UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE CAMPUS DE NATAL
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
CURSO BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MARIA DAS VIRGENS SILVA SOUZA

OTIMIZAÇÃO DO PLANEJAMENTO DE FLUXOS CURRICULARES USANDO ALGORITMO GENÉTICO: UM ESTUDO DE CASO NO CURSO DE CIÊNCIA DA COMPUTAÇÃO DA UERN NATAL

NATAL

2024

MARIA DAS VIRGENS SILVA SOUZA

OTIMIZAÇÃO DO PLANEJAMENTO DE FLUXOS CURRICULARES USANDO ALGORITMO GENÉTICO: UM ESTUDO DE CASO NO CURSO DE CIÊNCIA DA COMPUTAÇÃO DA UERN NATAL

Trabalho de Conclusão de Curso apresentado ao Departamento de Ciência da Computação como requisito parcial para conclusão do curso Bacharelado em Ciência da Computação..

Orientador(a): Dra. Adriana Takahashi
Co-orientador (a): Dra. Bartira Rocha

NATAL

2024

Catalogação da Publicação na Fonte.
Universidade do Estado do Rio Grande do Norte.

S586o Silva Souza, Maria das Virgens
OTIMIZAÇÃO DO PLANEJAMENTO DE FLUXOS CURRICULARES USANDO ALGORITMO GENÉTICO: UM ESTUDO DE CASO NO CURSO DE CIÊNCIA DA COMPUTAÇÃO DA UERN NATAL. / Maria das Virgens Silva Souza. - Natal, 2024.
63p.

Orientador(a): Profa. Dra. Adriana Takahashi.
Coorientador(a): Profa. Dra. Bartira Rocha.
Monografia (Graduação em Ciência da Computação). Universidade do Estado do Rio Grande do Norte.

1. Ciência da Computação. 2. Otimização. 3. Algoritmos genéticos. 4. Evasão. I. Takahashi, Adriana. II. Universidade do Estado do Rio Grande do Norte. III. Título.

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC´s) foi desenvolvido pela Diretoria de Informatização (DINF), sob orientação dos bibliotecários do SIB-UERN, para ser adaptado às necessidades da comunidade acadêmica UERN.

MARIA DAS VIRGENS SILVA SOUZA

OTIMIZAÇÃO DO PLANEJAMENTO DE FLUXOS CURRICULARES USANDO ALGORITMO GENÉTICO: UM ESTUDO DE CASO NO CURSO DE CIÊNCIA DA COMPUTAÇÃO DA UERN NATAL

Trabalho de Conclusão de Curso apresentado ao Departamento de Ciência da Computação como requisito parcial para conclusão do curso Bacharelado em Ciência da Computação..

Aprovada em: 23 / 07 / 2024.

Banca examinadora

Adriana Takahashi

Profª. Drª. Adriana Takahashi (Orientador)
Universidade do Estado do Rio Grande do Norte - UERN

Profª. Drª. Bartira Paraguaçu Falcão Dantas Rocha
Universidade do Estado do Rio Grande do Norte - UERN

André Gustavo P. da Silva

Profª. Me. André Gustavo P. Da Silva
Universidade do Estado do Rio Grande do Norte - UERN

Bruno Cruz de Oliveira

Profª. Me. Bruno Cruz de Oliveira
Universidade do Estado do Rio Grande do Norte - UERN

RESUMO

O acesso à educação superior brasileira teve grande incentivo e aumento de vagas nas últimas décadas, inclusive nas áreas de computação. Contudo, o número de ingressantes e de concluintes não tiveram o mesmo crescimento proporcional. Algumas das causas que podem ser apontadas para essa alta taxa de evasão é o pouco direcionamento oferecido ao aluno em relação ao seu fluxo curricular, principalmente nos semestres iniciais, e o desnivelamento de sua situação acadêmica no decorrer do curso. Tendo isso em vista, o presente trabalho se propõe a auxiliar a orientação acadêmica do curso na definição dos fluxos curriculares que proporcionem ao aluno a conclusão de sua graduação no menor tempo possível. Para tal propósito, foi realizado o estudo e aplicação de um algoritmo genético para a geração de sugestões de sequências ideais de disciplinas que possam ser cursadas e servirem de parâmetro para uma distribuição mais eficiente dos horários de aula nos semestres vindouros, possibilitando aos discentes melhor aproveitamento das ofertas. Para isso, foi realizado um estudo de caso tendo como base o curso de Ciência da Computação da Universidade do Estado do Rio Grande do Norte - Campus Natal.

Palavras-chave: Otimização; Algoritmos genéticos; Evasão.

ABSTRACT

Access to higher education in Brazil has been greatly encouraged and the number of available spots has increased significantly in recent decades, including in the field of computing. However, the number of new entrants and graduates has not grown proportionally. Some of the causes that can be pointed out for this high dropout rate are the lack of guidance offered to students regarding their curricular flow, especially in the initial semesters, and the imbalance of their academic situation throughout the course. With this in mind, the present work aims to assist the academic advising of the course in defining curricular flows that enable students to complete their graduation in the shortest possible time. For this purpose, a study and application of a genetic algorithm were conducted to generate suggestions for ideal sequences of subjects that can be taken and serve as a parameter for a more efficient distribution of class schedules in future semesters, enabling students to make better use of the offerings. To this end, a case study was conducted based on the Computer Science course at the State University of Rio Grande do Norte - Natal Campus.

Keywords: Optimization; Genetic algorithms; Dropout.

LISTA DE FIGURAS

Figura 01 – Exemplo de crossover..... 11
Figura 02 – Pseudocódigo de um algoritmo genético..... 12
Figura 03 – Diagrama de caso de uso do sistema..... 18
Figura 04 – Diagrama de classes do algoritmo..... 19
Figura 05 – Wireframe da tela inicial do sistema.....19
Figura 06 – Wireframe da exibição das sugestões geradas pelo algoritmo.....20
Figura 07 – Wireframe dos elementos de select e botões..... 20
Figura 08 – Matriz curricular para ingressantes até o ano de 2022..... 21
Figura 09 – Matriz curricular para ingressantes após o ano de 2022.....22
Figura 10 – Interface inicial do sistema..... 25
Figura 11 – Resultado do algoritmo na interface.....26
Figura 12 – Fluxograma do funcionamento do algoritmo genético.....27
Figura 12 – Representação inicial de um indivíduo da população..... 30
Figura 13 – Representação do agrupamento de um indivíduo por tipo de semestre30
Figura 14 – Representação final de uma solução..... 31
Figura 15 – Esquema das disciplinas não pagas pelo Aluno 10..... 39
Figura 16 – Resultado ideal gerado de forma manual.....39
Figura 17 – Melhor resultado gerado pelo algoritmo.....40
Figura 18 – Esquema das disciplinas não pagas e de seus pré-requisitos para o Aluno 03.....42
Figura 19 – Esquema das disciplinas não pagas e de seus pré-requisitos para o Aluno 07.....43

LISTA DE TABELAS

Tabela 01 – Dados dos ingressantes do ano de 2018 do curso de Ciência da Computação na UERN..... 23

Tabela 02 – Dados dos ingressantes do ano de 2017 do curso de Ciência da Computação na UERN..... 23

Tabela 03 – Dados dos ingressantes do ano de 2013 do curso de Ciência da Computação na UERN..... 23

Tabela 04 – Dados da amostra considerada para os testes.....27

Tabela 05 – Atributos das disciplinas.....28

Tabela 06 – Restrições de pré-requisitos..... 32

Tabela 07 – Penalização por quantidade de disciplinas no semestre..... 36

Tabela 08 – Resultados do algoritmo para semestres pares.....37

Tabela 09 – Resultados do algoritmo para semestres ímpares.....40

Tabela 10 – Resultados do Aluno 05 após aumento da população e quantidade de gerações.....41

Tabela 11 – Disciplinas não pagas dos alunos que tiveram soluções que violaram os requisitos..... 41

LISTA DE CÓDIGOS

Código 01 - Exemplo de uma aplicação Flask..... 13
Código 02 - Exemplo de função em javascript para envio de dados.....14
Código 03 – Função que representa as disciplinas de acordo com o tipo do semestre..... 28
Código 04 – Funções de criação e diversificação da população.....31
Código 05 – Função de verificação de violação de pré-requisito entre semestres... 33
Código 06 – Função de verificação de pré-requisitos no mesmo semestre.....33
Código 07 – Função de ordenação da população.....35
Código 08 – Função para soma de quantidade de semestres da solução.....35

SUMÁRIO

1 INTRODUÇÃO..... 4
1.1 Objetivos do trabalho.....5
1.1.1 Objetivo geral.....5
1.1.2 Objetivos específicos.....5
1.2 Metodologia.....6
1.3 Organização do Trabalho.....6
2 FUNDAMENTAÇÃO TEÓRICA.....8
2.1 Métodos de otimização..... 8
2.1.1 Métodos de otimização Determinísticos.....8
2.1.2 Métodos de otimização Estocásticos/aleatórios.....8
2.2 Algoritmos genéticos.....9
2.2.1 Representação da solução.....9
2.2.2 Geração da população inicial..... 10
2.2.3 Avaliação da população.....10
2.2.4 Seleção das soluções mais aptas..... 10
2.2.5 Diversificação da população..... 11
2.2.6 Reavaliação e geração da nova população.....12
2.3 Ferramentas utilizadas.....12
2.3.1 Python.....13
2.3.2 Flask..... 13
2.3.3 Javascript.....14
2.4 Trabalhos Relacionados..... 15
3 OTIMIZAÇÃO DE FLUXO CURRICULAR.....17
3.1 Descrição Geral..... 17
3.2 Requisitos Funcionais..... 17
3.3 Diagrama de classe..... 18
3.4 Prototipagem das telas..... 19
4 PROVA DE CONCEITO..... 21
4.1 Dados do curso.....21
4.2 Interface.....24
4.3 Estratégia de resolução..... 26
4.3.1 Descrição da Amostra..... 27
4.3.2 Representação de uma solução.....27
4.3.3 População.....30
4.3.4 Função de aptidão.....30
4.3.4.1 Análise dos pré-requisitos..... 31
4.3.4.2 Quantidade de semestres.....31
4.3.4.3 Quantidade de disciplinas por semestre.....32
4.3.4.4 Restrição por período atual..... 32

4.3.5 Análise dos resultados.....33
5 CONSIDERAÇÕES FINAIS..... 37
5.1 Conclusão..... 37
5.2 Trabalhos futuros..... 37
REFERÊNCIAS..... 39
APÊNDICE - Código do algoritmo genético gerado..... 42

1 INTRODUÇÃO

Nas últimas décadas o acesso à educação superior brasileira foi incentivado e facilitado através de diversas políticas públicas, dentre elas o investimento na construção de novas instituições de ensino que, segundo o Censo da Educação Superior de 2020 (INEP, 2021), teve um aumento de 3,32% desde o ano de 2010 até o ano de 2020. Acompanhando esse crescimento, a quantidade de cursos e vagas disponibilizadas por essas instituições também aumentou, chegando a 42,16% e 36,08%, respectivamente. Por consequência, o ingresso de novos alunos nas universidades também apresentou aumento, passando de 2.182.229 para 3.765.475 (aumento de 72,59%).

Contudo, ao contrário do que se é esperado, a quantidade de alunos que de fato concluíram o curso não acompanhou o crescimento do número de vagas e universidades, e na última década teve somente 30,39% de aumento. Essa situação ocorre principalmente nas áreas de tecnologia e exatas, onde em 2020, a cada 10,8 estudantes que ingressaram no ensino superior, somente 2,4 chegaram de fato a concluir a graduação, representando 77,78% de desistência na área, apesar da alta demanda do mercado (INEP, 2021). Também é possível visualizar essa discrepância (entre números de vagas, ingressantes e concluintes), ao analisarmos a quantidade de matrículas disponibilizadas no ano de 2020 para o curso de Sistemas da informação. Foram 209.182 vagas disponibilizadas para matrícula, das quais 124.317 (59,43% das vagas ofertadas) foram preenchidas e somente 28.618 desses ingressantes chegaram a concluir o curso (23% dos ingressantes). Tal situação se repete nos anos anteriores indicando que, por mais que o número de vagas tenha aumentado e o número de ingressantes esteja acompanhando esse aumento, a quantidade de concluintes se encontra em baixa quando comparado ao número de ingressantes.

Sobre tal perspectiva, é preciso entender o que tem levado o estudante do ensino superior brasileiro a desistência da graduação, já que tal situação, segundo Filho (2007), incide diretamente na carreira profissional do indivíduo, pois sem uma formação superior o acesso às melhores vagas no mercado de trabalho é dificultado. Essa situação também tem alto impacto nas instituições de ensino, que acabam com muitas vagas ociosas, perdas financeiras e com baixo índice de retenção de alunos. Além disso, a sociedade também é impactada pelas altas taxas de evasão, já que isso implica em mão de obra não qualificada e falta de oportunidades de crescimento no mercado de trabalho.

Um dos principais fatores que influenciam na evasão do aluno de ensino superior, segundo Polydoro (2005), é a desmotivação causada pelo mau desempenho em disciplinas básicas/chaves do curso, pois isso pode acarretar na necessidade de reorganização das disciplinas em diferentes semestres e turmas, prejudicando o gerenciamento das atividades curriculares do aluno e desmotivando-o a permanecer na instituição. Além disso, o desnivelamento do aluno também traz complicações para a oferta de disciplinas, pois adiciona mais complexidade no processo de organização de horários por parte da orientação

acadêmica do curso, que se vê na necessidade de ofertar uma grade de horários que ajude o aluno desnivelado a se formar e que não prejudique o aluno nivelado.

Considerando esse cenário, é preciso responder às seguintes questões de pesquisa:

- Como otimizar o fluxo curricular a partir dos períodos iniciais do curso?
- Como identificar qual o melhor fluxo curricular para alunos desnivelados considerando pré-requisitos e tipo de semestre, se ímpar ou par?
- Como auxiliar a orientação acadêmica na montagem dos horários e disciplinas ofertadas semestralmente?

Tendo em vista as questões de pesquisa apresentadas, acredita-se que o desenvolvimento de um algoritmo que, através de métodos de otimização, e levando em consideração os pré-requisitos das disciplinas, suas cargas horárias, o nivelamento do aluno e as disciplinas ofertadas pela universidade, assim como os horários e professores disponíveis, possa sugerir uma sequência ideal de disciplinas para serem ofertadas/cursadas. Com isso, é possível evitar atrasos na conclusão do curso e aprimorar/melhorar o aproveitamento acadêmico dos alunos, motivando-os a continuar na graduação.

1.1 Objetivos do trabalho

1.1.1 Objetivo geral

- O presente trabalho se propõe a auxiliar a orientação acadêmica do curso na sugestão de fluxos curriculares, com o objetivo de melhorar o processo de formação de um aluno do ensino superior, através de técnicas de resolução de problemas utilizando um algoritmo genético, tendo como estudo de caso o curso de Ciência da Computação da UERN no campus Natal.

1.1.2 Objetivos específicos

Os seguintes objetivos específicos foram realizados para se alcançar o objetivo geral apresentado:

- Revisão da literatura sobre evasão universitária no sistema de ensino brasileiro.
- Revisão literária sobre técnicas de resolução de problemas utilizando buscas e algoritmos genéticos para construção de planejamento curricular de alunos do ensino superior.
- Realização da análise e modelagem de requisitos para o desenvolvimento do software (desenvolvimentos de modelos e projeto de implementação)

- Realização de testes de funcionalidade e usabilidade, para verificar erros e possíveis melhorias

1.2 Metodologia

Segundo Rodrigues (2007), metodologia científica é um conjunto de abordagens, técnicas e processos utilizados para formular e resolver problemas de maneira sistemática. Tal metodologia pode ser classificada de acordo com sua finalidade, objetivo, abordagem, método ou procedimento.

Para obter os resultados e respostas acerca da problematização apresentada neste trabalho, realizou-se uma pesquisa documental sobre os índices de evasão registrados nos últimos anos no país e quais foram suas causas, apresentando também uma solução em forma de software para amenizá-las. Tal pesquisa foi baseada nos dados fornecidos nos últimos anos pelo portal do Ministério da Educação, por estudos realizados na área de evasão escolar e na área de computação. Dessa forma, todos os dados presentes no estudo são dados públicos e de acesso garantido aos cidadãos, sendo disponibilizados anualmente pelo governo, através do site oficial do INEP. Tal procedimento de pesquisa caracteriza-se como bibliográfico e documental, pois, conforme descrição dada por Santos (2006), é através da pesquisa bibliográfica e documental que o trabalho é situado dentro da grande área de pesquisa da qual faz parte, contextualizando-o, pois ao citar uma série de estudos prévios que servirão como ponto de partida para sua pesquisa, você acaba “afunilando” sua discussão.

Quanto a seus objetivos, foi utilizado a abordagem de pesquisa descritiva, pois segundo Wohlin e Aurum (2014), a pesquisa descritiva é aplicada para descrever um fenômeno ou características de um problema, aplicando-se no trabalho através da análise e obtenção dos dados citados acima, onde é possível descrever o processo de evasão e sugerir uma aplicação para amenizá-la.

A partir da pesquisa realizada foram identificadas algumas causas principais do problema, sendo elas: a falta de compatibilidade com o curso, a desmotivação pela reprovação em alguma disciplina chave e a falta de informações sobre o curso escolhido. Com isso em mente, é proposto neste trabalho o estudo e aplicação de técnicas de resolução de problemas utilizando buscas e algoritmos genéticos, visando auxiliar a orientação acadêmica na definição dos fluxos curriculares que proporcionem ao aluno a conclusão de sua graduação no menor tempo possível. Sendo assim, pode se afirmar que a finalidade do trabalho também é de natureza aplicada, pois segundo Gil (1999), o que caracteriza a pesquisa aplicada é o fato dela ser voltada para o interesse da aplicação, utilização e consequências práticas de conhecimento.

1.3 Organização do Trabalho

Para cumprir os objetivos citados, o presente trabalho organiza-se em 5 capítulos, incluindo a introdução acima.

No capítulo 2, serão descritos os referenciais teóricos consultados para o desenvolvimento do trabalho, sendo eles os métodos de otimização, o funcionamento de algoritmos genéticos, ferramentas de desenvolvimento e trabalhos correlacionados.

No capítulo 3, será descrito o processo de modelagem do sistema desenvolvido, apresentando seus requisitos, usuários, caso de uso, diagramas e prototipagem.

No capítulo 4, serão apresentados os resultados obtidos com a implementação do algoritmo desenvolvido, de acordo com as especificações apresentadas nos capítulos anteriores.

No capítulo 5, serão apresentadas as conclusões do trabalho e as perspectivas de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A seguir serão apresentados alguns referenciais teóricos pertinentes ao desenvolvimento do trabalho.

2.1 Métodos de otimização

Segundo Lacerda e Carvalho (1999), otimização é a busca da melhor solução para um dado problema e consiste em tentar várias soluções, utilizando as informações obtidas nessas tentativas, para encontrar soluções cada vez melhores. De forma matemática, pode-se entender que a otimização se propõe a encontrar o ponto mínimo/máximo de uma função objetivo. Tal função objetivo pode gerar diversos pontos máximos, conhecidos como máximos locais (valor ótimo considerando uma parte do espaço de busca), mas esses pontos podem não representar a solução de maior valor da função, que seria o seu máximo global (valor ótimo considerando todo o espaço de busca).

É nesse contexto de busca pela melhor solução, que os métodos de otimização podem ser aplicados. Tais métodos são divididos em duas categorias, os métodos determinísticos e os estocásticos/aleatórios.

2.1.1 Métodos de otimização Determinísticos

Também conhecidos como métodos clássicos, os métodos determinísticos sempre levam à mesma resposta se partirem de um mesmo ponto inicial (SECCHI, 2004). Tais métodos são baseados no cálculo de derivadas e/ou em suas aproximações. Sua principal vantagem é a convergência rápida em relação aos outros métodos de otimização, resultando também em menor custo computacional. No entanto, em problemas específicos, este método pode acabar convergindo para pontos ótimos locais e não globais, o que os torna não recomendados para funções que possuem vários ótimos locais.

Podem ser citados como exemplos de métodos de otimização determinísticos os métodos de Máxima Descida, o Método de Newton e o Método Simplex (SARAMAGO e STEFFEN, 2010).

2.1.2 Métodos de otimização Estocásticos/aleatórios

Segundo Brandão (2010, apud PÁDUA, 2020) os métodos estocásticos podem ser denominados como métodos heurísticos, onde uma heurística pode ser tida como uma técnica computacional que alcança sempre uma solução viável para um dado problema de otimização, utilizando um esforço computacional considerado razoável (GOLDBARG; GOLDBARG; LUNA, 2016, p. 47). Para Wilhelm (2022), métodos heurísticos são procedimentos que seguem uma ‘intuição’ para resolver um determinado problema e, apesar de não garantirem uma resolução ótima, em geral produzem rapidamente soluções finais de boa qualidade.

Apesar dos métodos estocásticos apresentarem melhores resultados na busca de ótimos globais, o custo operacional no cálculo da função objetivo pode ser elevado quando comparado com os métodos determinísticos.

2.2 Algoritmos genéticos

Dentre os métodos heurísticos para resolução de problemas e otimização, os algoritmos genéticos, e outras analogias evolucionárias formais, produzem soluções para problemas com capacidade crescente, operando em populações de soluções candidatas para o problema (LUGER, 2013). Além disso, segundo Goldberg (1989, p. 05), o algoritmo genético é um exemplo de procedimento de busca que utiliza a escolha aleatória como ferramenta para guiar uma busca altamente exploratória através da codificação de um espaço de parâmetros, com isso diferenciando-se dos demais métodos de busca tradicionais, devido sua robustez.

Ademais, os algoritmos genéticos também são conhecidos por inspirarem-se na teoria Darwiniana do processo biológico de evolução, seguindo a premissa da seleção natural e da sobrevivência do ser mais apto. Baseando-se nisso, Luger nos diz que os algoritmos genéticos podem ser caracterizados por três estágios distintos:

> Quando o algoritmo genético é usado para solucionar um problema, ele tem três estágios distintos: primeiro, as soluções potenciais individuais do domínio do problema são codificadas em representações que suportam as variações e operações de seleção necessárias; [...] No segundo estágio, algoritmos de acasalamento e mutação, análogos à atividade sexual de formas vivas biológicas, produzem uma nova geração de indivíduos que recombinam características de seus pais. Por fim, uma função de aptidão julga quais indivíduos são as “melhores” formas de vida, isto é, mais apropriados para a solução eventual do problema (LUGER, 2013, p.420).

Ainda sobre o paralelo entre o processo evolucionário e o algoritmo genético, Luger nos diz que a aprendizagem desses algoritmos se assemelha a uma competição em uma população de soluções evolutivas que são candidatas para o problema. Tais soluções são avaliadas por uma função de aptidão que determina se a solução é satisfatória o suficiente para continuar no processo de evolução e a partir disso criar novas populações de soluções candidatas.

Tendo isso em vista, de forma geral, o ciclo de vida de um algoritmo genético pode ser definido da seguinte forma:

2.2.1 Representação da solução

Antes de mais nada, é preciso ter em mente que os algoritmos genéticos trabalham com a ideia de representar um problema real de forma que ele possa ser interpretado e recombinado por um computador. Além disso, como dito anteriormente, por ser uma heurística, ele se propõe a encontrar soluções próximas ao ótimo em um tempo de execução aceitável, o que está diretamente associado à forma como as soluções são codificadas/representadas. Ao escolher uma codificação adequada para o problema, o processo de busca pode ser facilitado, permitindo que as soluções ótimas sejam encontradas e em tempo computacional aceitável. No ramo computacional, normalmente é utilizada a representação por cadeia de bits, onde cada número representa algum tipo de característica do indivíduo.

Apesar da cadeia de bits ser a forma mais comum, ela não é a única. A depender da complexidade do problema, outras formas de representação podem ser abordadas e demonstram maior eficiência na busca.

Outras formas de representação adotadas são: a codificação Real, onde a solução é representada por números reais; a Permutação, onde a ordem dentro da codificação importa, e as árvores de sintaxe, onde as soluções são representadas como árvores de expressões matemáticas.

2.2.2 Geração da população inicial

Após decidida a forma de representação das soluções, o algoritmo genético é iniciado com um conjunto de soluções possíveis que, normalmente, são geradas aleatoriamente. Esse conjunto de soluções iniciais é reconhecido como a população inicial do algoritmo.

2.2.3 Avaliação da população

Após gerada a população inicial, todas as possíveis soluções são avaliadas através de uma função de aptidão, que analisa se a solução é apta o suficiente para contribuir para a próxima geração. A função de aptidão ou função de avaliação é um ponto chave na evolução do algoritmo genético e é através dela que o algoritmo irá realizar a busca pela melhor solução ótima.

2.2.4 Seleção das soluções mais aptas

Se caracteriza como um processo iterativo, onde a cada iteração as duas melhores soluções são escolhidas como pais para um processo de recombinação que deve gerar uma geração futura de soluções filhas. Esse processo, chamado de seleção, é normalmente implementado através de um procedimento conhecido como roda da roleta. A seleção por roda da roleta considera um círculo dividido em *n* áreas, onde cada área é ocupada proporcionalmente pelo valor de aptidão de cada indivíduo. Com isso, coloca-se sobre esse círculo uma roleta com *n* cursores, igualmente espaçados, e depois gira-se essa roleta, selecionando os indivíduos

apontados por esses cursores, adicionando-os a uma lista (lista de parceiros ou população intermediária). Este procedimento é apenas um dos procedimentos existentes para o processo de seleção, mas também podem ser citados a seleção por torneio e a seleção por roda da roleta baseada em classificação.

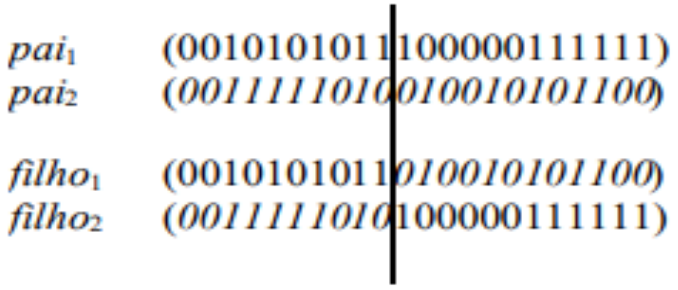
Na seleção por torneio, *n* indivíduos da população são selecionados aleatoriamente e competem entre si. O indivíduo que possuir a melhor aptidão é selecionado e incluído na população da próxima geração (GERAGHTY e RAZALI, 2011). A seleção por torneio dá a oportunidade a todos os indivíduos da população de serem selecionados, não necessariamente priorizando os de melhor resultado, o que garante uma população mais diversificada.

A seleção por roda da roleta baseada em classificação, segue o mesmo princípio da roda da roleta comum, porém, ao invés de considerar a porcentagem de aptidão do indivíduo, ela considera a classificação da aptidão de acordo com toda a população. Ainda segundo Geraghty e Razali (2011), o funcionamento da roda da roleta por classificação é realizado através da classificação dos indivíduos da população de acordo com sua aptidão, seguido do cálculo da probabilidade de seleção de acordo com essas classificações, diferenciado-se da roleta comum que leva em consideração somente a probabilidade da aptidão.

2.2.5 Diversificação da população

Após realizado o processo de seleção, as possíveis soluções do problema, também denominadas de cromossomos, escolhidas são submetidas a operação de *crossover* (cruzamento). Nesta etapa do processo um par de cromossomo é retirado aleatoriamente da lista de parceiros e tem suas cadeias de caracteres cortadas de forma aleatória, gerando duas combinações de dados. Em seguida esses dados são trocados, gerando dois novos cromossomos filhos. Dessa forma, os filhos gerados serão diferentes dos pais, mas ainda possuirão características diretas deles, como pode ser visto no esquema apresentado na Figura 01 (LACERDA e CARVALHO, 1999).

Figura 01 – Exemplo de crossover



Fonte: LACERDA e CARVALHO (1999)

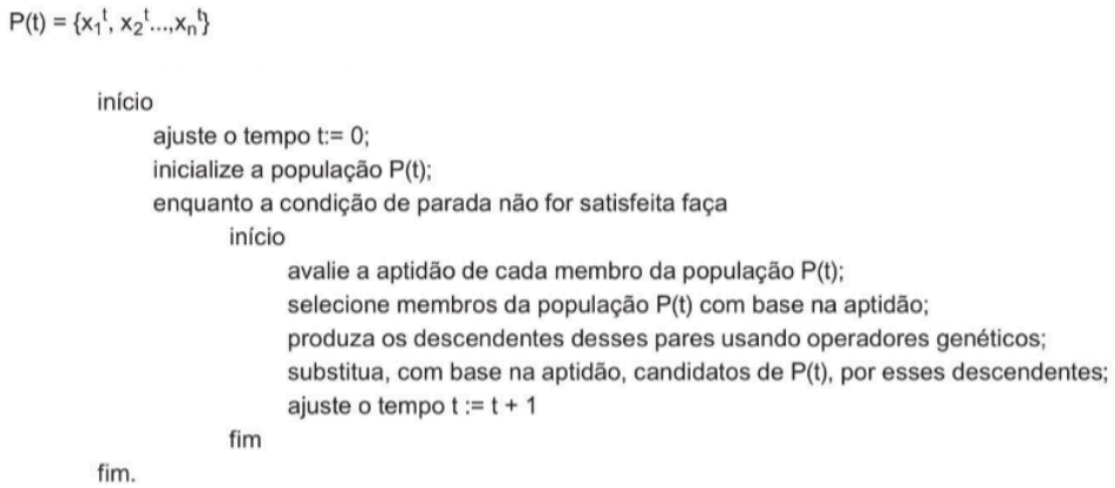
Após isso, os cromossomos filhos gerados pelo crossover são submetidos ao operador de mutação, que realiza uma alteração aleatória no indivíduo, levando em consideração uma taxa de mutação. Esta operação de mutação é interessante, pois melhora a diversidade da população. Além disso, como visto, a cada iteração do algoritmo o cromossomo é alterado, o que pode acarretar na perda de um cromossomo ótimo. Visando contornar essa situação, é possível aplicar ao algoritmo genético o método de elitismo, onde os melhores cromossomos encontrados são preservados e mantidos para a próxima iteração, podendo inclusive substituir os piores cromossomos da iteração. Com isso, a próxima interação irá trabalhar com melhores resultados, possibilitando que o algoritmo encontre a solução em menos iterações.

2.2.6 Reavaliação e geração da nova população

Após todo o processo de geração das novas soluções, a função de aptidão é novamente aplicada e os melhores resultados encontrados são levados para a nova população, descartando-se os que tiveram pior desempenho.

Na Figura 02, é possível visualizar a estruturação de um algoritmo genético através de seu pseudocódigo, levando em consideração que *P(t)* se refere a um conjunto de soluções candidatas x_i^t, no tempo *t*.

Figura 02 – Pseudocódigo de um algoritmo genético



Fonte: Luger, 2013.

2.3 Ferramentas utilizadas

Nesta seção serão descritas as ferramentas utilizadas para o desenvolvimento do projeto, sendo elas: A linguagem python para construção do algoritmo genético, o framework flask para o desenvolvimento da plataforma web e o javascript para comunicação com o servidor e geração dos HTML/CSS dinâmico.

2.3.1 Python

Segundo a ASSOCIATION FOR COMPUTING MACHINERY (2024), empresa responsável pela linguagem, o Python é uma linguagem de programação interpretada, interativa e orientada a objetos, que incorporou módulos, exceções, tipagem dinâmica, tipos de dados dinâmicos de alto nível e classes em uma única linguagem. Além disso, o python possui suporte para vários paradigmas de programação, além da já conhecida programação orientada a objetos, como a programação procedural e funcional.

A linguagem python é uma das mais populares no mundo e chama atenção devido as suas bibliotecas voltadas para inteligência artificial, análise de dados e resolução de problemas matemáticos, como as bibliotecas numpy e panda, mas também conta com bibliotecas já integradas a linguagem como a biblioteca random e math.

Uma aplicação prática da linguagem é sua utilização para algoritmos evolutivos, como o algoritmo genético, em que as bibliotecas listadas podem ser aplicadas para criação de populações ou cromossomos aleatórios, por exemplo. Assim como, também é possível realizar a leitura de e interpretação de diversos tipos de dados, através de bibliotecas para leitura de arquivos JSON e CSV.

2.3.2 Flask

Segundo o PyPI (2024), repositório de software para python, Flask é um mini framework na linguagem python voltado para criação de aplicações web. Apesar de ser tido como um ‘mini’ framework, o Flask é capaz de suportar aplicações complexas de forma rápida e fácil, mas com a vantagem de não impor dependências e layout de projeto complexo.

Além disso, o Flask conta com diversas extensões que são disponibilizadas e compartilhadas pela comunidade, permitindo que diversas funcionalidades sejam implementadas e que o framework esteja em constante evolução.

Em aplicações web, o framework Flask pode ser utilizado, por exemplo, para criação de uma API que permita a comunicação entre o front-end e o back-end de uma aplicação. No Código 01, é possível visualizar um exemplo da utilização desse framework, onde está sendo criado uma rota que aceita requisições do método POST, para receber dados de entrada de um usuário e retornar o resultado processado pela função *algoritmo()*.

Código 01 - Exemplo de uma aplicação Flask

```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/executar', methods=['POST'])
```

```
def executar_algoritmo():
    dados = request.json
    # Chamada para a função do algoritmo
    resultado = algoritmo(dados)
    return jsonify(resultado)
```

Fonte: elaborado pelo autor

2.3.3 Javascript

De acordo com o MDN Web Docs, site oficial da linguagem, o Javascript é uma linguagem de programação leve e interpretada, que é baseada em linguagem de scripts para páginas web, mas que pode ser usada em vários outros tipos de ambientes de software (MDN Web Docs, 2022) e pode ser usada tanto do lado do cliente, quanto do lado do servidor.

O JavaScript, junto ao HTML e ao CSS, é uma das principais tecnologias de desenvolvimento web utilizada nos dias atuais e tem como principal característica a capacidade de manipulação do DOM (Document Object Model), que consiste na representação de dados dos objetos que compõem a estrutura e o conteúdo de um documento na web (MDN Web Docs, 2023). Além disso, também é possível utilizá-lo para validação de formulários e comunicações assíncronas.

No resultado anual de 2023, das linguagens mais usadas no GitHub, o javascript está em primeiro lugar desde o ano de 2014 (GITHUB, 2023) e também aparece nessa mesma posição nos resultados anuais do StackOverflow, plataforma de compartilhamento de código, artigos e dúvidas sobre programação (STACKOVERFLOW, 2023). Além disso, nos resultados do StackOverflow, alguns frameworks baseados nessa linguagem ocupam o top 3 frameworks mais utilizados, sendo eles, em ordem: Node.Js, React e JQuery.

Um exemplo de aplicação da linguagem é a sua utilização para interação entre o front-end de uma aplicação e uma API, onde os dados são enviados através de uma função javascript e os resultados são retornados e exibidos na interface. O Código 02 exemplifica, através da função *enviarDados()*, como é possível enviar os dados de um usuário, através de uma requisição de método POST, para a rota */executar* de uma API, aguardar sua resposta em formato JSON e depois exibi-la em uma página web.

Código 02 - Exemplo de função em javascript para envio de dados

```
async function enviarDados(dados) {
    const response = await fetch('/executar', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify(dados),
```

```
    });
    const resultado = await response.json();
    console.log('Resultado:', resultado);
}
```

Fonte: elaborado pelo autor

2.4 Trabalhos Relacionados

Esta seção tem como objetivo apresentar um panorama geral de pesquisas e aplicações pertinentes ao tema de estudo deste trabalho. Através da revisão de literatura, é possível identificar as abordagens, metodologias e resultados obtidos por outros pesquisadores, bem como situar o presente trabalho dentro do contexto acadêmico e científico atual. Esta análise crítica não só evidencia a originalidade e relevância do problema abordado, mas também auxilia na identificação de lacunas e oportunidades para contribuições adicionais.

Para construção do presente trabalho, foram considerados estudos na área de evasão escolar em cursos de nível superior e otimização de fluxos curriculares.

Nos últimos anos diversos trabalhos foram publicados acerca da evasão no ensino superior brasileiro, entre eles, é possível citar o trabalho de Tontini e Walter (2012), que propõe um método para prever e mitigar o risco de evasão de alunos do ensino superior. Esse trabalho destaca-se na área, pois apresenta a aplicação do software desenvolvido em uma Instituição de Ensino Superior, assim como apresenta resultados satisfatórios quanto ao que se propõe, diminuindo em 18% o índice de evasão da instituição utilizada. O projeto apresentado por Tontini e Walter se assemelha ao presente trabalho, pois também busca atacar os níveis de evasão de um curso, utilizando uma instituição de Ensino Superior como estudo de caso, mas se difere no que diz respeito a coleta dos dados que foram utilizados para essa classificação, pois se basearam na aplicação de um questionário para os alunos da instituição.

Já o trabalho de Rodrigues (2015) apresenta um projeto que, indiretamente, também se propõe a atacar os índices de evasão. Para isso, o projeto apresenta um modelo matemático para otimização de matrícula semestral que tem como resultado uma sugestão de disciplinas, que podem otimizar o processo de graduação do aluno. Essa sugestão é baseada em um modelo de sequenciamento de projeto, levando em consideração o tempo de conclusão de curso e suas restrições (matérias com pré-requisitos ou que não são disponibilizadas semestralmente). Tal abordagem se assemelha ao presente trabalho por gerar uma sugestão de fluxo curricular ao aluno, levando em consideração pré-requisitos e disponibilidade de disciplinas no semestre, mas se difere na interface, no método utilizado para geração dessas sugestões e na forma como os dados são processados. Além disso, a solução proposta por Rodrigues é voltada exclusivamente para o curso de Engenharia de Produção da Universidade Federal de Ouro Preto e tem como objetivo permitir que o próprio aluno da universidade utilize sua ferramenta, o que não é o foco deste trabalho, pois somente está sendo realizado o estudo de caso no

curso de Ciência da Computação da UERN, mas o algoritmo proposto neste trabalho pode ser ajustado com facilidade para outras realidades e tem como objetivo ser uma ferramenta de auxílio a orientação acadêmica. Em relação aos resultados, dentro do contexto apresentado, Rodrigues utilizou o modelo proposto considerando seu próprio histórico escolar dentro da universidade, comparando uma tomada de decisão aleatória com a saída otimizada de seu modelo, como resultado, o modelo proposto conseguiu cumprir com o que foi proposto e otimizou o fluxo curricular, permitindo que o aluno concluísse a graduação antes do previsto.

Fregoneis (2002), também utiliza uma universidade específica para levantamento e análise de dados, tomando-a como estudo de caso dos índices de evasão, mas se difere do presente trabalho ao realizar somente a análise das motivações da evasão, sem indicar uma abordagem computacional para diminuir seus índices.

Boyer et al. (2021) se propõe a desenvolver um modelo para melhorar o processo de organização de horários para cursos universitários e também utiliza uma instituição específica para estudo de caso, mas tem por objetivo minimizar o tempo de deslocamento dos alunos e minimizar a capacidade excedente das salas de aula, enquanto o presente trabalho tem foco na otimização de tempo de conclusão de curso, principalmente para alunos desnivelados.

3 OTIMIZAÇÃO DE FLUXO CURRICULAR

A seguinte seção tem como objetivo descrever o processo de modelagem do sistema, apresentando seus requisitos, usuários, caso de uso, diagramas e prototipagem.

3.1 Descrição Geral

O sistema apresentado tem como objetivo exibir as 10 melhores sugestões de fluxo curricular, geradas através da implementação de um algoritmo genético. Este algoritmo utiliza como dados de entrada as disciplinas ainda não cursadas por um determinado aluno e conta com um único módulo, cujo principal usuário é o setor de orientação acadêmica.

Por meio do sistema, o usuário pode:

- Visualizar as disciplinas de uma matriz curricular específica.
- Selecionar as disciplinas já cursadas pelo aluno que será acompanhado.
- Selecionar o tipo de semestre (se semestre par ou semestre ímpar) que será cursado.
- Acionar a geração das melhores sugestões de fluxo.

Como resposta, o sistema exibe as melhores sugestões de fluxo curricular baseadas nas disciplinas que ainda não foram cursadas.

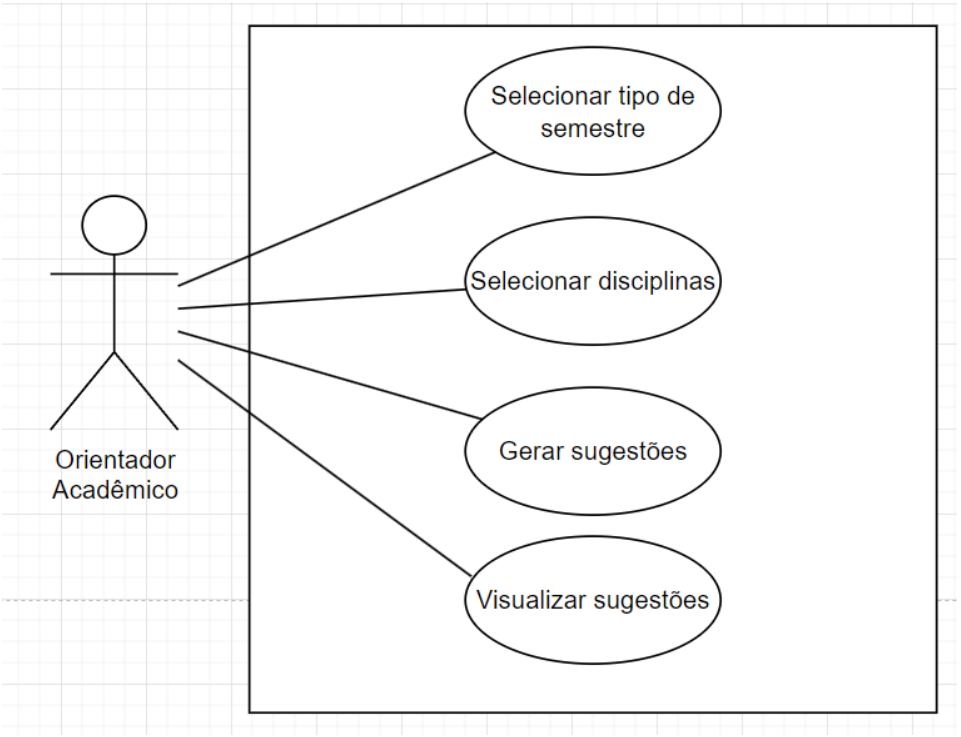
3.2 Requisitos Funcionais

Os requisitos funcionais do sistema incluem as seguintes funcionalidades:

- **Selecionar o tipo de semestre:** Permite a seleção do semestre que será considerado para a geração das sugestões de fluxo, podendo ser ele par ou ímpar.
- **Selecionar disciplinas cursadas:** Permite a seleção das disciplinas que o aluno já cursou e foi aprovado.
- **Gerar sugestões de fluxos:** Gera as melhores sugestões de fluxo curricular com base nas seleções feitas.
- **Visualizar sugestões:** Exibe as sugestões de fluxo geradas pelo sistema.

A Figura 03 exibe o diagrama de caso de uso que foi desenhado para demonstrar a funcionalidade da aplicação.

Figura 03 – Diagrama de caso de uso do sistema

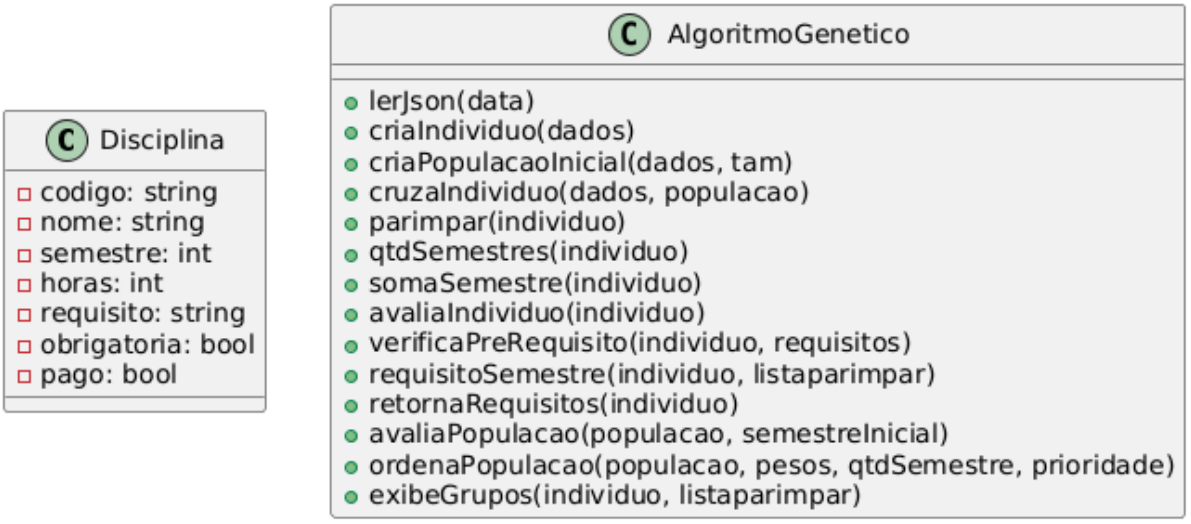


Fonte: elaborado pelo autor

3.3 Diagrama de classe

Os diagramas de classes exemplificados na Figura 04, apresentam a estrutura do sistema, destacando suas principais classes, atributos e métodos. A classe *Disciplina* é a responsável por definir os atributos das disciplinas que serão processadas pelo algoritmo, e o diagrama da classe *AlgoritmoGenetico*, armazena os métodos que compõem o algoritmo genético. O sistema foi modelado com base no objetivo de gerar as melhores sugestões de fluxo curricular para os alunos, utilizando um algoritmo genético.

Figura 04 – Diagrama de classes do algoritmo



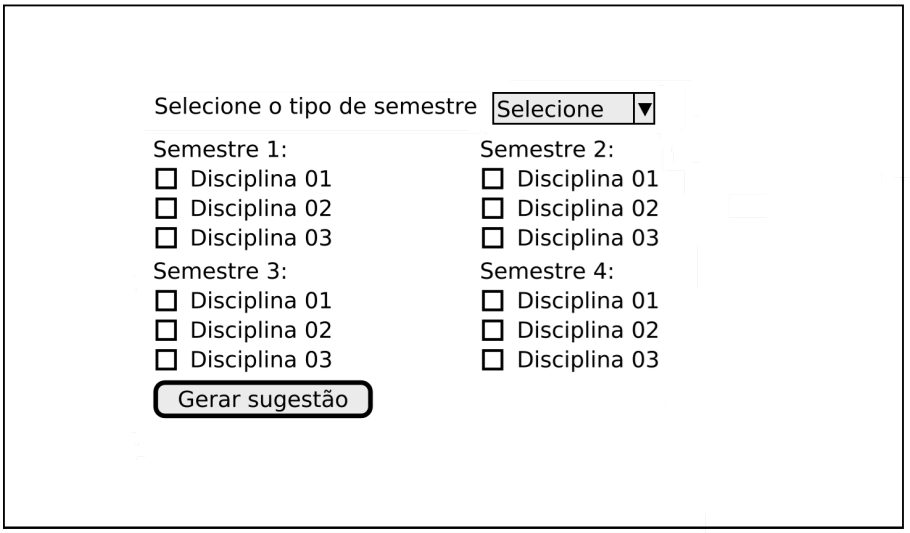
Fonte: elaborado pelo autor

3.4 Prototipagem das telas

Nesta seção serão apresentados os *wireframes* desenvolvidos para as principais telas do sistema. Esses *wireframes* servem como um guia visual para o desenvolvimento da interface do usuário, destacando a disposição dos elementos e as funcionalidades disponíveis.

A Figura 05 apresenta a tela inicial do sistema, onde é possível selecionar um tipo de semestre (através de um *select* com a opção de ímpar ou par), as disciplinas cursadas (através de um elemento de *checkbox*) e gerar as sugestões (através de um botão com texto).

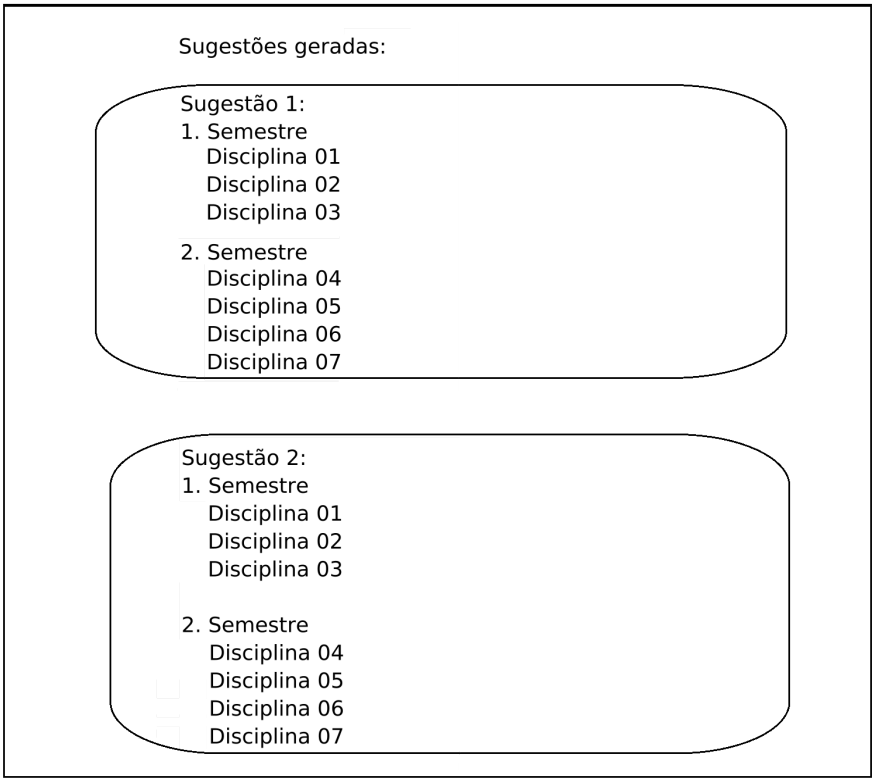
Figura 05 – Wireframe da tela inicial do sistema



Fonte: elaborado pelo autor

A Figura 06 mostra o *wireframe* da tela de resultados que foram gerados pelo algoritmo, sendo subdividido pelas sugestões, semestres e suas respectivas disciplinas.

Figura 06 – Wireframe da exibição das sugestões geradas pelo algoritmo.



Fonte: elaborado pelo autor

Para fins de documentação, a Figura 07 agrupa os elementos de *select* e botões usados no wireframe do sistema.

Figura 07 – Wireframe dos elementos de select e botões.

Fonte: elaborado pelo autor

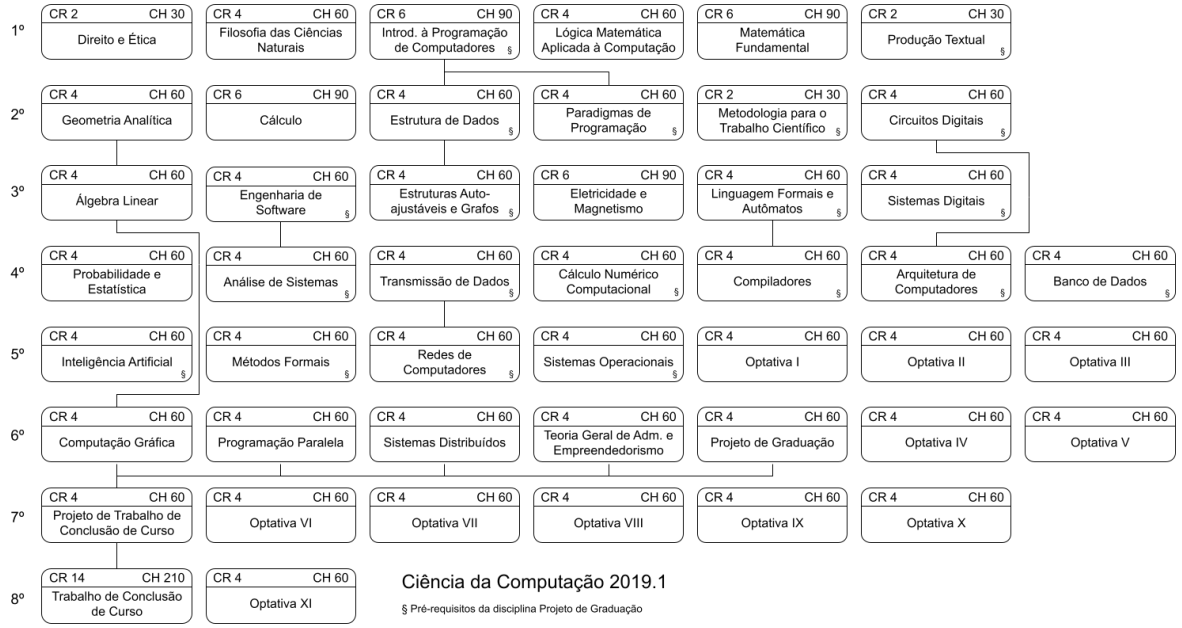
4 PROVA DE CONCEITO

Neste capítulo serão mostrados os resultados obtidos com a implementação do algoritmo genético para a geração de sugestões de fluxos curriculares no curso de Ciência da Computação da UERN Natal.

4.1 Dados do curso

Como visto anteriormente, existem altos índices de evasão nos cursos de nível superior do país, sobretudo nas áreas de computação. Para exemplificar e analisar tal situação, escolheu-se o curso de Ciência da Computação ofertado pelo campus da UERN-Natal, na modalidade de Bacharelado. O curso citado possui carga horária total de 3.200 horas e período estimado de conclusão entre 4 (previsto) a 6 anos (máximo). Atualmente existem duas matrizes ativas, uma com 36 disciplinas obrigatórias e 11 optativas, para ingressantes até 2022 e que pode ser vista na Figura 08.

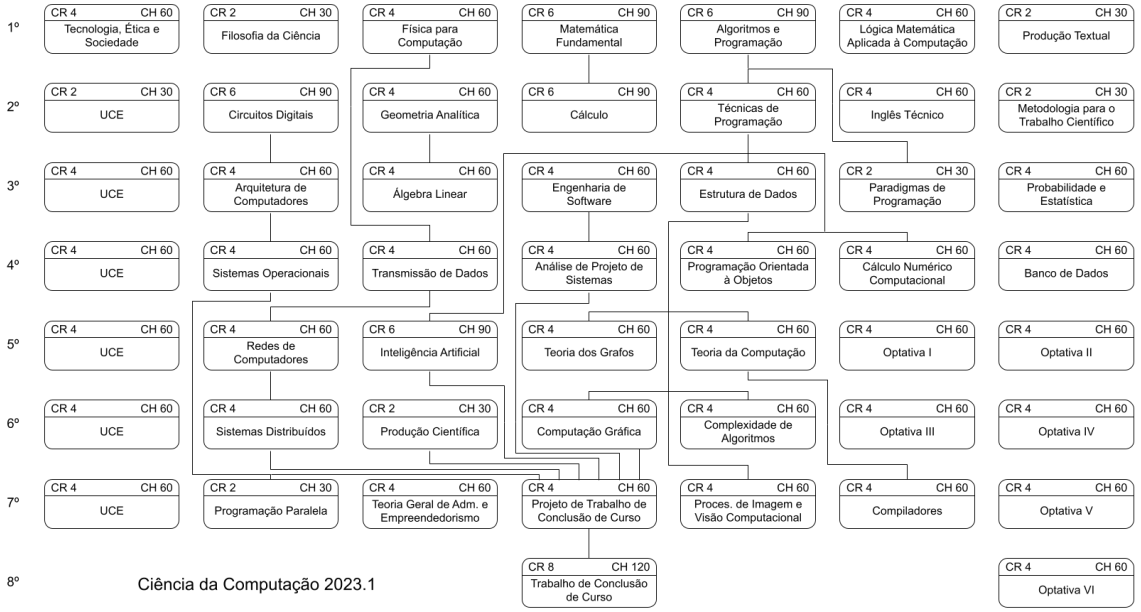
Figura 08 – Matriz curricular para ingressantes até o ano de 2022.



Fonte: Universidade do Estado do Rio Grande do Norte, 2024

A outra grade ativa é para os alunos ingressantes a partir do ano de 2023 e conta com 39 obrigatórias, 6 optativas e 6 disciplinas de extensão, como pode ser visto na Figura 09.

Figura 09 – Matriz curricular para ingressantes após o ano de 2022.



Fonte: Universidade do Estado do Rio Grande do Norte, 2024

Em ambas as matrizes as disciplinas são majoritariamente ofertadas pelo período da manhã, mas é possível que, em casos especiais, disciplinas adicionais sejam oferecidas em horário inverso. Em relação ao número de ingressantes e de concluintes, é possível notar o mesmo padrão apresentado anteriormente, onde mais de 80% dos ingressantes não chegam a concluir a graduação do curso dentro do prazo previsto, como pode ser visto nos dados disponibilizados pelo INEP (2024), onde, dos 156 alunos que ingressaram entre 2013 e 2018, apenas 22 alunos chegaram a concluir o curso até o ano de 2022, representando apenas 14,10% de conclusão.

Além disso, os dados fornecidos nos permitem perceber que, apesar do tempo estimado pela universidade ser de 4 anos, apenas 3 dos 22 concluintes conseguiram se encaixar nessa duração esperada. Contudo, apesar de não formados no período estipulado, os alunos ingressantes não necessariamente deixaram a instituição ou o curso escolhido, pois é possível verificar que a quantidade de desistência nos anos seguintes não representa a totalidade dos alunos vinculados à instituição.

É possível visualizar essa informação ao analisar a Tabela 01, em que 36 alunos ingressaram no curso no ano de 2018, mas até o ano de 2022 (período ideal de integralização), apenas 18 alunos desistiram da formação e 2 concluíram a formação, o que significa que 16 alunos continuaram na universidade.

Tabela 01 – Dados dos ingressantes do ano de 2018 do curso de Ciência da Computação na UERN

Ano de Ingresso	Ano de Referência	Prazo de Integralização em Anos	Prazo de Acompanhamento do Curso em anos	Ano Máximo de Acompanhamento do Curso	Quantidade de Ingressantes no Curso	Quantidade de Permanência no Curso no ano de referência	Quantidade de Concluintes no Curso no ano de referência	Quantidade de Desistência no Curso no ano de referência	Quantidade de Falecimentos no Curso no ano de referência
2018	2018	5	8	2025	36	30	0	6	0
2018	2019	5	8	2025	36	20	0	10	0
2018	2020	5	8	2025	36	18	0	2	0
2018	2021	5	8	2025	36	18	0	0	0
2018	2022	5	8	2025	36	10	2	6	0

Fonte: INEP (2024)

Esse mesmo comportamento se repete nos anos seguintes, como pode ser notado na Tabela 02, onde dos 23 ingressantes no ano de 2017, apenas 13 desistiram e 1 estudante chegou a conclusão, enquanto os outros 13 permaneceram na instituição, ultrapassando o tempo de formação indicado como ideal.

Tabela 02 – Dados dos ingressantes do ano de 2017 do curso de Ciência da Computação na UERN

Ano de Ingresso	Ano de Referência	Prazo de Integralização em Anos	Ano de Integralização do Curso	Prazo de Acompanhamento do Curso em anos	Ano Máximo de Acompanhamento do Curso	Quantidade de Ingressantes no Curso	Quantidade de Permanência no Curso no ano de referência	Quantidade de Concluintes no Curso no ano de referência	Quantidade de Desistência no Curso no ano de referência	Quantidade de Falecimentos no Curso no ano de referência
2017	2017	5	2021	8	2024	23	20	0	3	0
2017	2018	5	2021	8	2024	23	18	0	2	0
2017	2019	5	2021	8	2024	23	10	0	8	0
2017	2020	5	2021	8	2024	23	9	1	0	0
2017	2021	5	2021	8	2024	23	9	0	0	0

Fonte: INEP (2024)

Na Tabela 03 é possível visualizar um período maior de acompanhamento, onde os 25 ingressantes do ano de 2015 são acompanhados até o ano de 2021, o período máximo de acompanhamento realizado pelo INEP e período máximo de conclusão do curso de Ciência da Computação. Ao analisar esse período mais prolongado, é possível perceber que a quantidade de concluintes não é proporcional a quantidade de ingressantes, pois dos 25 alunos ingressantes, somente 7 concluíram o curso dentro do período máximo estipulado (8 anos), sendo somente 2 deles no período ideal de 4 anos.

Tabela 03 – Dados dos ingressantes do ano de 2013 do curso de Ciência da Computação na UERN

Ano de Ingresso	Ano de Referência	Prazo de Integralização em Anos	Ano de Integralização do Curso	Prazo de Acompanhamento do Curso em anos	Ano Máximo de Acompanhamento do Curso	Quantidade de Ingressantes no Curso	Quantidade de Permanência no Curso no ano de referência	Quantidade de Concluintes no Curso no ano de referência	Quantidade de Desistência no Curso no ano de referência	Quantidade de Falecimentos no Curso no ano de referência
2013	2013	5	2017	8	2020	25	25	0	0	0
2013	2014	5	2017	8	2020	25	24	0	1	0
2013	2015	5	2017	8	2020	25	11	2	11	0
2013	2016	5	2017	8	2020	25	8	0	3	0
2013	2017	5	2017	8	2020	25	8	0	0	0
2013	2018	5	2017	8	2020	25	5	3	0	0
2013	2019	5	2017	8	2020	25	2	2	1	0
2013	2020	5	2017	8	2020	25	2	0	0	0
2013	2021	5	2017	8	2020	25	2	0	0	0

Fonte: INEP (2024)

4.2 Interface

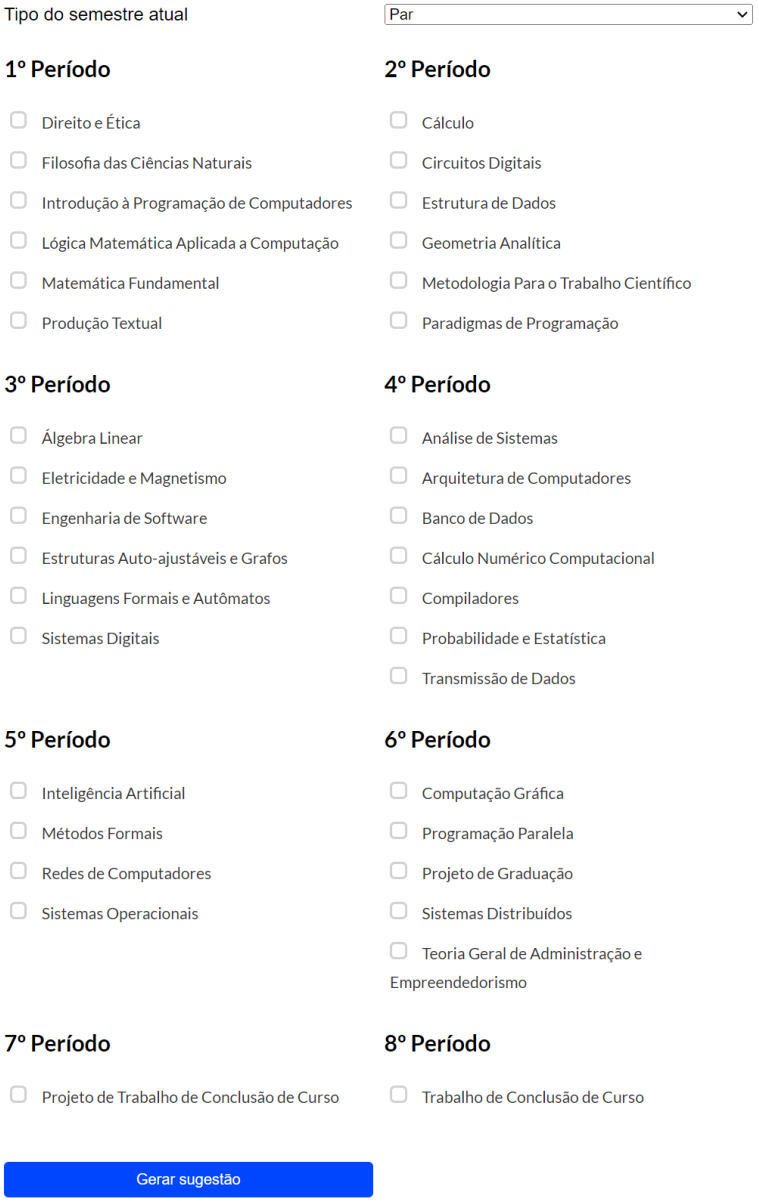
A intenção da interface é permitir que a orientação acadêmica possa visualizar as disciplinas de acordo com os semestres e que possa informar quais delas já foram cursadas pelo aluno acompanhado. Para gerar as sugestões, além de selecionar as disciplinas, é preciso que o orientador informe o semestre para o qual está gerando sugestões e clique na opção de “Gerar Resultados”. Feito isso, os dados selecionados são enviados para o servidor e as 10 melhores sugestões são exibidas em tela.

A interface web foi construída usando HTML, CSS e JavaScript. O HTML e CSS foram utilizados para deixar a interface mais agradável e simples ao usuário, mas não foi o foco do trabalho, visto que o foco é o algoritmo genético que será executado. Além disso, a maioria dos componentes exibidos em tela foi gerado de forma dinâmica, utilizando javascript. Isto foi feito para que os dados exibidos não sejam estáticos e para que possam se adaptar de acordo com a matriz curricular desejada.

Já a interação de cliente servidor foi criada utilizando o framework Flask e JavaScript (apenas para enviar os dados e receber a resposta do servidor).

Na Figura 10, é possível ver o resultado da tela inicial do projeto web, onde as disciplinas são exibidas de acordo com seu semestre sugerido na matriz curricular e é possível selecionar o tipo de período que deve ser gerado.

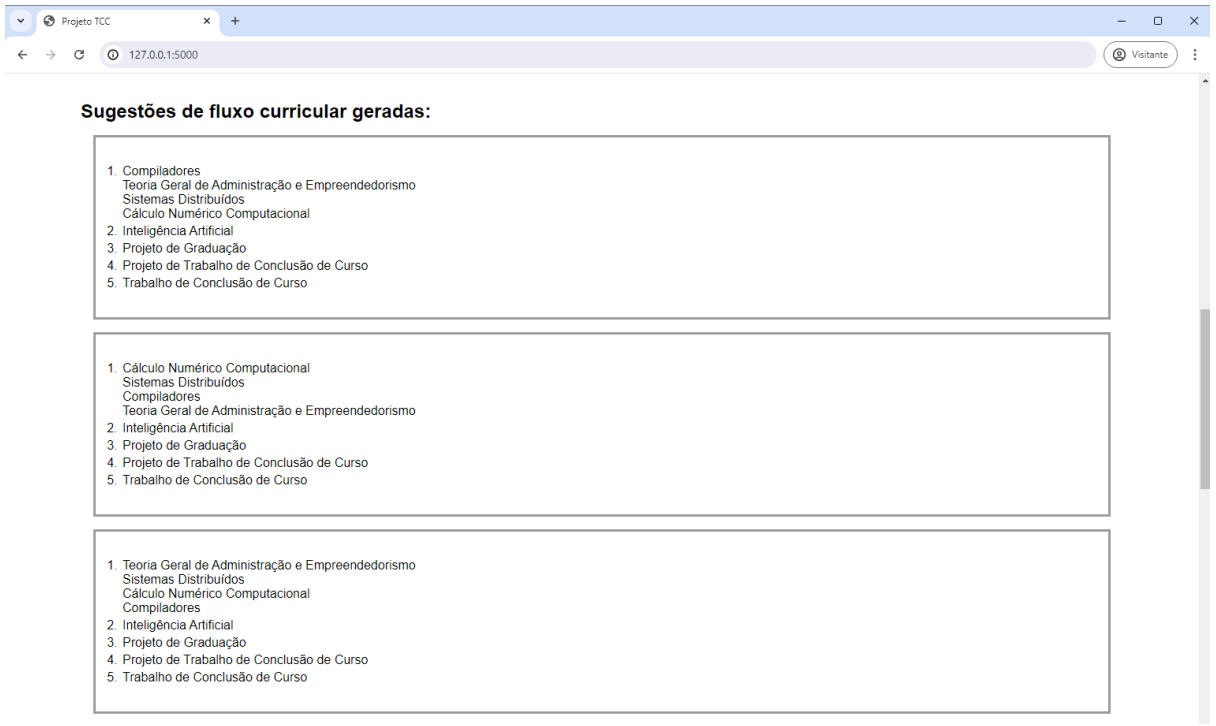
Figura 10 – Interface inicial do sistema



Fonte: elaborado pelo autor

Os dados selecionados nessa tela são usados como dados de entrada para o algoritmo genético, que os processa e retorna as 10 melhores possíveis soluções em formato de lista de listas. Após esse processamento, as listas retornadas são percorridas pelo javascript e exibidas em tela, separando cada solução possível e suas respectivas disciplinas por semestre, como pode ser visto na Figura 11.

Figura 11 – Resultado do algoritmo na interface



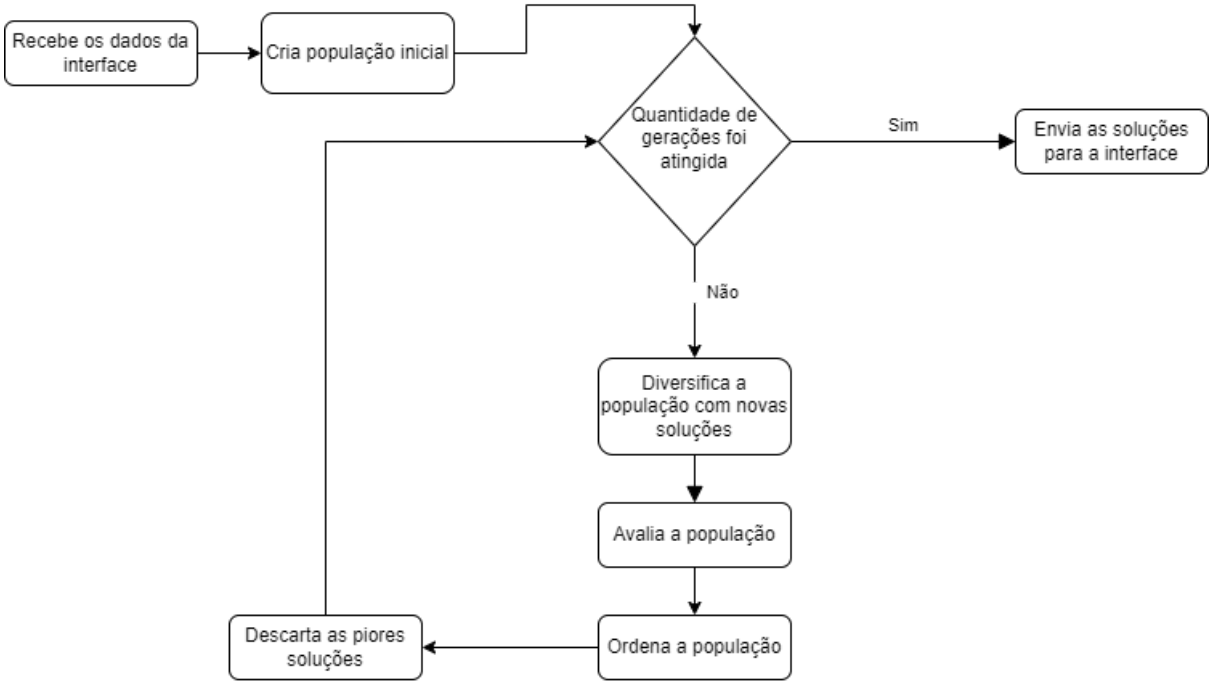
Fonte: elaborado pelo autor

4.3 Estratégia de resolução

Como visto nas seções anteriores, um algoritmo genético é uma técnica de busca e otimização inspirada nos princípios da evolução natural, como seleção, cruzamento e mutação. Aplicado ao problema de geração de sugestões de fluxo curricular, o algoritmo genético pode ser utilizado para explorar e identificar as melhores combinações possíveis de disciplinas a serem cursadas por um aluno, considerando suas disciplinas já pagas e o tipo de semestre. O processo começa com a criação de uma população inicial de possíveis soluções (fluxos curriculares). Em seguida, através de iterações sucessivas, as soluções mais promissoras são selecionadas para formar novas soluções (descendentes), que podem ser ainda ligeiramente modificadas por mutações. Esse ciclo de avaliação, seleção, cruzamento e mutação continua até que se encontre uma solução otimizada que atenda aos critérios estabelecidos.

De forma geral, o algoritmo apresentado neste trabalho pode ser exemplificado pelo fluxograma mostrado na Figura 12. Esta seção tem como objetivo descrever os dados trabalhados pelo algoritmo desenvolvido, as restrições consideradas para sua função de aptidão e os resultados encontrados com sua aplicação, levando em consideração a matriz curricular do curso de Ciência da Computação vigente até o ano de 2022.

Figura 12 – Fluxograma do funcionamento do algoritmo genético



Fonte: elaborado pelo autor

4.3.1 Descrição da Amostra

Para os testes do algoritmo e para análise dos resultados foram considerados 10 fluxos curriculares gerados pela orientação acadêmica e que refletem a situação dos alunos do curso de ciência da computação da UERN-NATAL. Na Tabela 04, é possível visualizar a quantidade de disciplinas que ainda não foram pagas, o ano de entrada de cada uma das amostras, sua situação com base no período máximo de integralização do curso (nivelado = ainda está dentro do período de integralização ideal, considerando o ano de 2024 como referência, e desnivelado = não está dentro do período ideal de integralização), o ano máximo de integralização do curso e o ano ideal para essa integralização.

Tabela 04 – Dados da amostra considerada para os testes

Aluno Exemplo	Disciplinas obrigatórias não cumpridas	Ano de entrada	Situação	Ano máximo de integralização	Ano ideal de integralização
Aluno 01	13	2019	Desnivelado	2025	2023
Aluno 02	17	2022	Nivelado	2025	2026
Aluno 03	9	2019	Desnivelado	2025	2023
Aluno 04	8	2019	Desnivelado	2025	2023
Aluno 05	24	2022	Nivelado	2028	2026

Aluno 06	15	2022	Nivelado	2028	2026
Aluno 07	6	2017	Desnivelado	2023	2021
Aluno 08	22	2019	Desnivelado	2025	2023
Aluno 09	14	2019	Desnivelado	2025	2023
Aluno 10	16	2018	Desnivelado	2025	2023

Fonte: elaborada pelo autor

4.3.2 Representação de uma solução

Através da interface web, as disciplinas que não foram pagas pelo aluno são enviadas para o servidor (onde o algoritmo é armazenado) em formato JSON. Após o recebimento dessas informações, a primeira coisa feita pelo algoritmo é a ‘transformação” dos dados em objetos do tipo Disciplina e a criação de uma lista com a junção de todas as disciplinas retornadas. O objeto gerado para cada uma das disciplinas retornadas pela interface possui os atributos que estão descritos na Tabela 05.

Tabela 05 – Atributos das disciplinas

Atributo	Tipo	Descrição	Exemplo
codigo	int	código que identifica a disciplina	35
nome	string	Nome da disciplina	Projeto de Graduação
semestre	int	Semestre que a disciplina é oferecida regularmente	6
horas	int	Carga horária	60
requisito	list [int]	Lista de pré-requisitos da disciplina (a lista é composta pelo código das disciplinas que são pré-requisito)	[1,2,3,4,5,6,7,8,9,10]
pago	int	Indica se a disciplina já cursada e	0

		finalizada pelo aluno	
obrigatoria	int	Indica se a disciplina é obrigatória	1

Fonte: elaborada pelo autor

Para realizar as operações e processamentos necessários, visando melhor desempenho e maior abstração dos dados, o algoritmo verifica e agrupa as disciplinas de acordo com o seu semestre, sendo 1 para ímpar e 2 para par. No código 03 é possível visualizar a função que realiza essa separação de acordo com o atributo *semestre* do objeto.

Código 03 – Função que representa as disciplinas de acordo com o tipo do semestre

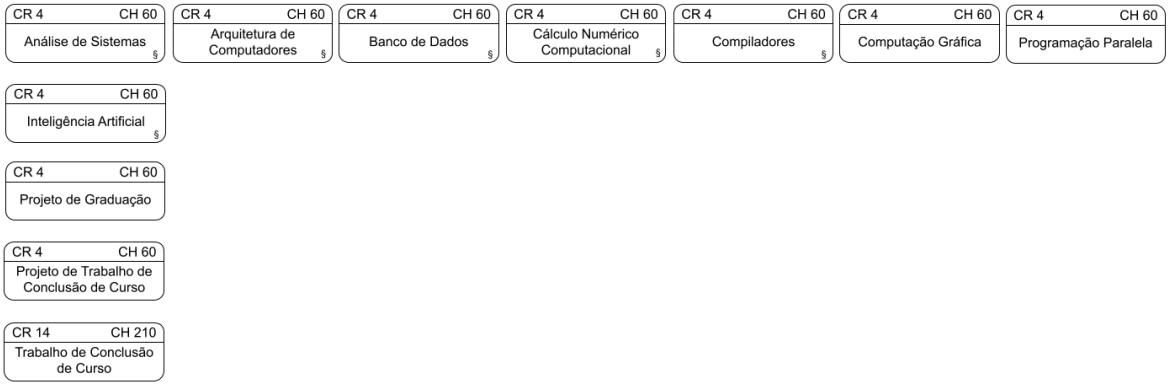
```
    def parimpar(individuo):
        disciplinas = []
        tam = len(individuo)
        for i in range(tam):
            if(individuo[i].semestre%2==0):
                disciplinas.insert(i, 2)
            else:
                disciplinas.insert(i, 1)
        return disciplinas
```

Fonte: elaborado pelo autor

Além disso, somente as disciplinas com a característica *obrigatoria* = 1, foram consideradas como entrada para o algoritmo. Essa característica indica que a disciplina deve ser cursada obrigatoriamente antes da entrega do Trabalho de Conclusão de Curso e que seu não cumprimento impossibilita o aluno de se formar. As matrizes do curso também possuem disciplinas optativas (*obrigatoria* = 0), mas essas disciplinas não foram consideradas no processamento do algoritmo, pois possuem um comportamento próprio em sua oferta.

Na Figura 12, é possível ver a representação de um indivíduo como lista de disciplinas, que seria o estado inicial dos dados que serão processados pelo algoritmo.

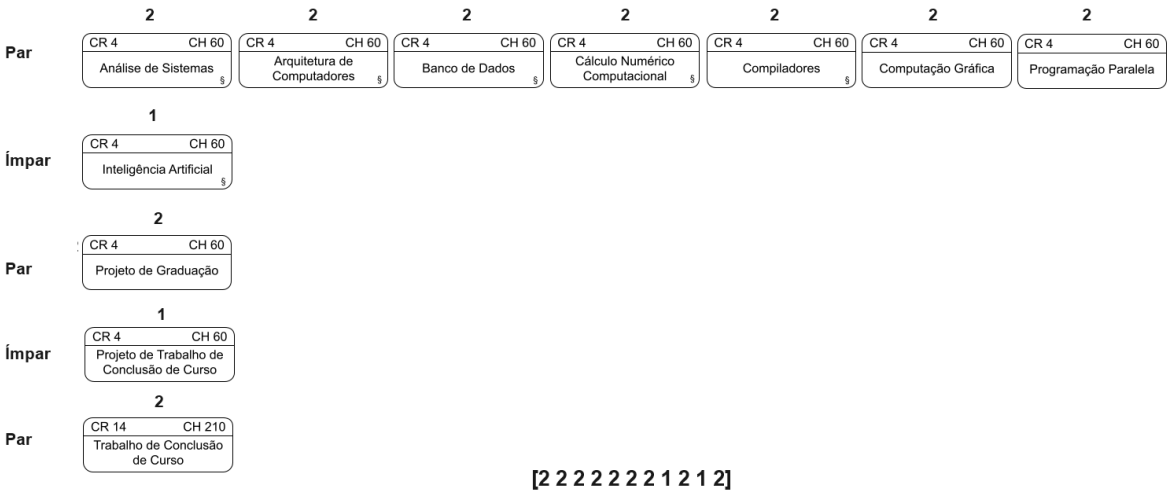
Figura 12 – Representação inicial de um indivíduo da população



Fonte: elaborado pelo autor

Na Figura 13, está representado o agrupamento das disciplinas de acordo com o tipo de semestre, se par ou ímpar. Esse agrupamento, em que o número 1 representa disciplinas de semestre ímpar e o número 2 representa as disciplinas de semestres pares, é o principal dado processado pelo algoritmo. Através desse agrupamento é realizado a contagem de semestre, quantidade de disciplinas por semestre e comparação de semestre atual.

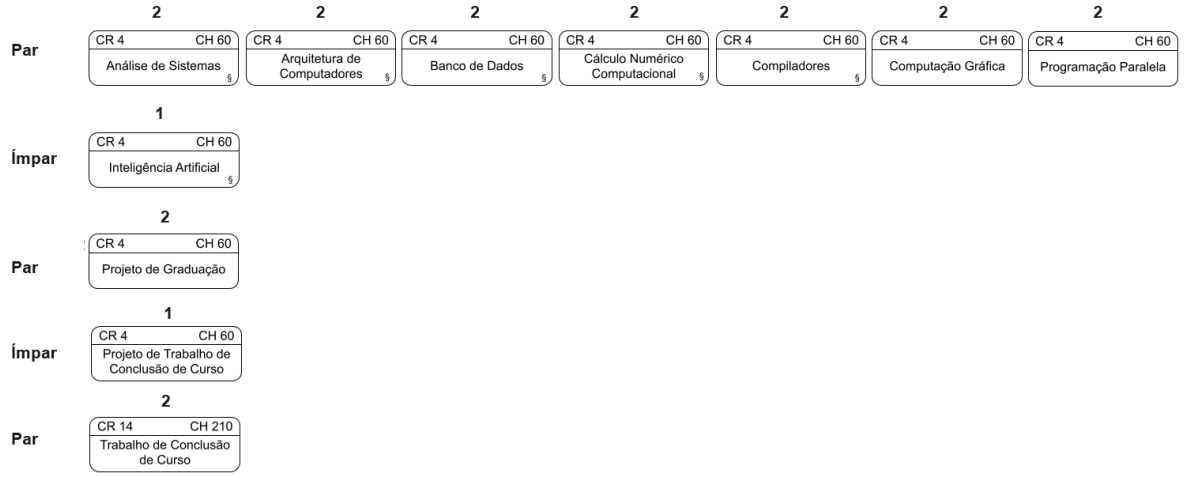
Figura 13 – Representação do agrupamento de um indivíduo por tipo de semestre



Fonte: elaborado pelo autor

A Figura 14, representa a saída de sugestões do algoritmo, onde cada lista interna de disciplinas representa um semestre.

Figura 14 – Representação final de uma solução



Fonte: elaborado pelo autor

4.3.3 População

A população inicial usada no algoritmo foi de 100 possíveis soluções e todas foram geradas de forma aleatória através de uma função de *random* do próprio python. Durante o processamento do algoritmo, essa população inicial foi também acrescida de mais 100 soluções, geradas através de uma abordagem que visa diversificar a população, semelhante à que vimos anteriormente como *crossover* (cruzamento).

Essas funções estão representadas no Código 04, onde a função *criaPopulcaoInicial()* recebe como parâmetro os dados do histórico do aluno e o tamanho da população inicial que será criada. O tamanho da população é usado para definir quantas vezes a função de criação de indivíduo (solução) será chamada. Já a função *cruzaIndividuo()* é responsável pela diversificação da população, para isso, ela recebe os dados do histórico do aluno e a população inicial que foi gerada. Após receber essas informações, ela cria novas soluções, levando em consideração o tamanho da população inicial.

Código 04 – Funções de criação e diversificação da população.

```
    def criaPopulacaoInicial(dados, tam):
        # Recebe um tamanho tam e retorna um conjunto de individuos
chamado populacao
        populacao = []
        for i in range(tam):
            individuo = AlgoritmoGenetico.criaIndividuo(dados)
            populacao.append(individuo)
        return populacao
```

```
    def cruzaIndividuo(dados, populacao):
        aux = []
        for _ in range(len(populacao)):
            linha = []
            linha = AlgoritmoGenetico.criaIndividuo(dados)
            aux.append(linha.copy())
        return aux
```

Fonte: elaborado pelo autor

Após essa diversificação, ocorre a avaliação da população e sua ordenação, que prioriza as soluções de menor peso de acordo com a função de aptidão que será apresentada posteriormente. Além disso, a cada iteração as 100 piores soluções (de maior peso) são descartadas, priorizando manter para a próxima geração somente as melhores soluções.

4.3.4 Função de aptidão

Conforme visto anteriormente, a função de aptidão tem por objetivo analisar as soluções e indicar se ela é apta o suficiente para contribuir para a próxima geração e gerar uma solução ótima. A função de aptidão aplicada neste trabalho busca encontrar a solução de menor peso e que respeite as restrições necessárias, sendo elas, em ordem de importância: cumprimento dos pré-requisitos, menor quantidade de semestres, quantidade de disciplinas em cada semestre e período atual do aluno (se período impar ou par). Cada restrição teve um peso atribuído e determinou a ordenação das melhores soluções geradas pelo algoritmo. A seguir, cada restrição será detalhada.

4.3.4.1 Análise dos pré-requisitos

Ao analisar a matriz curricular do curso e a forma como a orientação acadêmica definia as sugestões de fluxo para os alunos, foi visto que um dos maiores desafios na sugestão correta era o respeito aos pré-requisitos, visto que, mesmo que uma disciplina fosse ofertada, caso o aluno não pagasse o pré-requisito, ele não poderia cursá-la. Sendo assim, sugestões de fluxo que descumprissem pelo menos um pré-requisito já não levariam a uma sugestão ótima. Com base nessa constatação, o cumprimento dos pré-requisitos foi o primeiro parâmetro para ordenação dos melhores resultados.

Inicialmente, qualquer quebra de restrição aumentaria o peso da sugestão de fluxo e faria com que ele fosse penalizado em relação a outras sugestões. Contudo, como o algoritmo exibe sugestões considerando não somente o período atual, mas os próximos períodos, e a universidade dispõe da oferta de disciplinas de férias ou de disciplinas que podem ser pagas em outros campus, foi aberta uma flexibilização em relação a restrição de pré-requisito no mesmo semestre. Sendo assim,

considerando essas possibilidades, as restrições de pré-requisito foram consideradas conforme exemplificado na Tabela 06.

Tabela 06 – Restrições de pré-requisitos

Tipo	Restrição
0	Nenhuma restrição violada
1	Ocorrência de uma ou mais disciplinas que são pré-requisito no mesmo semestre do pré-requisito
2	Ocorrência de pré-requisito após a disciplina
3	Ocorrência de uma ou mais disciplinas que são pré-requisito no mesmo semestre do pré-requisito e ocorrência do pré-requisito após a disciplina

Fonte: elaborado pelo autor

Os tipos de restrição 1 e 2 possuem, cada um, uma função própria para seu cálculo, que estão exemplificadas no Código 05. Na função de verificação de pré-requisitos entre os semestres, chamada de *verificaPreRequisito()*, o indivíduo (solução) e os requisitos de suas disciplinas são enviados como parâmetro e a partir dessas informações, a função verifica se as disciplinas do semestre seguintes possuem algum pré-requisito que ainda não foi pago. Caso sim, um valor é adicionado à lista de violação de requisitos e o retorno da função é a soma desses valores.

Código 05 – Função de verificação de violação de pré-requisito entre semestres

```
   def verificaPreRequisito(individuo, requisitos):
        resultado = []
        subdisc = []
        codigos=[]
        for i in range(len(individuo)):
          codigos.append(individuo[i].codigo)
        tamsubreq = [len(item) if isinstance(item, list) else 0 for
item in requisitos]
        cont=0
        for i in range(len(codigos)):
            subdisc = codigos[:(i+1)]
            if (tamsubreq[i] != 0):
                aux = requisitos[i]
                for j in range(len(aux)):
```

```
                    for k in range(len(subdisc)):
                      if(aux[j]==0):
                        cont=1
                      else:
                        if(aux[j] == subdisc[k]):
                          cont+=1
                  if(cont==len(aux)):
                    resultado.append(0)
                  else:
                    resultado.append(1)
                  cont=0
          return sum(resultado)
```

Fonte: elaborado pelo autor

Além disso, a função de *requisitoSemestre()* verifica se houve violação de pré-requisitos dentro de um mesmo semestre, ou seja, se uma disciplina e seu pré-requisito estão juntos dentro de um mesmo semestre. O código 06, demonstra como é feito o agrupamento das disciplinas, para definição de quais disciplinas estão em cada semestre e depois é feito a verificação do pré-requisito.

Código 06 – Verificação de pré-requisitos no mesmo semestre

```
    def requisitoSemestre(individuo, listaparimpar):
        soma =0
        requisitos=[]
        codigos=[]
        for i in range(len(individuo)):
            requisitos.append(individuo[i].requisito)
            codigos.append(individuo[i].codigo)
        agrupados2 = []
        agrupados3 = []
        semestre_atual2 = []
        semestre_atual3 = []
        semestre_anterior = None
        for semestre, codigo, requisito1 in zip(listaparimpar,
codigos, requisitos):
            if(semestre!=semestre_anterior):
                if(semestre_atual2 and semestre_atual3):
                    agrupados2.append(semestre_atual2)
                    agrupados3.append(semestre_atual3)
                semestre_atual2 = [codigo]
                semestre_atual3 = requisito1.copy()
                semestre_anterior=semestre
```

```
            else:
                semestre_atual2.append(codigo)
                semestre_atual3.extend(requisito1)
        if semestre_atual2 and semestre_atual3:
            agrupados2.append(semestre_atual2)
            agrupados3.append(semestre_atual3)
        resultado = []
        for i in range(len(agrupados2)):
            lista2 = agrupados2[i]
            lista3 = agrupados3[i]
            soma=0
            for elem in lista3:
                if (elem != 0):
                    if (elem in lista2):
                        soma += 1
                    else:
                        soma += 0
            resultado.append(soma)
        return sum(resultado)
```

Fonte: elaborado pelo autor

Essa padronização de cumprimento dos pré-requisitos é utilizada na função de ordenação da população, denominada *ordenaPopulacao()*, que faz com que os indivíduos que não possuam restrição tendam a ficar no topo e serem selecionados para a próxima geração. Para isso, a função recebe como parâmetro a população a ser ordenada, a lista de pesos de cada solução da população, a quantidade de semestres de cada uma das soluções e a lista de restrições quebradas para cada solução. A partir desses dados, a população é ordenada através da função *sorted()* do python, primeiro considerando as restrições, depois a quantidade de semestres e, por fim, os pesos de cada solução da população, como pode ser visto no Código 07.

Código 07 – Função de ordenação da população

```
    def ordenaPopulacao(pop, pesos, qtdSemestre, normalizado):
        ordenados = list(zip(normalizado, qtdSemestre, pesos, pop))
        dados_ordenados = sorted(ordenados, key=lambda x: (x[0],
x[1], x[2]))
        _,_,_,popordenada = zip(*dados_ordenados)
        popordenada = list(popordenada)
        return popordenada
```

Fonte: elaborado pelo autor

4.3.4.2 Quantidade de semestres

Como dito anteriormente, a intenção do algoritmo é que o aluno conclua sua graduação no melhor tempo possível e dentro de suas possibilidades, sendo assim, a quantidade de semestres e a quantidade de disciplinas por semestre foi um importante tópico a ser avaliado em sua função de aptidão.

No caso da quantidade de semestres, foi criada uma função auxiliar que realizava a contagem de semestres em cada indivíduo, esta função pode ser vista com mais detalhes no Código 08, e que é chamada dentro da função responsável por avaliar cada indivíduo. O retorno dessa lista também foi usado como parâmetro para a função de ordenação, que tende a colocar no topo os indivíduos que não descumpriram nenhuma restrição de pré-requisito e que possuam a menor quantidade de semestres, possibilitando que os melhores resultados sejam levados para a próxima geração.

Código 08 – Função para soma de quantidade de semestres da solução

```
def qtdSemestres(individuo):
    lista = individuo
    cont_semestre = 1
    ultimo=lista[0]
    for i in lista[1:]:
        if(i!=ultimo):
            cont_semestre+=1
            ultimo=i
    return cont_semestre
```

Fonte: elaborado pelo autor

4.3.4.3 Quantidade de disciplinas por semestre

Assim como feito para o cálculo da quantidade de semestres, a quantidade de disciplinas por semestre foi realizada por uma função auxiliar chamada dentro da função de avaliação do indivíduo. No entanto, após analisar a matriz curricular e consultar a orientação acadêmica, algumas restrições adicionais e penalizações se fizeram necessárias.

Observando a matriz usada como base (matriz de 2019) é possível perceber que a carga horária máxima de um semestre, priorizando que o aluno só ocupe um turno de seu dia e levando em consideração o padrão de horários do campus, é de 420 horas, o equivalente a 7 disciplinas de 60 horas. Visando não ultrapassar esse valor, as sugestões que possuam mais de 7 disciplinas em um ou mais semestres foram penalizadas para que tendam a ser descartadas. Além disso, como o objetivo

é que o aluno possa concluir o curso em um tempo aceitável, sugestões com menos de 4 disciplinas em um ou mais semestres também foram penalizadas.

Na Tabela 07, é possível visualizar melhor as penalizações por disciplinas no semestre. As penalizações foram escolhidas com a intenção de penalizar semestres que possuam poucas disciplinas, pois isso tende a aumentar a quantidade de semestres das sugestões de fluxo. Após alguns testes, utilizando diversos valores, foi visto que os valores apresentados na tabela retornavam resultados satisfatórios e refletiam a necessidade do algoritmo de descartar sugestões com menos de 4 disciplinas e com mais de 7.

Tabela 07 – Penalização por quantidade de disciplinas no semestre

Quantidade de disciplinas	Penalização
1	Quantidade de disciplinas não pagas/2
2	6
3	4
> 7	Quantidade de disciplinas não pagas

Fonte: elaborado pelo autor

4.3.4.4 Restrição por período atual

A universidade usada como estudo de caso possui um padrão para ofertas das disciplinas, que consiste na separação de semestres pares e semestres ímpares. Ou seja, existem disciplinas que só podem ser pagas/ofertadas quando o semestre for par e existem disciplinas que só podem ser pagas/ofertadas quando o semestre for ímpar. Com isso, visando evitar que o algoritmo sugira fluxos e disciplinas que não possam de fato serem cursados, um peso foi atribuído de acordo com o semestre. Caso o primeiro semestre a ser sugerido seja definido como ímpar, então todas as sugestões iniciadas com uma disciplina de semestre par são penalizadas. O mesmo ocorre com semestres pares que sejam iniciados com disciplinas ímpares.

4.3.5 Análise dos resultados

Para a análise de desempenho do algoritmo foi considerada uma população inicial de 100 soluções (indivíduos), que foi processada e recombinada por 10.000 gerações, e cada amostra foi testada 10 vezes para cada tipo de semestre.

As médias apresentadas na Tabela 08, foram obtidas com o teste do algoritmo para o caso do semestre atual ser par.

Na coluna de “Tempo de processamento” é exibida a média de tempo que o algoritmo genético levou para processar as disciplinas e gerar as sugestões, ignorando o tempo de comunicação entre cliente-servidor.

A coluna de “Satisfação das restrições”, exibe a porcentagem de restrições que foram respeitadas (os tipos e restrições foram listados na Tabela 06), que teve 100% de satisfação, demonstrando que todas as amostras geraram sugestões válidas para o problema proposto.

A coluna de “Menor quantidade de semestres indicada pelo algoritmo”, exibe a menor quantidade de semestres que o algoritmo conseguiu gerar como possível solução para cada uma das amostras, levando em consideração os 10 testes realizados para cada uma.

Já a coluna de “Quantidade de semestres ideal”, mostra a quantidade de semestres mais otimizada, segundo avaliação manual do histórico da amostra, e é tida como parâmetro de melhor solução para o algoritmo.

Realizando a comparação entre a quantidade ideal de semestres e a menor quantidade gerada pelo algoritmo, é possível ver que somente 2 amostras não geraram soluções ideais, demonstrando que o algoritmo é capaz de encontrar respostas válidas e ideais na maioria dos testes.

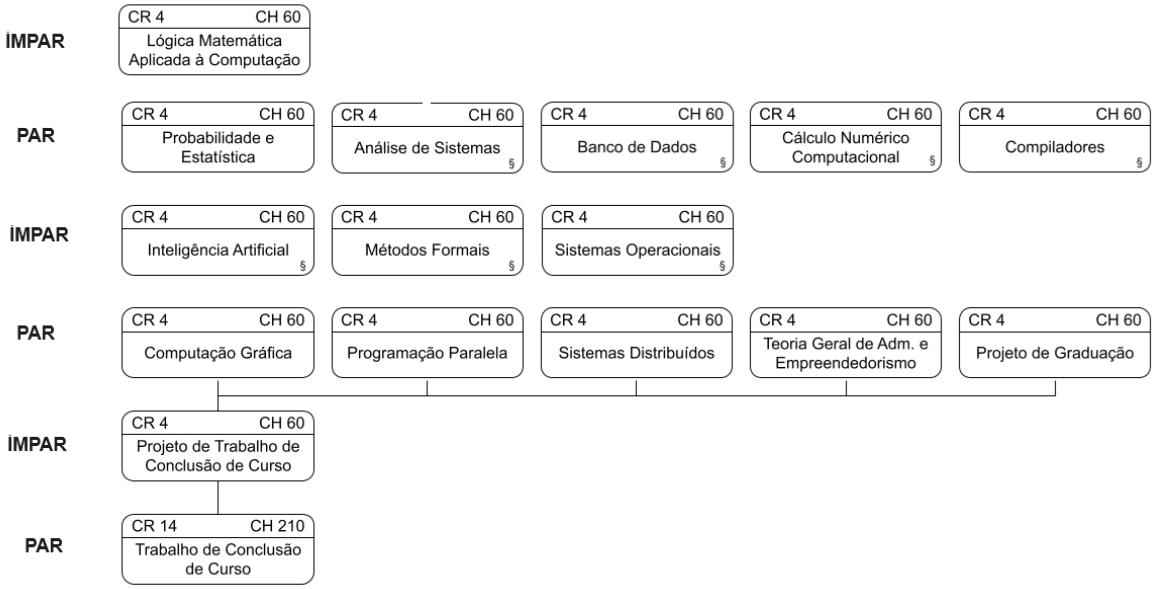
Tabela 08 – Resultados do algoritmo para semestres pares

Aluno Exemplo	**Tempo de processamento (em segundos)**	**Satisfação das restrições**	**Menor quantidade de semestres indicada pelo algoritmo**	**Quantidade de semestres ideal**
Aluno 01	89,73	100%	5	5
Aluno 02	162,63	100%	7	7
Aluno 03	93,22	100%	5	5
Aluno 04	69,91	100%	5	5
Aluno 05	230,96	100%	11	7
Aluno 06	132,53	100%	5	5
Aluno 07	45,93	100%	3	3
Aluno 08	216,37	100%	9	7
Aluno 09	131,84	100%	5	5
Aluno 10	150,30	100%	5	5

Fonte: elaborado pelo autor

Como exemplo de resultado ideal gerado pelo algoritmo, é possível citar o Aluno 10, que obteve a mesma quantidade de semestres proposta na solução ideal. Na Figura 15, é possível visualizar a lista de disciplinas não pagas por esse aluno e o seu tipo de semestre.

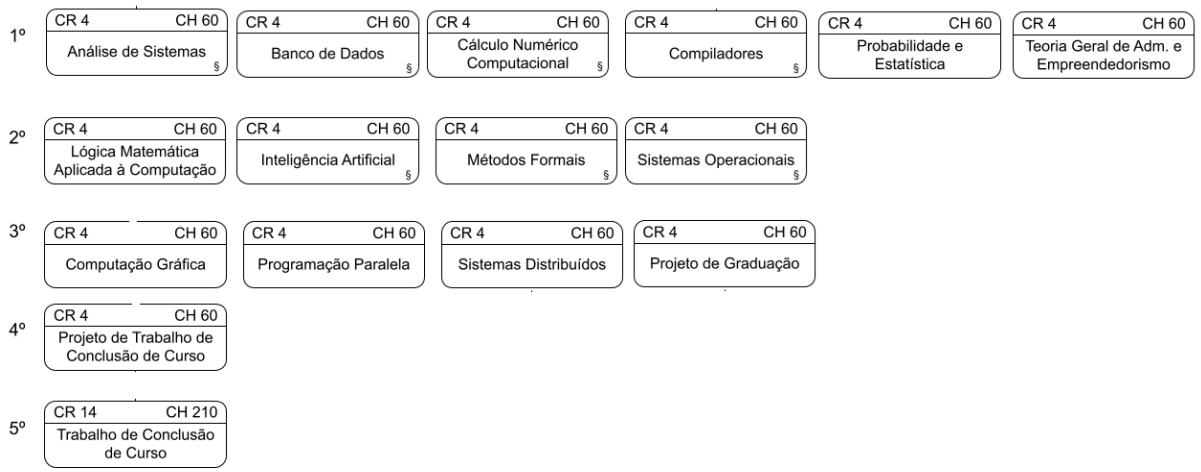
Figura 15 – Esquema das disciplinas não pagas pelo Aluno 10



Fonte: elaborado pelo autor

A quantidade de semestres ideal foi montada de forma manual, verificando a matriz curricular do aluno e simulando o processo realizado pela orientação acadêmica do curso. Como resultado, os semestres montados neste processo estão listados na Figura 16.

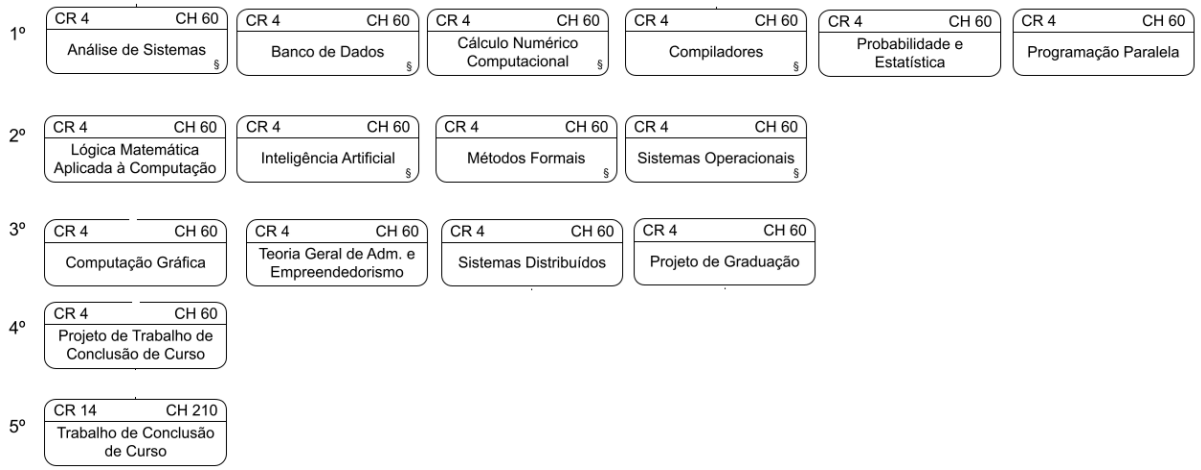
Figura 16 – Resultado ideal gerado de forma manual



Fonte: elaborado pelo autor

Na Figura 17, é possível visualizar o melhor resultado gerado pelo algoritmo genético, que é muito semelhante ao resultado ideal esperado, se diferenciando somente pelo período em que as disciplinas de Programação Paralela e Teoria Geral de Admin. e Empreendedorismo estão sendo alocadas.

Figura 17 – Melhor resultado gerado pelo algoritmo



Fonte: elaborado pelo autor

Já as médias apresentadas na Tabela 09, são referentes aos testes realizados para o semestre atual do tipo ímpar e apresentam violações de restrições que não foram notadas para o teste com semestre inicial par.

Das 10 amostras testadas, somente 8 delas apresentaram resultados válidos, sendo 5 deles ideais.

Tabela 09 – Resultados do algoritmo para semestres ímpares

Aluno Exemplo	Tempo de processamento (em segundos)	Satisfação das restrições	Quantidade de semestres indicados pelo algoritmo	Quantidade de semestres ideal
Aluno 01	109,84	100%	6	6
Aluno 02	148,21	100%	8	8
Aluno 03	70,26	66% (tipo 1)	4	6
Aluno 04	62,42	100%	4	4
Aluno 05	294,38	100%	14	6

Aluno 06	131,11	100%	6	5
Aluno 07	43,83	0% (tipo 3)	2	4
Aluno 08	210,01	100%	10	8
Aluno 09	118,90	100%	6	6
Aluno 10	136,32	100%	6	6

Fonte: elaborado pelo autor

Durante os testes foi visto que ao aumentar a quantidade de indivíduos na população inicial e a quantidade de gerações, os resultados retornados pelo algoritmo, em alguns casos, apresentaram sugestões melhores, mas com um custo computacional elevado. Tal comportamento já era esperado do algoritmo, que tende a demorar mais tempo para atingir o critério de parada (quantidade de gerações) de acordo com o tamanho da população/geração.

Além disso, ainda sobre a Tabela 08 e 09, é possível verificar que a quantidade de disciplinas a serem processadas também interfere no tempo de processamento, onde os alunos com mais disciplinas a serem pagas demoraram mais tempo para gerarem resultados.

Na Tabela 10, foram utilizados os dados do Aluno 05 para processamento do algoritmo, com diferentes tamanhos de população e quantidade de gerações. Ao analisar os resultados apresentados para esse aluno, fica nítido esse aumento de tempo e melhora das soluções com o aumento da população e da quantidade de gerações.

Tabela 10 – Resultados do Aluno 05 após aumento da população e quantidade de gerações

Quantidade de gerações	Tamanho da população	Tempo em segundos	Quantidade de semestres
10000	100	296,04	10
20000	500	2.283,66	8
30000	1000	6.882,40	8

Fonte: elaborado pelo autor

Como pôde ser visto nas tabelas anteriores, somente dois alunos tiveram as restrições não respeitadas para os casos testados com semestre ímpar. Em ambos os casos as sugestões não foram válidas devido a quantidade de disciplinas pares e ímpares que ainda não haviam sido pagas. Para exemplificar, na Tabela 11 estão as disciplinas não pagas desses dois alunos.

Tabela 11 – Disciplinas não pagas dos alunos que tiveram soluções que violaram os requisitos

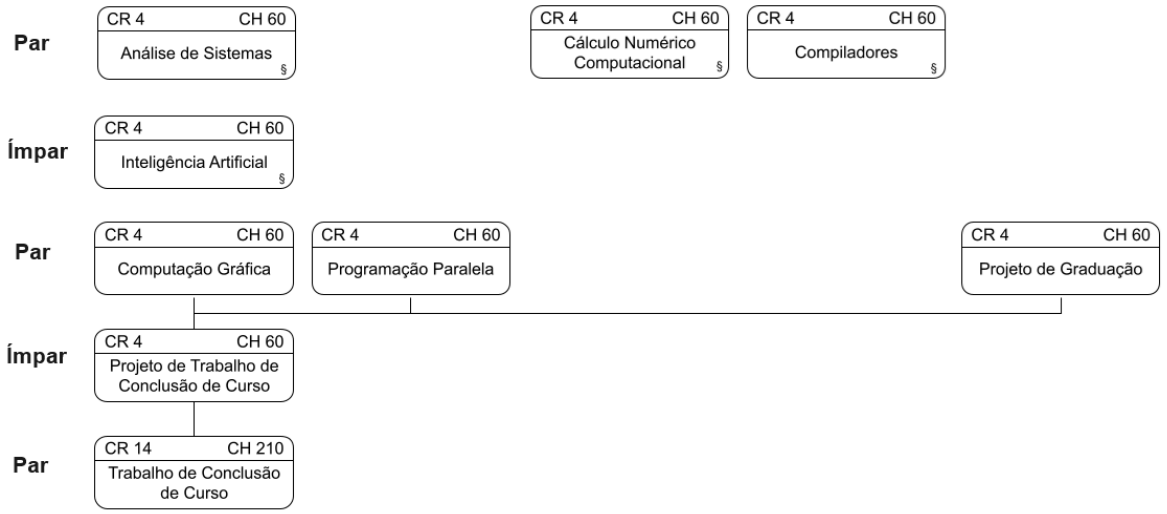
Aluno Exemplo	Disciplinas	Tipo de semestre

Aluno 03	Análise de Sistemas (par) Cálculo Numérico Computacional (par) Compiladores (par) Inteligência Artificial (ímpar) Computação Gráfica (par) Programação Paralela (par) Projeto de Graduação (par) Projeto de Trabalho de Conclusão de Curso (ímpar) Trabalho de Conclusão de Curso (par)	Ímpar
Aluno 07	Computação Gráfica (par) Programação Paralela (par) Projeto de Graduação (par) Sistemas Distribuídos (par) Projeto de Trabalho de Conclusão de Curso (ímpar) Trabalho de Conclusão de Curso (par)	Ímpar

Fonte: elaborado pelo autor

Conforme mostrado na Figura 18, o Aluno 03 não possui disciplinas ímpares o suficiente para gerar um semestre intermediário que separe os pré-requisitos da disciplina de Projeto de Graduação, por isso, para semestre ímpares não foi possível encontrar uma solução válida.

Figura 18 – Esquema das disciplinas não pagas e de seus pré-requisitos para o Aluno 03

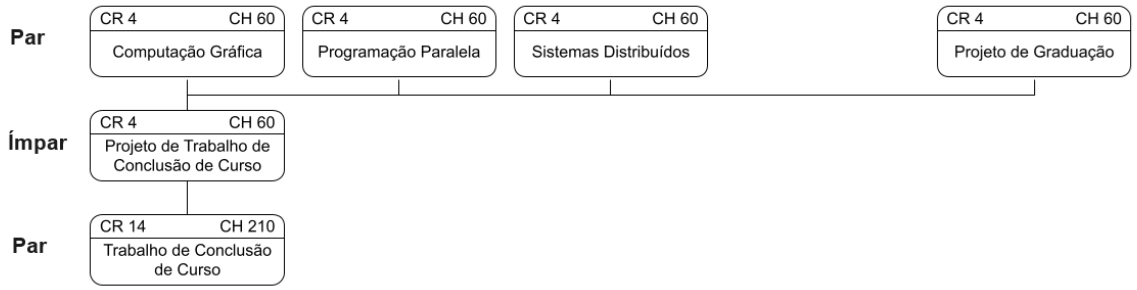


Fonte: elaborado pelo autor

Como pode ser visto na Figura 19, o Aluno 07 também não possui disciplinas o suficiente para um semestre intermediário. A única disciplina a ser paga para um semestre ímpar é Projeto de Trabalho de Conclusão de Curso, que tem como pré-requisito todas as disciplinas pares (exceto Trabalho de Conclusão de Curso),

sendo assim, também é impossível para o algoritmo identificar um semestre que não descumpra um dos requisitos.

Figura 19 – Esquema das disciplinas não pagas e de seus pré-requisitos para o Aluno 07



Fonte: elaborado pelo autor

Para chegar ao resultado ideal nesses casos, o que pode ser feito é a inserção das disciplinas optativas em um semestre intermediário entre os pré-requisitos (desde que o aluno não tenha pago todas as optativas ainda). Como essas disciplinas não foram tratadas nesta versão do algoritmo genético, então as restrições são inevitavelmente quebradas.

Outra opção possível seria a oferta de disciplinas em caráter especial. Com o auxílio do algoritmo, a orientação acadêmica já possui ciência da situação do aluno e pode solicitar com antecedência disciplinas especiais que supram as necessidades do aluno e diminuam o seu período de curso.

CONSIDERAÇÕES FINAIS

Nesta seção serão apresentadas as conclusões do trabalho e as sugestões de trabalhos futuros.

5.1 Conclusão

Após os dados apresentados na seção anterior, é possível afirmar que a proposta de implementação de um algoritmo genético para auxiliar a orientação acadêmica, através da sugestão de fluxos curriculares com base em pré-requisitos e tipo de semestre, foi cumprida e demonstrou resultados aceitáveis.

Além disso, as questões de pesquisa levantadas no início do trabalho podem ser respondidas da seguinte forma:

- Como otimizar o fluxo curricular a partir dos períodos iniciais do curso?

A otimização do fluxo curricular desde os períodos iniciais foi abordada pela capacidade do algoritmo de gerar sugestões válidas para 85% dos casos testados. Apesar de o algoritmo ainda não gerar soluções ideais para todos os casos, ele fornece sugestões válidas que podem ser consideradas no acompanhamento acadêmico dos estudantes iniciantes, indicando uma direção clara para a montagem do fluxo curricular.

- Como identificar qual o melhor fluxo curricular para alunos desnivelados considerando pré-requisitos e tipo de semestre, se ímpar ou par?

O algoritmo demonstrou eficiência em identificar os melhores fluxos curriculares para alunos desnivelados, considerando pré-requisitos e tipo de semestre. Ele gerou sugestões ideais para 90% dos casos em semestres pares e 50% dos casos em semestres ímpares. Para os casos onde as sugestões não foram 100% válidas, o algoritmo indicou qual restrição foi quebrada e sugeriu os melhores fluxos dentro das possibilidades apresentadas.

- Como auxiliar a orientação acadêmica na montagem dos horários e disciplinas ofertadas semestralmente?

A visão de sugestão de todos os semestres e a exibição de mais de uma opção de fluxo são informações que auxiliam significativamente a orientação acadêmica na tomada de decisão sobre a oferta de disciplinas. Essas sugestões permitem que a orientação acadêmica considere ofertas de disciplinas em caráter especial, disciplinas de férias ou prioridade de disciplinas optativas, facilitando a montagem dos horários e disciplinas ofertadas semestralmente.

Além disso, a possibilidade de refinar o algoritmo, aumentando o tamanho da população e a quantidade de gerações, mostra que é possível alcançar resultados ideais, embora isso aumente consideravelmente o tempo de execução do algoritmo.

Esse ajuste é especialmente relevante para alunos nos semestres iniciais do curso, que possuem mais disciplinas não pagas.

Em resumo, o algoritmo, em seu estado atual, ainda não gera soluções ideais para todos os casos, especialmente para alunos iniciantes. No entanto, ele consegue fornecer sugestões válidas que podem ser usadas pela orientação acadêmica para melhorar o acompanhamento dos estudantes e otimizar o planejamento curricular. Com melhorias no desempenho, espera-se que o algoritmo se torne uma ferramenta ainda mais eficaz para auxiliar na gestão acadêmica.

5.2 Trabalhos futuros

Como visto nas seções anteriores, o algoritmo apresentado ainda requer melhorias, principalmente em relação ao seu desempenho. Alguns pontos que podem ser abordados em trabalhos futuros para aprimorar esse quesito são:

- Implementar diferentes funções de seleção e cruzamento, avaliando qual delas apresenta melhores resultados para a convergência.
- Simplificar a representação das soluções, diminuindo a quantidade de características do objeto.
- Diminuir a quantidade de laços de repetição utilizados, revisando o código e aplicando boas práticas de codificação.
- Testar o algoritmo com diferentes grades e de diferentes cursos, não só os voltados para área de tecnologia.
- Permitir a inserção dos dados da matriz curricular pelo usuário, possibilitando a importação em outros formatos, como PDF e CSV.

Em relação às sugestões que violam restrições, uma possibilidade para que elas não ocorram é a adição das disciplinas optativas e de suas restrições. Como visto na seção de Prova de Conceito, os alunos que tiveram sugestões que violam as restrições de pré-requisito e requisito no mesmo semestre, são os alunos que não possuem disciplinas o suficiente para a distribuição correta dos semestres. Tendo isso em vista, alguns testes fora do algoritmo foram feitos e foi visto que com a adição das optativas não cursadas é possível encontrar resultados ideais para esses casos.

Além disso, também é possível visualizar melhorias no que diz respeito à interface e a apresentação das soluções. Atualmente, os tipos de restrição quebradas não são exibidos em tela, somente retornados para o algoritmo, o que requer que a orientação acadêmica precise revisar as sugestões antes de apresentá-las ao aluno. Futuramente, o ideal é que essas violações sejam informadas ao usuário.

REFERÊNCIAS

ASSOCIATION FOR COMPUTING MACHINERY. **What is Python? Python Documentation, 2023.** Disponível em: <https://docs.python.org/pt-br/3/faq/general.html#what-is-python/>. Acesso em: 14 jul. 2024.

BOYER, Omid et al. **Developing a model for the university course timetabling problem: a case study**. *Complexity*, vol. 2021, Article ID 9940866, 12 pages, 2021. Disponível em: <https://www.hindawi.com/journals/complexity/2021/9940866/>. Acesso em: 14 jul. 2023.

FREGONEIS, Jucelia Geni Pereira. **Estudo do Desempenho Acadêmico nos Cursos de Graduação dos Centros de Ciências Exatas e de Tecnologia da Universidade Estadual de Maringá: período 1995 - 2000**. Dissertação (Mestrado em Engenharia de Produção) - Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal de Santa Catarina (UFSC), Florianópolis: 2022. Disponível em: <http://repositorio.ufsc.br/xmlui/handle/123456789/84196/>. Acesso em: 22 jul. 2022.

GERAGHTY, John. RAZALI, Mohd Noraini. **Genetic Algorithm Performance with Different Selection Strategies in Solving TSP**. 2011. London p. 1134-1139. Disponível em: <https://www.iaeng.org/publication/WCE2011/WCE2011_pp1134-1139.pdf>. Acesso em: 14 jul. 2024.

GOLDBARG, M. C.; GOLDBARG, E. G.; LUNA, H. P. L. **Otimização combinatória e meta-heurísticas: algoritmos e aplicações.** 1. ed. Rio de Janeiro: Elsevier, 2016. ISBN 978-85-352-7812-5.

GOLDBERG, David E.. **Genetic Algorithms in Search, Optimization, and Machine Learning.** EUA: Addison-Wesley, 1989.

GIL, A. C. **Métodos e técnicas de pesquisa social**. São Paulo: Atlas, 1999 Disponível em: <https://ayanrafael.files.wordpress.com/2011/08/gil-a-c-mc3a9todos-e-tc3a9cnicas-de-pesquisa-social.pdf/>. Acesso em: 22 abr. 2022

GITHUB.**Octoverse: O estado do código aberto e a ascensão da IA em 2023.** Disponível em: <https://github.blog/2023-11-08-the-state-of-open-source-and-ai/>. Acesso em: 13 jul. 2024.

INEP. **CENSO DA EDUCAÇÃO SUPERIOR 2020: Principais Resultados**. Disponível em:

<https://download.inep.gov.br/educacao_superior/censo_superior/documentos/2020/tabelas_de_divulgacao_censo_da_educacao_superior_2020.pdf/>. Acesso em: 20 abr. 2022.

INEP. **Indicadores de Fluxo da Educação Superior.** Disponível em: <https://www.gov.br/inep/pt-br/acesso-a-informacao/dados-abertos/indicadores-educacionais/indicadores-de-fluxo-da-educacao-superior/>. Acesso em: 13 de jul. 2024.

LACERDA, E G M e CARVALHO, André Carlos Ponce de Leon Ferreira. **Introdução aos algoritmos genéticos**. 1999, Anais.. Rio de Janeiro: EntreLugar, 1999. Acesso em: 08 ago. 2023.

LUGER, George F. **Inteligência artificial**. 6º Edição. São Paulo: Pearson Education do Brasil, 2013.

MOZILLA DEVELOPER NETWORK. **Introduction to the DOM**. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction/>. Acesso em: 13 jul. 2024.

MOZILLA DEVELOPER NETWORK. **JavaScript**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/>. Acesso em: 13 jul. 2024.

PACHECO, Marco Aurélio Cavalcanti. **ALGORITMOS GENÉTICOS: PRINCÍPIOS E APLICAÇÕES.** Disponível em:<https://www.inf.ufsc.br/~mauro.roisenberg/ine5377/Cursos-ICA/CE-intro_apost.pdf/> Acesso em: 10 ago. 2023.

PyPI. **Flask**. Disponível em: <https://pypi.org/project/Flask/>. Acesso em: 13 jul. 2024.

Rodrigues, Leticia Francischini. **Um Modelo Para Otimização Da Matrícula Semestral De Cursos De Graduação Baseado Em Sequenciamento De Projetos.** Trabalho de conclusão de curso (Engenharia da Produção) – Universidade Federal de Ouro Preto, João Monlevade, 2015. Disponível em: <https://monografias.ufop.br/bitstream/35400000/295/1/MONOGRAFIA_ModeloOtimiza%c3%a7%c3%a3oMatr%c3%adcula.pdf/>. Acesso em: 20 abr. 2022.

SANTOS, Luiz Fernando Amaral dos. **Apostila Metodologia da Pesquisa Científica II.** Faculdade Metodista de Itapeva, 2006. Disponível em: <https://www.academia.edu/9596249/Faculdade_Metodista_de_Itapeva_APOSTILA_METODOLOGIA_DA_PESQUISA_CIENT%C3%8DFICA_II/>

SARAMAGO, Simone Pereira; STEFFEN JR., Valder. **Técnicas heurísticas de otimização aplicadas em engenharia.** Horizonte Científico, Uberlândia, v. 4, n. 2, p. 1-10, 2010. Disponível em: <https://seer.ufu.br/index.php/horizontecientifico/article/view/4425>. Acesso em: 14 jul. 2024.

SECCHI, Argimiro R. **Aula 01: Métodos de otimização.** Disponível em: <http://www2.peq.coppe.ufrj.br/Pessoal/Professores/Arge/COQ897/Naturais/aulas_piloto/aula1.pdf/>. Acesso em: 18 out. 2004.

SILVA, Elaine Cristina Reis; PEREIRA, Tábata Fernandes. **EVASÃO ESCOLAR NO ENSINO PÚBLICO SUPERIOR: UMA REVISÃO SISTEMÁTICA DA LITERATURA**. Curitiba, 24 jun. 2021. Disponível em: <https://www.brazilianjournals.com/index.php/BRJD/article/download/31822/pdf/>. Acesso em: 26 abr. 2022.

SILVA FILHO, Roberto Leal Lobo e *et al*. **A EVASÃO NO ENSINO SUPERIOR BRASILEIRO**. 2007. Instituto Lobo Para O Desenvolvimento da Educação, da Ciência e da Tecnologia. Disponível em: <https://www.scielo.br/j/cp/a/x44X6CZfd7hqF5vFNnHhVWg/?lang=pt&format=pdf/>. Acesso em: 27 abr. 2022.

STACKOVERFLOW. **2023 Developer Survey.** Disponível em: <https://survey.stackoverflow.co/2023/#most-popular-technologies-webframe/>. Acesso em: 13 jul. 2024.

TONTINI, Gérson; WALTER, Silvana Anita. **Pode-se identificar a propensão e reduzir a evasão de alunos?: ações estratégicas e resultados táticos para instituições de ensino superior.** Avaliação: Revista da Avaliação da Educação Superior (Campinas) [online], v. 19, n. 1, p. 89-110, mar. 2014. Disponível em: <https://doi.org/10.1590/S1414-40772014000100005/>. Acesso em: 29 abr. 2022. ISSN 1982-5765.

UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE. **Matriz Curricular**. Disponível em: <https://portal.uern.br/natal/dcc/matriz-curricular/>. Acesso em: 17 jul. 2024.

WOHLIN, C., AURUM, A. **Towards a decision-making structure for selecting a research design in empirical software engineering**. Empir Software Eng 20, 1427–1455 (2015). Disponível em: <https://doi.org/10.1007/s10664-014-9319-7/>. Acesso em: 29 abr. 2022.

APÊNDICE - Código do algoritmo genético gerado

```
import numpy as np
import random
from app import app
class Disciplina:
    def __init__(self, codigo, nome, semestre, horas, requisito,
obrigatoria, pago):
        self.nome = nome
        self.codigo = codigo
        self.semestre = semestre
        self.horas = horas
        self.requisito = requisito
        self.pago = pago
        self.obrigatoria = obrigatoria
    def __repr__(self):
       return f'{self.codigo}'
    def lerJson(data):
        disciplinas = []
        # Iterar sobre as disciplinas na matriz
        for disciplina_json in data['matriz']:
        # Criar objeto Disciplina para cada disciplina
            disciplina = Disciplina(
                disciplina_json['codigo'],
                disciplina_json['nome'],
                disciplina_json['semestre'],
                disciplina_json['horas'],
                disciplina_json['requisito'],
                disciplina_json['obrigatoria'],
                disciplina_json['pago'],
                disciplina_json['peso']
            )
            # Adicionar disciplina à lista
            disciplinas.append(disciplina)
        return disciplinas
    def __lt__(self, other):
        # Define como comparar duas instâncias de Disciplina
        return self.peso < other.peso
class AlgoritmoGenetico:
    def criaIndividuo(dados):
```

```
        todas = Disciplina.lerJson(dados)
        obrigatorias = []
        for i in range(len(todas)):
            if(todas[i].pago!=1):
                obrigatorias.append(Disciplina(todas[i].codigo,
todas[i].nome, todas[i].semestre, todas[i].horas,
todas[i].requisito, todas[i].obrigatoria, todas[i].pago))
        for i in range(len(obrigatorias)):
            for j in range(len(obrigatorias[i].requisito)):
                if(obrigatorias[i].requisito[j] != 0):
                    for k in range(len(todas)):
                        if(obrigatorias[i].requisito[j] ==
todas[k].codigo and todas[k].pago == 1):
                            obrigatorias[i].requisito[j] = 0
        for i in range(len(obrigatorias)):
            for j in range(len(obrigatorias[i].requisito)):
                if(len(obrigatorias[i].requisito)>1):
                    for elemento in obrigatorias[i].requisito[:]:
#cópia da lista para evitar problemas com o loop
                        if elemento == 0:

obrigatorias[i].requisito.remove(elemento)
                if(len(obrigatorias[i].requisito)==0):
                    obrigatorias[i].requisito.append(0)
        random.shuffle(obrigatorias)
        return obrigatorias
    def criaPopulacaoInicial(dados, tam):
        # Recebe um tamanho tam e retorna um conjunto de
individuos chamado populacao
        populacao = []
        for i in range(tam):
            individuo = AlgoritmoGenetico.criaIndividuo(dados)
            populacao.append(individuo)
        return populacao
    def cruzaIndividuo(dados, populacao):
        aux = []
        for _ in range(len(populacao)):
            linha = []
            linha = AlgoritmoGenetico.criaIndividuo(dados)
            aux.append(linha.copy())
```

```
        return aux
    def parimpar(individuo):
        disciplinas = []
        tam = len(individuo)
        for i in range(tam):
            if(individuo[i].semestre%2==0):
                disciplinas.insert(i, 2)
            else:
                disciplinas.insert(i, 1)
        return disciplinas
    def qtdSemestres(individuo):
        lista = individuo
        cont_semestre = 1
        ultimo=lista[0]
        for i in lista[1:]:
            if(i!=ultimo):
                cont_semestre+=1
                ultimo=i
        return cont_semestre
    def somaSemestre(individuo):
        soma = []
        cont=1
        for i in range(1, len(individuo)):
            if(individuo[i]==individuo[i-1]):
                cont +=1
            else:
                soma.append(cont)
                cont=1
        soma.append(cont)
        return soma
    def semestreAtual(individuo):
        inicio = 0
        if(individuo[0].semestre %2==0):
            inicio = 2
        else:
            inicio = 1
        return inicio
    def avaliaIndividuo(individuo):
        peso = 0
        # quantidade de semestres
```

```
        individuos = AlgoritmoGenetico.parimpar(individuo)
        qtd_semestre = AlgoritmoGenetico.qtdSemestres(individuos)
        # quantidade de disciplinas por semestre
        qtd_disc_semestre =
AlgoritmoGenetico.somaSemestre(individuos)
        for i in range(len(qtd_disc_semestre)):
            if qtd_disc_semestre[i] > 7:
                peso += len(individuos)
            elif qtd_disc_semestre[i] == 3:
                peso += 4
            elif qtd_disc_semestre[i] == 2:
                peso += 6
            elif qtd_disc_semestre[i] == 1:
                peso += len(individuos)//2
        peso += qtd_semestre
        return peso, qtd_semestre
    def verificaPreRequisito(individuo, requisitos):
        resultado = []
        subdisc = []
        codigos=[]
        for i in range(len(individuo)):
          codigos.append(individuo[i].codigo)
        tamsubreq = [len(item) if isinstance(item, list) else 0
for item in requisitos] # lista o tamanho da lista de requisitos
de cada disciplina (disciplinas com um requisito recebe zero tbm)
        cont=0
        for i in range(len(codigos)):
            subdisc = codigos[:(i+1)]
            if (tamsubreq[i] != 0):
                aux = requisitos[i]
                for j in range(len(aux)):
                  for k in range(len(subdisc)):
                    if(aux[j]==0):
                      cont=1
                    else:
                      if(aux[j] == subdisc[k]):
                        cont+=1
                if(cont==len(aux)):
                  resultado.append(0)
                else:
```

```
                    resultado.append(1)
                cont=0
        return sum(resultado)
    def requisitoSemestre(individuo, listaparimpar):
        soma =0
        requisitos=[]
        codigos=[]
        for i in range(len(individuo)):
            requisitos.append(individuo[i].requisito)
            codigos.append(individuo[i].codigo)
        agrupados2 = []
        agrupados3 = []
        semestre_atual2 = []
        semestre_atual3 = []
        semestre_anterior = None
        for semestre, codigo, requisito1 in zip(listaparimpar,
codigos, requisitos):
            if(semestre!=semestre_anterior):
                if(semestre_atual2 and semestre_atual3):
                    agrupados2.append(semestre_atual2)
                    agrupados3.append(semestre_atual3)
                semestre_atual2 = [codigo]
                semestre_atual3 = requisito1.copy()
                semestre_anterior=semestre
            else:
                semestre_atual2.append(codigo)
                semestre_atual3.extend(requisito1)
        if semestre_atual2 and semestre_atual3:
            agrupados2.append(semestre_atual2)
            agrupados3.append(semestre_atual3)
        resultado = []
        for i in range(len(agrupados2)):
            lista2 = agrupados2[i]
            lista3 = agrupados3[i]
            soma=0
            for elem in lista3:
                if (elem != 0):
                    if (elem in lista2):
                        soma += 1
                    else:
```

```
                        soma += 0
            resultado.append(soma)
        return sum(resultado)
    def retornaRequisitos(individuo):
        requisito = []
        for i in range(len(individuo)):
            requisito.append(individuo[i].requisito)
        return requisito
    def avaliaPopulacao(populacao, semestreInicial):
        peso_total = []
        peso_semestre_atual = []
        peso_pre_requisito = []
        semestres_par_impar = []
        peso_semestre_par_impar = []
        par_impar = []
        requisito = []
        normalizado = []
        semestre = []
        for i in range(len(populacao)):
            p, s =
AlgoritmoGenetico.avaliaIndividuo(populacao[i])
            peso_total.append(p)
            semestre.append(s)

requisito.append(AlgoritmoGenetico.retornaRequisitos(populacao[i]
))

semestres_par_impar.append(AlgoritmoGenetico.semestreAtual(popula
cao[i]))

par_impar.append(AlgoritmoGenetico.parimpar(populacao[i]))
            if semestres_par_impar[i] == semestreInicial:
                peso_semestre_par_impar.append(0)

peso_pre_requisito.append(AlgoritmoGenetico.verificaPreRequisito(
populacao[i], requisito[i]))

peso_semestre_atual.append(AlgoritmoGenetico.requisitoSemestre(po
pulacao[i], par_impar[i]))
            else:
```

```
                peso_semestre_par_impar.append(len(populacao[i]))
                peso_pre_requisito.append(len(populacao[i]))
                peso_semestre_atual.append(len(populacao[i]))
        normalizado = np.full(len(populacao), -1)
        for i in range(len(populacao)):
            if(peso_pre_requisito[i] == 0 and
peso_semestre_atual[i] == 0):
                normalizado[i]=0
            if(peso_pre_requisito[i] == 0 and
peso_semestre_atual[i] != 0):
                normalizado[i]=1
            if(peso_pre_requisito[i] != 0 and
peso_semestre_atual[i] == 0):
                normalizado[i]=2
            if(peso_pre_requisito[i] != 0 and
peso_semestre_atual[i] != 0):
                normalizado[i]=3
        return peso_total, peso_pre_requisito,
peso_semestre_atual, peso_semestre_par_impar, normalizado,
semestre
    def ordenaPopulacao(pop, pesos, qtdSemestre, normalizado):
        ordenados = list(zip(normalizado, qtdSemestre, pesos,
pop))
        dados_ordenados = sorted(ordenados, key=lambda x: (x[0],
x[1], x[2]))
        _,_,_,popordenada = zip(*dados_ordenados)
        popordenada = list(popordenada)
        return popordenada
    def exibeGrupos(individuo, listaparimpar):
        requisitos=[]
        codigos=[]
        for i in range(len(individuo)):
            requisitos.append(individuo[i].requisito)
            codigos.append(individuo[i].nome)
        agrupados2 = []
        agrupados3 = []
        semestre_atual2 = []
        semestre_atual3 = []
        semestre_anterior = None
        for semestre, codigo, requisito1 in (zip(listaparimpar,
```

```
codigos, requisitos)):
            if(semestre!=semestre_anterior):
                if(semestre_atual2 and semestre_atual3):
                    agrupados2.append(semestre_atual2)
                    agrupados3.append(semestre_atual3)
                semestre_atual2 = [codigo]
                semestre_atual3 = requisito1.copy()
                semestre_anterior=semestre
            else:
                semestre_atual2.append(codigo)
                semestre_atual3.extend(requisito1)
        if semestre_atual2 and semestre_atual3:
            agrupados2.append(semestre_atual2)
            agrupados3.append(semestre_atual3)
        return agrupados2
def ag(matriz, tam, geracoes, semestreInicial):
    pop = AlgoritmoGenetico.criaPopulacaoInicial(matriz, tam)
    pesos = []
    for i in range(geracoes):
        nova_populacao = AlgoritmoGenetico.cruzaIndividuo(matriz,
pop)
        populacao = pop + nova_populacao
        peso_total, peso_pre_requisito, peso_semestre_atual,
peso_semestre_par_impar, normalizado, qtdSemestre =
AlgoritmoGenetico.avaliaPopulacao(populacao, semestreInicial)
        pesos = [a + b + c for a, b, c in zip(peso_total,
peso_semestre_par_impar, qtdSemestre)]
        pop_ordenada =
AlgoritmoGenetico.ordenaPopulacao(populacao, qtdSemestre, pesos,
normalizado)
        metade =  len(populacao)//2
        pop = pop_ordenada[:metade]
    peso_total1, peso_pre_requisito1, peso_semestre_atual1,
peso_semestre_par_impar1, normalizado1, qtdSemestre1 =
AlgoritmoGenetico.avaliaPopulacao(pop, semestreInicial)
    # lista com o resultado da análise das restrições de cada
indivíduo
    # se for 0 - Nenhuma restrição violada
    # se for 1 - Restrição de disciplina pré-requisito no mesmo
semestre
```

```
    # se for 2 - Restrição de disciplinas pré-requisito em
semestres diferentes
    # se for 3 - Violação das restrições 1 e 2
    print("Restrições: ", normalizado1)
    resultados = []
    for i in range(10):
        resultados.append(AlgoritmoGenetico.exibeGrupos(pop[i],
AlgoritmoGenetico.parimpar(pop[i])))
    return resultados
```