J. L. A. E. J. R. (Ed.). **Mendel:** das Leis da hereditariedade à engenharia genética. 1. ed. São paulo : [s.n.], 2017.

O Que É um Banco de Dados Relacional. **oracle**. Disponivel em: <https://www.oracle.com/br/database/what-is-a-relational-database/>. Acesso em: 23 mar. 2021.

ROSA, J. L. G. **Fundamentos da inteligência artificial**. 1. ed. São Paulo: LTC, 2011.

UNDERSTANDING the Network Database Model. **mariaDB**, 2016. Disponivel em: <https://mariadb.com/kb/en/understanding-the-network-database-model/>. Acesso em: 21 mar. 2021.

WHAT-IS-DATABASE. **oracle**. Disponivel em: <https://www.oracle.com/br/database/what-is-database/>. Acesso em: 23 mar. 2021.

ODED GOLDREICH. **Completude P, NP e NP: os princípios da complexidade computacional**. 1º ed. Cambridge University Press Editora, 2010.

ROBERT H. TAMARIN. **Princípios de Genética**. 7º ed., FUNPEC, outubro, 2011.