



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO - CTC
SISTEMAS DE INFORMAÇÃO

ARTHUR MACHADO CAPAVERDE

GYMSYS: ALGORITMO GENÉTICO PARA OTIMIZAÇÃO DE AGENDAMENTOS EM
ACADEMIAS

Florianópolis
2025

ARTHUR MACHADO CAPAVERDE

GYMSYS: ALGORITMO GENÉTICO PARA OTIMIZAÇÃO DE AGENDAMENTOS EM
ACADEMIAS

Trabalho apresentado como requisito para
obtenção do título de Bacharel em Siste-
mas de Informação, no Centro Tecnológico
- CTC, da Universidade Federal de Santa
Catarina.

Orientador(a): Luciana de Oliveira Rech

Florianópolis
2025

RESUMO

Com a ascensão da computação evolucionária, os algoritmos genéticos ganharam espaço como uma forma efetiva de encontrar soluções satisfatórias para problemas complexos. Um ramo de aplicação dos algoritmos genéticos é na programação de grade de horários, onde são utilizados para otimizar o agendamento de tarefas ou pessoas em um conjunto de horários. Enquanto o uso de algoritmos genéticos em otimização de agendamentos é extensivamente documentado, cenários como academias ainda foram pouco explorados e requerem mais investigações práticas para avaliar sua eficácia em contextos como a alocação de horários de aulas, otimização de equipamentos ou a personalização de planos de treino para os alunos. Este projeto tem como objetivo otimizar a alocação de alunos em horários considerando os equipamentos utilizados em seus treinos, reduzindo o tempo de espera por equipamentos em uso ao evitar alocar alunos que utilizam o mesmo equipamento no mesmo horário. O algoritmo genético deve, em tempo hábil, gerar soluções que otimizem o aproveitamento dos recursos da academia.

Palavra-chave: Computação evolucionária. Algoritmos genéticos. Programação ótima de horários.

ABSTRACT

With the rise of evolutionary computing, genetic algorithms have gained prominence as an effective way to find satisfactory solutions to complex problems. One application of genetic algorithms is in scheduling, where they are used to optimize the allocation of tasks or people across time slots. Although the use of genetic algorithms in scheduling optimization is extensively documented, application scenarios such as gyms have been less explored and require further practical investigation to assess their effectiveness in contexts such as class scheduling, equipment optimization, or the customization of workout plans for students. This project aims to optimize the allocation of students to time slots, considering the equipment used in their workouts, reducing waiting times for equipments in use by avoiding scheduling students who use the same equipment at the same time. The genetic algorithm should, in a timely manner, generate solutions that optimize the gym's resource utilization.

Keywords: Evolutionary computing. Genetic Algorithms. Optimal time schedule.

LISTA DE FIGURAS

Figura 1 – Correlação entre Net Promoter Score e disponibilidade de equipamentos. Adaptado de GYMetrix (Gymetrix, s.d., p. 2).	11
Figura 2 – Ciclo de execução de um algoritmo genético. Adaptado de (Linden, 2012, p. 27).	15
Figura 3 – Pontos de corte de um cromossomo. Adaptado de (Linden, 2012, p. 40).	19
Figura 4 – Modelagem do Cromossomo	32
Figura 5 – Arquitetura do software	33
Figura 6 – <i>Crossover</i> entrelaçado parte 1	35
Figura 7 – <i>Crossover</i> entrelaçado parte 2	35
Figura 8 – Interface gráfica da simulação com academia vazia	39
Figura 9 – Interface gráfica da simulação com academia em uso	39
Figura 10 – Desempenho dos métodos de seleção na simulação do cenário A .	44
Figura 11 – Execução do algoritmo guloso	48
Figura 12 – Desempenho dos algoritmos na simulação do cenário D	50

LISTA DE TABELAS

Tabela 1 – Comparação entre trabalhos relacionados	29
Tabela 2 – Especificações detalhadas do sistema utilizado nos testes.	40
Tabela 3 – Cenários avaliados nos experimentos realizados	41
Tabela 4 – Resultados dos experimentos para o cenário A.	43
Tabela 5 – Resultados dos experimentos para o cenário B.	45
Tabela 6 – Resultados dos experimentos para o cenário C.	46
Tabela 7 – Resultados dos experimentos para o cenário D.	49

LISTA DE SIGLAS

BRKGA	<i>Biased Random-key Genetic Algorithm</i> (Algoritmo Genético de Chaves Aleatórias Viciadas)
BRKGA2p	<i>Biased Random-Key Genetic Algorithm with 2-point crossover</i> (Algoritmo Genético de Chaves Aleatórias Viciadas com crossover de 2 pontos)
BRKGAMp	<i>Biased Random-Key Genetic Algorithm with Multi-point crossover</i> (Algoritmo Genético de Chaves Aleatórias Viciadas com crossover de múltiplos pontos)
GA2p	<i>Genetic Algorithm with 2-point crossover without elitism</i> (Algoritmo Genético com crossover de 2 pontos sem elitismo)
GA2pE	<i>Genetic Algorithm with 2-point crossover with elitism</i> (Algoritmo Genético com crossover de 2 pontos com elitismo)
GAMp	<i>Genetic Algorithm with Multi-point crossover without elitism</i> (Algoritmo Genético com crossover de múltiplos pontos sem elitismo)
GAMpE	<i>Genetic Algorithm with Multi-point crossover with elitism</i> (Algoritmo Genético com crossover de múltiplos pontos com elitismo)
RKGA	<i>Random-key Genetic Algorithm</i> (Algoritmo Genético de Chaves Aleatórias)
RKGA2p	<i>Random-Key Genetic Algorithm with 2-point crossover</i> (Algoritmo Genético de Chaves Aleatórias com crossover de 2 pontos)
RKGAMp	<i>Random-Key Genetic Algorithm with Multi-point crossover</i> (Algoritmo Genético de Chaves Aleatórias com crossover de múltiplos pontos)

SUMÁRIO

1	INTRODUÇÃO	9
1.1	JUSTIFICATIVA	10
1.2	OBJETIVOS	11
1.2.1	Objetivos específicos	12
1.3	MÉTODOS DE PESQUISA	12
1.4	ESTRUTURA DO TRABALHO	12
2	FUNDAMENTAÇÃO TEÓRICA	14
2.1	COMPUTAÇÃO EVOLUCIONÁRIA	14
2.2	ALGORITMOS GENÉTICOS	14
2.2.1	Inicialização	15
2.2.2	Codificação de Genes	15
2.2.3	População	16
2.2.3.1	Elitismo	16
2.2.4	Teste <i>Fitness</i>	16
2.2.5	Métodos de Seleção	16
2.2.5.1	Seleção por torneio	17
2.2.5.2	Seleção por roleta	17
2.2.5.3	Seleção por <i>ranking</i>	18
2.2.6	<i>Crossover</i>	18
2.2.6.1	<i>Crossover</i> um ponto	18
2.2.6.2	<i>Crossover</i> uniforme	19
2.2.7	Mutação	19
2.2.8	Critério de Parada	20
2.3	PROGRAMAÇÃO PARALELA	20
2.3.1	<i>Mutexes</i>	21
2.3.2	Semáforos	21
2.4	CONSIDERAÇÕES FINAIS	21
3	ESTADO DA ARTE	23
3.1	<i>A GENETIC ALGORITHM FOR THE REAL-WORLD UNIVERSITY COURSE TIMETABLING PROBLEM</i>	23
3.2	<i>RESEARCH AND APPLICATIONS OF SHOP SCHEDULING BASED ON GENETIC ALGORITHMS</i>	24
3.3	<i>COMPARATIVE STUDY OF GENETIC ALGORITHMS FOR THE SCHOOL TIMETABLING PROBLEM</i>	25
3.4	COMPARAÇÕES	27
3.5	CONSIDERAÇÕES FINAIS	29

4	PROPOSTA	30
4.1	MODELAGEM DO CROMOSSOMO	31
4.2	ARQUITETURA DO ALGORITMO GENÉTICO	32
4.3	CICLO DE EXECUÇÃO DO ALGORITMO	33
4.3.1	Inicialização	33
4.3.2	Seleção	34
4.3.3	Crossover entrelaçado	34
4.3.4	Crossover um ponto	36
4.3.5	Mutação	36
4.3.6	Teste <i>Fitness</i>	36
4.3.7	Critério de Parada	37
4.4	CONSIDERAÇÕES FINAIS	37
5	ANÁLISE DE EXPERIMENTOS	38
5.1	SOBRE A SIMULAÇÃO	38
5.2	ESPECIFICAÇÕES DO SISTEMA	40
5.3	RESULTADOS	41
5.3.1	Cenário A - Métodos de Seleção	42
5.3.2	Cenário B - <i>Crossover</i>	44
5.3.3	Cenário C - Número de Gerações	46
5.3.4	Cenário D - Algoritmo Genético Vs. Algoritmo Guloso	47
6	CONCLUSÕES	51
6.1	SUGESTÃO PARA TRABALHOS FUTUROS	51
	REFERÊNCIAS	53
A	ARTIGO ACADÊMICO	55
B	CÓDIGO FONTE	70

1 INTRODUÇÃO

A Computação Evolucionária é uma área da Inteligência Artificial inspirada nos processos de evolução biológica para resolver problemas complexos. Ela aplica princípios da seleção natural para encontrar soluções em espaços de pesquisa grandes. Dentro desse contexto, os Algoritmos Genéticos emergem como uma das técnicas mais utilizadas devido à sua capacidade de encontrar soluções satisfatórias para problemas de otimização e combinação em tempo hábil.

Estes algoritmos funcionam simulando um processo evolutivo, no qual uma população de possíveis soluções para o problema é iterativamente modificada e aprimorada por meio de operações como seleção, *crossover* e mutação. A seleção é responsável por escolher os indivíduos mais aptos para reprodução, gerando a próxima geração. O *crossover* permite a troca de informações entre as soluções, criando novos indivíduos com características combinadas. Por fim, a mutação altera aleatoriamente as características de uma solução, promovendo diversidade e ajudando a evitar a convergência prematura em soluções subótimas.

Segundo a literatura, os algoritmos genéticos têm diversas aplicações, como otimização de rotas, problemas de empacotamento e criação de grades de horários (Jong, 1997). Este trabalho propõe o cenário de uma academia de treino físico, com recursos limitados, como espaço e máquinas de treino. Nessa academia, os alunos devem reservar previamente o horário de seu treino para evitar a superlotação do ambiente. No entanto, mesmo com o número de alunos por horário limitado, a quantidade de máquinas disponíveis pode ser insuficiente, gerando competição pelo uso de um equipamento. Ao permitir que os alunos informem uma lista de horários em que têm disponibilidade para treinar, é possível identificar um problema de otimização de grade de horários, com o objetivo de reduzir o número de alunos alocados no mesmo horário que utilizam as mesmas máquinas, otimizando, assim, o uso dos recursos da academia. Portanto, este trabalho visa estudar a implementação de um algoritmo genético para a otimização de grade de horários em academias.

Através da avaliação da eficácia do algoritmo para diferentes parâmetros e a adaptação do seu funcionamento com base nos dados obtidos, o trabalho apresenta valor científico como projeto de pesquisa. Por se tratar da busca de uma solução para um problema real de otimização de grade de horários com recursos limitados, a pesquisa apresentada demonstra valor prático como ferramenta auxiliar para a administração de academias de treino físico.

Neste trabalho são desenvolvidas competências como compreensão do uso de diferentes estruturas de dados para otimização de tempo de processamento, levantamento de parâmetros para avaliação do comportamento de algoritmos, conheci-

mento das diferentes técnicas utilizadas na área da computação evolucionária e formas de codificar problemas do mundo físico para a compreensão de máquinas.

Portanto, são usados algoritmos genéticos como ferramenta para a resolução do problema clássico de otimização de grade de horários por sua eficácia em gerar resultados satisfatórios para problemas complexos, de maneira a apresentar resultados com melhoria significativa da solução.

1.1 JUSTIFICATIVA

Na busca por dados de academias reais para averiguar o impacto do tempo de espera por máquinas em seu funcionamento, foi analisado o estudo de caso conduzido pela empresa Gymetrix, intitulado *The Importance of Equipment Availability to Gym Customers* (A Importância da Disponibilidade de Equipamentos para Clientes de Academias). Esse estudo analisou mais de 14 mil respostas de clientes com o objetivo de investigar como a disponibilidade de equipamentos influencia na satisfação dos usuários (Gymetrix, s.d.).

Para isso, a empresa realizou pesquisas em que os participantes indicavam seus três equipamentos favoritos e descreviam sua percepção quanto à disponibilidade dos mesmos. As opções de resposta eram: (A) “O equipamento geralmente está livre e estou satisfeito”; (B) “O equipamento geralmente está ocupado, mas não me incomoda”; e (C) “O equipamento geralmente está ocupado e isso me incomoda”.

Também foi questionada a probabilidade de o cliente recomendar a academia, em uma escala de 0 a 10. Os resultados foram analisados com base no *Net Promoter Score* (NPS), ou Índice de Promoção Líquida, uma métrica utilizada para medir a lealdade do cliente a partir da chance de recomendar um produto ou serviço (Reichheld, 2003). A Figura 1 apresenta os dados coletados.

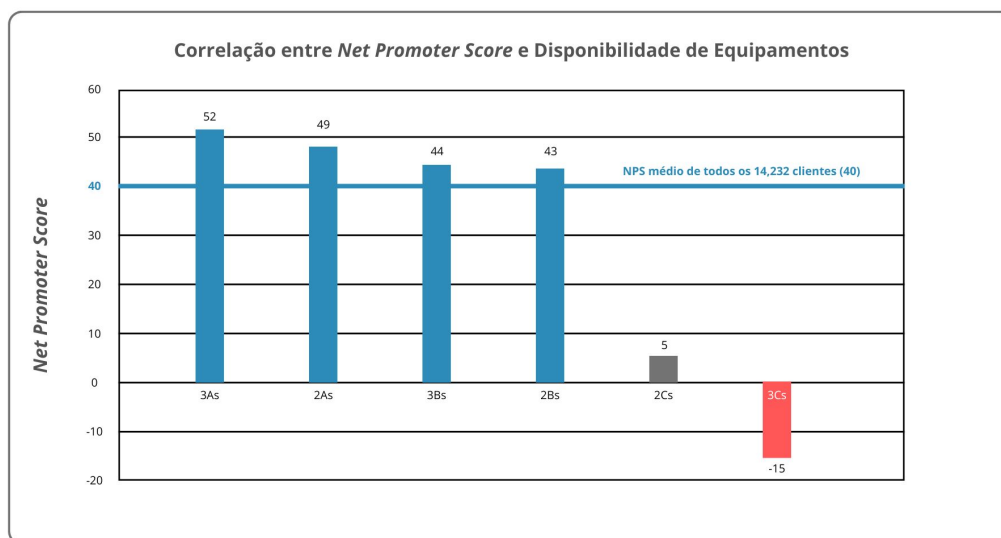


Figura 1 – Correlação entre Net Promoter Score e disponibilidade de equipamentos. Adaptado de GYMetrix (Gymetrix, s.d., p. 2).

Clientes que responderam “A” para os três equipamentos favoritos apresentaram um NPS médio de 52 — valor 12 pontos acima da média geral. Por outro lado, aqueles que marcaram “C” para dois equipamentos registraram um NPS de apenas 5, e os que responderam “C” para os três casos apresentaram um NPS de -15, ou seja, 67 pontos abaixo do grupo mais satisfeito. Outra observação foi de que muitas dessas respostas negativas ocorreram em academias que tinham capacidade excedente, ou seja, o problema não poderia ser atribuído à falta de espaço.

A análise evidenciou que a indisponibilidade dos equipamentos utilizados é uma das principais causas de insatisfação em academias, mesmo quando as mesmas apresentam capacidade subutilizada em termos gerais. O estudo conclui que garantir o acesso adequado aos equipamentos utilizados pelos clientes é essencial para promover a satisfação do usuário e maximizar o retorno sobre o investimento. Apesar disso, a disponibilidade de equipamentos ainda é um aspecto frequentemente negligenciado pela indústria de academias. Com base neste caso, é possível inferir que a redução do tempo de espera por máquinas proposta por este trabalho é justificada pelo seu valor prático e potencial para aumentar a fidelização de clientes, otimizando tanto a experiência do usuário quanto os resultados financeiros do negócio.

1.2 OBJETIVOS

O objetivo geral deste trabalho é desenvolver um algoritmo genético para a otimização de grade de horários em academias de treino físico que produza soluções satisfatórias em tempo hábil.

1.2.1 Objetivos específicos

Vislumbrando alcançar o objetivo geral, foram alinhados os seguintes objetivos específicos:

- Realizar levantamento bibliográfico do estado da arte sobre o uso de algoritmos genéticos na resolução do problema de otimização de grade de horários;
- Propor e implementar um algoritmo genético para o cenário proposto;
- Avaliação e análise detalhada do algoritmo proposto, incluindo otimização de desempenho;

1.3 MÉTODOS DE PESQUISA

A presente pesquisa se enquadra, quanto à natureza, como uma pesquisa aplicada. Com relação aos objetivos, é classificada como exploratória e explicativa. É exploratória por investigar a aplicação do algoritmo genético num cenário específico (academia de treino físico) ainda pouco abordado na literatura. É também explicativa por buscar compreender como diferentes escolhas de operadores e parâmetros do algoritmo impactam na qualidade das soluções obtidas e no desempenho do sistema.

No que se refere à abordagem metodológica, adota-se uma abordagem quantitativa, uma vez que os dados gerados pelos experimentos são analisados numericamente por meio de métricas de desempenho como a qualidade da solução (fitness) e o tempo de execução. Trata-se de um estudo de caso com experimentação computacional. O algoritmo proposto foi desenvolvido com base em princípios da computação evolucionária e testado em um ambiente simulado que replica o funcionamento de uma academia real utilizando técnicas de programação concorrente para simular o comportamento dos alunos e o acesso às máquinas. A simulação foi executada em diferentes cenários com parâmetros calibrados para representar situações típicas de utilização de academias, de modo a validar a aplicabilidade e robustez do modelo proposto.

1.4 ESTRUTURA DO TRABALHO

No capítulo 2, são apresentados os principais conceitos que envolvem o uso de algoritmos genéticos, seu funcionamento e as etapas que acontecem a cada geração. No capítulo 3, são comparadas diferentes implementações que tratam do uso de algoritmos genéticos na criação de grades de horários. Com isto, são exploradas suas características e destacadas as principais diferenças em relação à proposta deste trabalho. No capítulo 4, é apresentado o modelo do algoritmo desenvolvido, os detalhes de sua arquitetura e ciclo de execução. No capítulo 5, foi realizada uma avaliação

sistemática do algoritmo proposto através de quatro cenários com o objetivo de calibração e a validação do algoritmo. Por fim, no capítulo 6, são discutidos os resultados obtidos e indicam-se perspectivas para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, é apresentada a base teórica fundamental para a compreensão da proposta. Inicialmente, são introduzidos os conceitos de computação evolucionária, destacando o paralelo entre a evolução biológica e os algoritmos evolutivos. Em seguida, abordam-se os princípios dos algoritmos genéticos, detalhando as etapas que compõem seu ciclo de vida, como inicialização, cálculo do teste *fitness*, seleção, mutação e critério de parada.

2.1 COMPUTAÇÃO EVOLUCIONÁRIA

A computação evolucionária é uma área de pesquisa dentro da ciência da computação, inspirada na teoria da evolução (Darwin, 1859). Ao observar a maneira como espécies são testadas pelo ambiente de forma que apenas as mais aptas sobrevivem e reproduzem, é feito um paralelo com um método de resolução de problemas: a tentativa e erro. Imaginando os indivíduos de uma espécie como uma população de soluções, o ambiente avalia a aptidão das soluções para selecionar quais irão sobreviver e reproduzir. Seguindo este processo, a população irá se desenvolver de forma a ter bom desempenho para o ambiente, ou cenário, em que está inserida. Através desta analogia, foi criado o ramo da computação evolucionária (Eiben; Smith, 2015). Entre as vertentes desta área, estão os algoritmos genéticos.

2.2 ALGORITMOS GENÉTICOS

Os Algoritmos Genéticos são uma técnica de computação evolucionária que simula a evolução de uma população de soluções para um problema específico. Nesse processo, o desempenho das soluções é avaliado para gerar uma nova população com base nas características das soluções selecionadas da população anterior, repetindo o ciclo até que o critério de parada seja atingido. O ciclo de execução do algoritmo genético pode ser visualizado na figura 2. Essa técnica é altamente eficiente na busca por soluções dentro de um espaço de possíveis respostas, frequentemente encontrando resultados próximos à solução ótima (Linden, 2012).

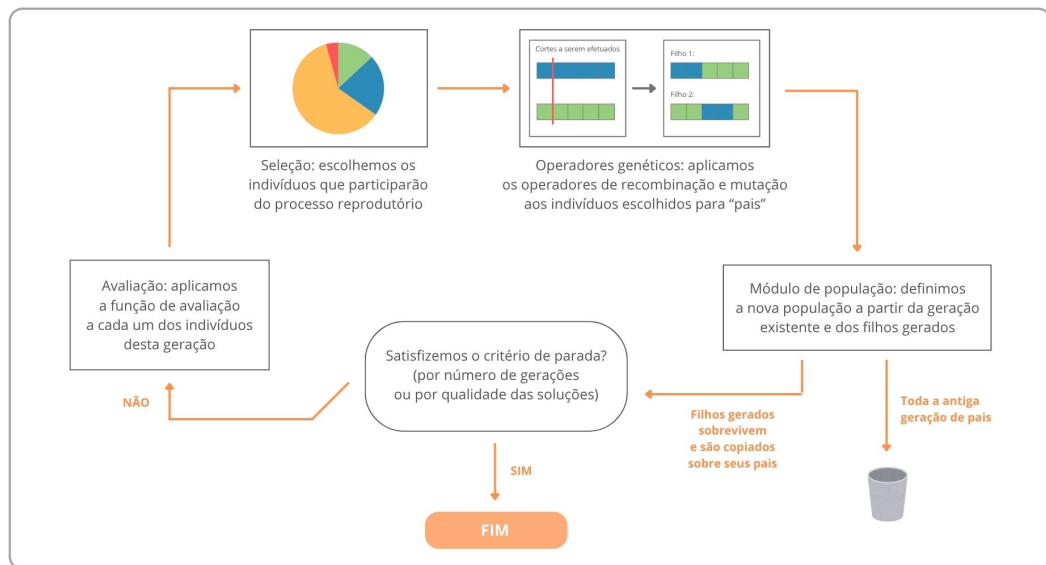


Figura 2 – Ciclo de execução de um algoritmo genético. Adaptado de (Linden, 2012, p. 27).

2.2.1 Inicialização

Na maioria das implementações, a inicialização da população é feita de forma simples: os indivíduos são instanciados de forma aleatória com base no espaço solução. Desta maneira, os indivíduos em teoria irão estar devidamente espalhados pelo espaço solução. Porém, isto não é garantido. Então, algumas implementações dividem o espaço solução em pedaços menores de forma que cada indivíduo seja inicializado em um espaço diferente. Desta forma, é possível ter um alcance amplo do universo de soluções possíveis. (Linden, 2012)

2.2.2 Codificação de Genes

Em um algoritmo genético, a codificação de genes, ou representação cromossômica, é a forma como o problema é traduzido para uma linguagem que pode ser processada pelo computador. Cada unidade dessa representação é chamada de gene. A combinação de todos os genes forma o cromossomo da solução, e essa alegoria pode ser feita de diversas maneiras. O conjunto de todos os cromossomos constitui a população de soluções, que será iterativamente melhorada pelo algoritmo genético. A codificação de genes pode ser realizada de diferentes formas, mas deve-se buscar manter a representação simples, sendo boa prática não permitir representações para estados inválidos da solução (Linden, 2012).

2.2.3 População

A população representa um conjunto de soluções para o problema proposto e será aprimorada iterativamente pelo algoritmo, com cada iteração sendo chamada de geração. A implementação mais simples do módulo de população envolve a substituição total da geração anterior pela nova. No entanto, existem diversas estratégias alternativas que podem gerar resultados melhores, como o elitismo. Outro aspecto importante a ser analisado é o tamanho da população. Se a quantidade de indivíduos for muito pequena, isso pode prejudicar a variabilidade genética, dificultando a obtenção de boas soluções. Por outro lado, se for excessivamente grande, o tempo de execução pode se tornar muito longo (Linden, 2012).

2.2.3.1 Elitismo

O elitismo é uma modificação simples no módulo da população que, apesar de pouco impactar o tempo de execução, assegura que o desempenho do algoritmo genético melhore a cada nova geração. A ideia central do elitismo é que os n melhores indivíduos de uma geração não devem ser descartados, mas sim preservados e transmitidos para a próxima, garantindo a continuidade de seus genomas. Em muitos esquemas de avaliação de um algoritmo genético, a performance é medida apenas pela qualidade da melhor solução. Assim, ao manter o melhor indivíduo da geração k na população da geração $k+1$, garantimos que o melhor indivíduo da nova geração seja, no mínimo, tão bom quanto o da geração anterior (caso nenhum indivíduo superior seja gerado). O custo adicional dessa abordagem é mínimo, uma vez que é necessário avaliar o *fitness* de cada indivíduo para realizar a seleção, sendo suficiente apenas registrar o índice do melhor indivíduo para copiá-lo para a próxima geração (Linden, 2012).

2.2.4 Teste *Fitness*

O teste *fitness*, ou função de avaliação, é o mecanismo pelo qual o algoritmo avalia a qualidade das soluções geradas. Muitas vezes, essa função é a única ligação direta entre o algoritmo genético e o problema específico. Ela atribui uma pontuação a cada indivíduo da população com base na sua eficácia como solução. Essa pontuação é então utilizada pelo método de seleção para determinar quais indivíduos irão gerar a próxima geração (Linden, 2012).

2.2.5 Métodos de Seleção

O método de seleção determina como os pais que darão origem a próxima geração são escolhidos. A princípio, pode parecer que a escolha mais óbvia seria

selecionar apenas os indivíduos mais aptos. No entanto, indivíduos com baixa aptidão podem possuir características úteis para a construção da melhor solução, e essas características podem não estar presentes em outros indivíduos da população. Outro aspecto importante é que, ao selecionar apenas os melhores indivíduos, a população pode se tornar cada vez mais homogênea, o que diminui a variabilidade genética. Esse fenômeno, denominado convergência genética, deve ser evitado. Dentre os métodos de seleção encontrados na literatura, destacam-se: a seleção por torneio, a seleção por roleta e a seleção por *ranking* (Linden, 2012).

2.2.5.1 Seleção por torneio

A seleção por torneio é um dos métodos mais utilizados em algoritmos genéticos devido à sua simplicidade e eficiência. Nesse método, um pequeno grupo de indivíduos é selecionado aleatoriamente da população — com ou sem reposição — e participa de um torneio. O indivíduo com a melhor nota de *fitness* dentro do grupo é selecionado como vencedor e incluído na nova geração. Esse processo é repetido até que o número desejado de indivíduos seja atingido (Jong, 1997).

O tamanho deste grupo pode ser ajustado, de maneira a influenciar diretamente a pressão seletiva do algoritmo. Grupos maiores aumentam a probabilidade de seleção dos indivíduos mais aptos, enquanto grupos menores favorecem a diversidade genética. Uma das principais vantagens da seleção por torneio está na sua eficiência computacional e facilidade de paralelização. Diferentemente de outros métodos que exigem cálculos globais, como a média da aptidão da população (seleção proporcional) ou a ordenação completa da população (seleção por *ranking*), a seleção por torneio permite que as decisões sejam tomadas localmente em pequenos grupos. Isso a torna particularmente adequada para implementações paralelas e distribuídas de algoritmos evolutivos (Jong, 1997).

2.2.5.2 Seleção por roleta

A seleção por roleta é um método em que a probabilidade de um indivíduo ser escolhido para reprodução está diretamente relacionada à sua nota de *fitness* em comparação com os demais membros da população. Essa técnica se assemelha a uma roleta em que cada indivíduo ocupa uma fatia proporcional ao seu desempenho, sendo então realizada uma espécie de sorteio. Assim, indivíduos com notas de *fitness* mais altas têm maior probabilidade de serem escolhidos. Esse processo é repetido várias vezes até formar a nova geração de indivíduos (Eiben; Smith, 2015).

Apesar disso, esse método apresenta algumas limitações. Quando há um ou poucos indivíduos com notas de *fitness* muito superiores ao restante da população, eles acabam sendo selecionados repetidamente, o que pode levar à convergência prematura — ou seja, o algoritmo se concentra em uma região limitada do espaço

de soluções e pode não encontrar a solução ideal. Por outro lado, quando todos os indivíduos possuem notas de fitness semelhantes, o processo de seleção se torna quase aleatório, reduzindo a pressão seletiva e tornando mais lenta a evolução da população. Mesmo com essas desvantagens, a seleção por roleta continua sendo uma técnica importante e serve de base para o desenvolvimento e entendimento de métodos mais sofisticados (Eiben; Smith, 2015).

2.2.5.3 Seleção por *ranking*

A seleção por ranking surgiu como uma alternativa à seleção por roleta, buscando corrigir algumas de suas limitações. Em vez de utilizar diretamente os valores de nota de fitness, esse método ordena os indivíduos com base em seu desempenho e atribui probabilidades de seleção conforme sua posição no ranking. Dessa forma, o processo mantém uma pressão seletiva constante, independentemente das diferenças absolutas de fitness (Eiben; Smith, 2015).

Essa abordagem parte da ideia de que a convergência prematura ocorre, muitas vezes, devido à presença de indivíduos com desempenho muito superior à média da população. Esses “super indivíduos” tendem a gerar muitos descendentes, o que reduz a diversidade genética e pode impedir que outras soluções promissoras sejam exploradas. A seleção por ranking busca mitigar esse problema, limitando a influência excessiva desses indivíduos na formação das próximas gerações (Michalewicz, 1996).

2.2.6 **Crossover**

O *crossover* é o operador genético primário: ele recebe dois indivíduos, denominados pais, e produz um ou dois novos indivíduos, denominados filhos, ao combinar partes dos pais. Neste contexto, duas implementações bem estabelecidas são o *crossover* um ponto, que divide o cromossomo em um ponto aleatório para realizar a recombinação, e o *crossover* uniforme, que decide gene a gene, com igual probabilidade, de qual dos pais o material genético será herdado. (Affenzeller et al., 2009)

2.2.6.1 *Crossover* um ponto

No *crossover* um ponto, após a seleção dos pais pelo método de seleção, define-se aleatoriamente um ponto de corte dentro do cromossomo. Esse ponto de corte representa uma posição entre dois genes consecutivos. Assim, em um cromossomo com n genes, existem $n - 1$ possíveis pontos de corte. Na figura 3 são visualizados os 4 pontos de corte de um cromossomo com 5 genes. A partir do ponto sorteado, cada pai é dividido em duas partes: uma à esquerda e outra à direita do ponto de corte (Linden, 2012).

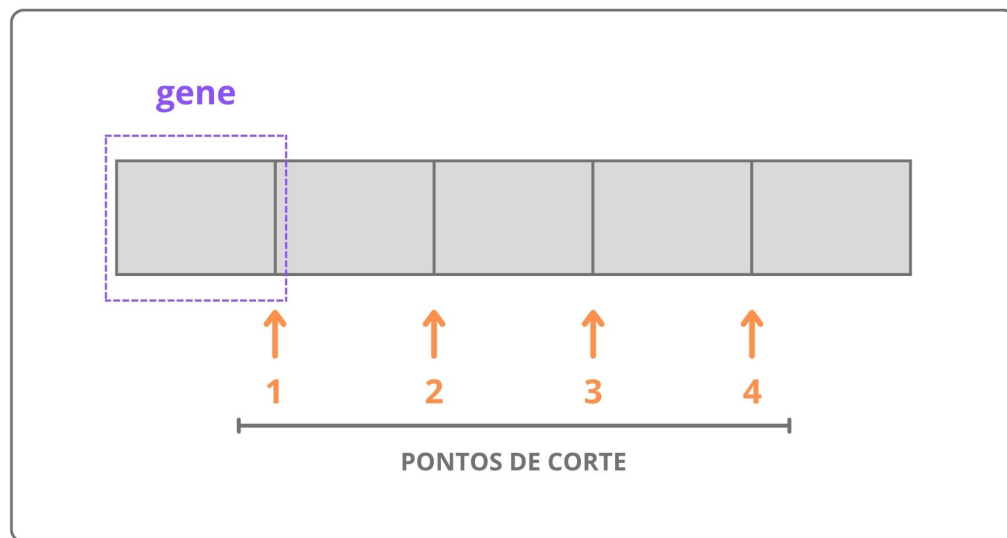


Figura 3 – Pontos de corte de um cromossomo. Adaptado de (Linden, 2012, p. 40).

A geração dos filhos ocorre por meio da combinação dessas partes. O primeiro filho é formado pela concatenação da parte esquerda do primeiro pai com a parte direita do segundo pai. O segundo filho, por sua vez, é composto pela parte esquerda do segundo pai unida à parte direita do primeiro. Esse mecanismo permite a troca de segmentos genéticos entre os indivíduos da população, contribuindo para a diversidade e a exploração do espaço de soluções ao longo das gerações do algoritmo genético (Linden, 2012).

2.2.6.2 *Crossover* uniforme

O *crossover* uniforme funciona da seguinte forma: para cada gene do cromossomo, sorteia-se aleatoriamente um número (0 ou 1). Se o número sorteado for 1, o primeiro filho herda o gene do primeiro pai e o segundo filho herda o gene do segundo pai. Se for 0, o primeiro filho recebe o gene do segundo pai e o segundo filho recebe o gene do primeiro pai (Linden, 2012). Esse tipo de cruzamento tende a garantir que cerca de 50% dos genes de cada pai sejam transmitidos aos filhos. No entanto, ele também reduz a chance de preservar grupos de genes que funcionam bem em conjunto, quando estes estão concentrados em apenas um dos pais (Eiben; Smith, 2015).

2.2.7 Mutação

O operador de mutação atua nos filhos gerados, alterando aleatoriamente seus genes. Para cada gene de cada filho, há uma chance extremamente baixa de ser modificado para um valor aleatório válido. Esse operador é fundamental, pois pode introduzir características favoráveis à solução ótima em uma população onde

essas características não estão presentes. Além disso, a mutação ajuda a evitar a convergência genética, promovendo maior diversidade na população (Linden, 2012).

2.2.8 Critério de Parada

O critério de parada define o momento de término da execução do algoritmo genético, com o melhor indivíduo da última iteração da população sendo considerado o resultado. O critério mais comum é o número de gerações completadas desde o início da execução. No entanto, outros critérios também podem ser adotados, como a obtenção de uma avaliação suficientemente alta no teste *fitness* por um indivíduo ou quando, ao longo de várias gerações consecutivas, não se observa uma melhoria significativa na qualidade dos indivíduos (Linden, 2012).

2.3 PROGRAMAÇÃO PARALELA

A programação paralela pode ser definida como um modelo que busca criar programas compatíveis com ambientes preparados para executar instruções de código simultaneamente (Palach, 2014). Nesse contexto, dois conceitos fundamentais se destacam: paralelismo e concorrência. O paralelismo ocorre quando múltiplas computações são realizadas ao mesmo tempo, enquanto a concorrência se refere à possibilidade de múltiplas computações interagirem entre si (Wittman et al., 2013).

Um exemplo de paralelismo sem concorrência seria a execução de duas *threads* em núcleos distintos de um sistema com múltiplos núcleos: ambas realizam tarefas simultaneamente, mas não interagem diretamente. No entanto, se essas *threads* tentarem acessar um recurso compartilhado — como um arquivo — ao mesmo tempo, podem surgir problemas de concorrência. Isso acontece porque, na tentativa de explorar ao máximo o paralelismo, é comum enfrentar desafios relacionados ao controle das interações entre *threads* (Wittman et al., 2013).

Threads são sequências de execução que podem ser executadas de forma independente e acessar a memória compartilhada. Em processadores com múltiplos núcleos, várias *threads* podem ser executadas simultaneamente. Como essas *threads* compartilham memória, elas podem interagir de maneira complexa e, muitas vezes, imprevisível. Ao escrever um programa concorrente, é necessário garantir que ele funcione corretamente, mesmo com diferentes ordens de execução possíveis entre as *threads*. Isso porque dois trechos de código executados concorrentemente podem modificar a mesma variável compartilhada e gerar resultados incorretos — o que é conhecido como condição de corrida (*race condition*). Para evitar isso, certas seções do código, chamadas de seções críticas, não devem ser acessadas por mais de uma *thread* ao mesmo tempo. Para proteger essas regiões, são utilizados mecanismos de

exclusão mútua, entre os quais se destacam os *mutexes* e os semáforos (Wittman et al., 2013).

2.3.1 *Mutexes*

Um *mutex* é uma variável utilizada para proteger seções críticas do código, garantindo que apenas uma *thread* as execute por vez, como se fossem uma única instrução indivisível. Ele tem dois estados: livre e ocupado. Quando uma *thread* tenta acessar uma seção crítica, ela tenta adquirir o *mutex*; se estiver livre, ela o obtém e prossegue, tornando-se a detentora do *mutex*. Se já estiver ocupado, a *thread* fica bloqueada até que o *mutex* seja liberado. Ao finalizar sua execução, a *thread* libera o *mutex*, que volta a ficar disponível. Caso existam outras *threads* aguardando, uma delas é notificada, adquire o *mutex* e acessa a seção crítica com exclusividade (Arpaci-dusseau; Arpaci-dusseau, 2018).

2.3.2 Semáforos

Um semáforo é um objeto utilizado no controle de concorrência que possui um valor inteiro interno, manipulado por meio de duas operações principais: *acquire()* e *release()*. Antes de qualquer uso, o semáforo deve ser inicializado com um valor que determina seu comportamento inicial. A função *acquire()* é responsável por decrementar esse valor e, caso ele seja maior ou igual a 1, a *thread* continua sua execução normalmente; se for 0 ou menor, a *thread* é bloqueada e entra em uma fila de espera até que outra *thread* execute a operação *release()*. Esta última, por sua vez, incrementa o valor do semáforo e, caso existam *threads* bloqueadas aguardando acesso, uma delas é despertada para prosseguir com a execução. O valor interno do semáforo, quando negativo, representa a quantidade de *threads* atualmente em espera. Dessa forma, os semáforos são empregados para controlar o acesso a seções críticas, especialmente em implementações que permitem o acesso simultâneo de múltiplas *threads* (Arpaci-dusseau; Arpaci-dusseau, 2018).

2.4 CONSIDERAÇÕES FINAIS

Neste capítulo, foram apresentados os principais fundamentos teóricos que embasam o desenvolvimento da proposta deste trabalho. A computação evolucionária foi introduzida como uma área inspirada nos mecanismos de evolução biológica, destacando o potencial dos algoritmos evolutivos na resolução de problemas complexos por meio de processos iterativos de seleção e adaptação.

Dentre as abordagens dessa área, os algoritmos genéticos se destacam pela sua simplicidade de implementação e pela eficácia em encontrar soluções satisfatórias em espaços de busca extensos. Foram detalhados os principais componentes que

compõem o ciclo de vida desses algoritmos — desde a inicialização da população até os critérios de parada — ressaltando a importância de uma representação adequada das soluções, bem como de mecanismos que equilibrem a exploração do espaço de soluções, como a seleção, o *crossover* e a mutação.

Adicionalmente, foi discutido o papel da programação paralela no contexto da computação moderna, com ênfase nos conceitos de paralelismo e concorrência. Essa contextualização foi realizada para facilitar a compreensão da simulação implementada nos experimentos do capítulo 5. Também foram apresentadas ferramentas utilizadas para a proteção de seções críticas do código, como *mutexes* e *semáforos*.

Com isso, este embasamento teórico fornece os alicerces necessários para a compreensão e implementação das metodologias apresentadas nos capítulos seguintes, permitindo uma análise crítica dos resultados obtidos e das decisões de projeto adotadas ao longo do desenvolvimento deste trabalho.

3 ESTADO DA ARTE

Neste capítulo, são apresentados três artigos relevantes para o trabalho proposto. O primeiro aborda a aplicação de um algoritmo genético na criação de grades de horários para aulas em universidades. O segundo apresenta um algoritmo genético aprimorado para o problema de agendamento *job shop*, introduzindo uma abordagem de busca local cíclica em diferentes etapas para atingir a convergência de forma mais rápida. Por fim, o terceiro artigo realiza um estudo comparativo de diferentes algoritmos genéticos aplicados ao problema de agendamento de aulas em universidades.

3.1 A GENETIC ALGORITHM FOR THE REAL-WORLD UNIVERSITY COURSE TIMETABLING PROBLEM

O estudo conduzido por (Wong et al., 2022) propõe um algoritmo genético para resolver o problema de criação de grades de horários para aulas em universidades. A construção manual dessa grade é um processo complexo, demorado e suscetível a erros, especialmente devido às múltiplas restrições envolvidas. Este problema é classificado como NP-difícil, o que implica que não pode ser resolvido de forma ótima por métodos exatos em tempo polinomial.

O objetivo do problema é alocar um conjunto de aulas (frequentadas por diferentes grupos de alunos) a um conjunto de salas e horários, respeitando restrições rígidas, como: apenas uma aula pode ser lecionada por sala em um dado horário, a sala deve comportar o número de alunos matriculados e um professor não pode estar atribuído a mais de uma aula simultaneamente. Além disso, o problema envolve restrições flexíveis, como minimizar aulas únicas em um dia ou evitar aulas consecutivas para os mesmos alunos. O estudo utiliza dados reais de um semestre da University Malaysia Sabah-Labuan International Campus (UMS-LIC) para validar a eficácia do algoritmo em cenários reais.

O artigo então demonstra como foram implementadas as etapas do algoritmo genético. A inicialização foi feita por busca tabu com amostragem e perturbação, como demonstrado em um dos trabalhos referenciados (Goh et al., 2017). A função *fitness* foi projetada para avaliar soluções com base no número de violações das restrições flexíveis, atribuindo pontuações mais altas às soluções que apresentam menos violações. O critério de parada é atingido quando nenhuma violação é detectada ou após transcorrido um tempo fixo. Para a seleção, foram testados os métodos de *steady-state* e roleta, avaliando qual produzia melhores resultados.

A operação de *crossover* foi omitida da implementação devido à possibilidade de gerar soluções inválidas, o que exigiria uma função de reparo complexa. Como

alternativa, utilizou-se a função de mutação com um número fixo de ocorrências para evoluir os cromossomos pais em seus filhos. Foram implementadas duas modalidades de mutação: transferência, onde um evento é movido para um bloco vazio, e troca, na qual dois eventos têm suas posições permutadas. Para realizar a mutação, dois pontos são selecionados na solução, onde genes presentes podem ser transferidos ou trocados a depender de um dos pontos estar vazio, desde que a solução resultante seja válida. Desta forma, duas novas soluções são criadas a partir das soluções pais. Estas novas soluções então substituem as soluções da população com a menor nota de *fitness*.

Os primeiros experimentos realizados tiveram como objetivo determinar os parâmetros ideais para otimizar o desempenho do algoritmo genético. Esses experimentos foram conduzidos variando um parâmetro por vez, enquanto os demais eram mantidos constantes. Com base nos resultados, foram definidos os valores ideais para o tamanho da população e o número de mutações por geração. Posteriormente, foram realizadas duas séries de experimentos para comparar os métodos de seleção *steady-state* e roleta, com o método *steady-state* apresentando melhor desempenho.

A performance dos experimentos foi comparada com a da grade de horários gerada manualmente pelos funcionários do setor administrativo da UMS-LIC. Enquanto a grade gerada automaticamente cumpriu todas as restrições rígidas, a manual apresentou alunos matriculados em aulas conflitantes e aulas alocadas em salas incompatíveis. Em relação às restrições flexíveis, a grade automática obteve uma nota média de *fitness* de 2463,94 entre suas execuções, enquanto a grade manual obteve uma nota de 5304. Esses resultados demonstram o êxito da implementação proposta, pois indicam que o método automatizado oferece uma solução superior à abordagem manual.

3.2 RESEARCH AND APPLICATIONS OF SHOP SCHEDULING BASED ON GENETIC ALGORITHMS

No estudo conduzido por (Zhao; Kong, 2016), é apresentada a implementação de um algoritmo genético aplicado ao problema de *job shop scheduling*, que consiste na alocação de maquinários e tarefas em uma grade de horários, considerando tempos de execução variados para cada máquina. O objetivo do estudo é aprimorar a estrutura básica de um algoritmo genético por meio da implementação de busca cíclica em diferentes etapas da execução, e avaliar sua eficácia.

A busca cíclica é implementada nas etapas de *crossover* e mutação. O método de *crossover* implementado foi uma variação do *crossover* de dois pontos. Para a mutação, é feita a inversão de sub-sequências de genes representando a ordem de execução. Nessas etapas, os cromossomos gerados têm suas notas de *fitness* comparadas às de seus pais. Caso a nota de *fitness* do filho seja inferior à do pai,

o algoritmo repete a operação até obter um indivíduo mais apto ou atingir um limite máximo de tentativas pré-definido. Esse processo compõe o conceito de busca cíclica mencionado anteriormente. A ideia central é utilizar esse método para melhorar a convergência, acelerar o processamento do algoritmo e evitar o surgimento de um mínimo local do espaço de busca. Para o *fitness*, foi utilizado o tempo de processamento total para que todas as máquinas finalizem suas tarefas. Neste artigo, o critério de parada do algoritmo não foi especificado.

Com o objetivo de verificar a eficiência da implementação, foram realizados testes utilizando dois tipos de *benchmarks* usados na literatura: FT06 (Muth; Thompson, 1963), composto por 6 máquinas e 6 tarefas, sendo considerado de resolução mais simples, e LA01 (Lawrence, 1984), composto por 10 tarefas e 5 máquinas, que apresenta maior complexidade de resolução. Para ambos os testes, o algoritmo apresentou boa velocidade e convergência.

Como teste adicional, foi realizado um comparativo entre o algoritmo proposto e um algoritmo genético generalizado. Os resultados indicaram que o algoritmo proposto teve um menor tempo de execução e apresentou uma curva de convergência mais estável. Dessa forma, o algoritmo alcançou seus objetivos principais: evitar a convergência em mínimos locais e melhorar tanto o tempo de processamento quanto a estabilidade na convergência.

3.3 COMPARATIVE STUDY OF GENETIC ALGORITHMS FOR THE SCHOOL TIMETABLING PROBLEM

O artigo de (Dias et al., 2024) apresenta um algoritmo genético como solução para o problema de criação de grades de horários para aulas em universidades. O estudo também inclui uma análise comparativa de diferentes implementações desta solução, explorando os resultados obtidos para diversos parâmetros e destacando como as variações influenciam fatores como o tempo de execução e a qualidade das soluções geradas.

O cenário do problema é então detalhado: matérias e professores devem ser alocados a horários de modo que não ocorram conflitos. Exemplos de conflitos incluem a atribuição de um mesmo professor a duas matérias no mesmo horário ou duas disciplinas de um mesmo semestre curricular estarem programadas para ocorrerem simultaneamente. O problema é classificado como NP-difícil, portanto, métodos exatos não conseguem resolvê-lo de forma ótima em tempo polinomial.

O estudo explica o funcionamento das variações implementadas do algoritmo genético usual: o algoritmo genético de chaves aleatórias (RKGA, do inglês *random-key genetic algorithm*) e sua versão viciada (BRKGA, do inglês *biased random-key genetic algorithm*). Ambas as variações utilizam uma representação das soluções como vetores de chaves, que são decodificados para o cálculo do *fitness* e, posteri-

ormente, recodificados. Além disso, após serem inicializadas de forma aleatória, as populações são divididas em dois grupos: elites e não elites, com todos os indivíduos elite sendo diretamente transferidos para a próxima geração. Por fim, essas versões também incluem a inserção de soluções mutantes geradas aleatoriamente em cada nova população, com a finalidade de aumentar a diversidade genética.

O RKGA e o BRKGA diferem na maneira como os indivíduos são selecionados para a operação de *crossover*. No RKGA, dois indivíduos do grupo de não elites são escolhidos para gerar novos descendentes. Já no BRKGA, um dos indivíduos selecionados para o cruzamento deve obrigatoriamente pertencer ao grupo de elites.

A nota de *fitness* foi calculada com base nas restrições aplicadas às soluções, classificadas em rígidas e flexíveis. Entre as restrições rígidas, destacam-se: o tempo alocado na grade de horários para uma matéria deve ser compatível com sua carga horária de créditos, e, caso uma matéria seja atribuída a um horário, é necessário que um professor também seja designado a ela. Já as restrições flexíveis incluem aspectos como horários preferenciais dos professores, a alocação de duas matérias ao mesmo professor no mesmo horário e atribuições que conflitam com agendamentos previamente definidos.

A execução de oito métodos foi avaliada: algoritmo genético com *crossover* de 2 pontos sem elitismo (GA2p), algoritmo genético com *crossover* de 2 pontos com elitismo (GA2pE), algoritmo genético com *crossover* de múltiplos pontos sem elitismo (GAMp) e com elitismo (GAMpE), RKGA com *crossover* de 2 pontos (RKGA2p), BRKGA com *crossover* de 2 pontos (BRKGA2p), RKGA com *crossover* de múltiplos pontos (RKGAMp) e BRKGA com *crossover* de múltiplos pontos (BRKGAMp). As execuções foram analisadas com diferentes valores para os parâmetros de tamanho da população (10, 15, 20, 50, 100 e 200 indivíduos) e número de gerações (10, 50, 100, 200 e 400 gerações). A taxa de mutação foi mantida fixa em 0,05%.

As métricas de avaliação consideradas foram: número de restrições rígidas violadas, número de restrições flexíveis violadas, número total de restrições violadas, nota de *fitness* e tempo de execução. Os experimentos evidenciaram que o aumento no tamanho da população e no número de gerações contribui para melhorar a qualidade das soluções, explorando um espaço de busca mais amplo. Esse comportamento foi especialmente notável em métodos como o BRKGAMp, que apresentou reduções significativas de violações e melhor nota de *fitness*.

Por outro lado, em alguns casos (por exemplo, GA2p e RKGA2p com 200 indivíduos), não foram observadas melhorias significativas, mesmo com o aumento das populações e gerações. Além disso, constatou-se que o uso de elitismo em GA não proporcionou benefícios substanciais. Esse padrão foi consistente em ambas as estratégias de *crossover* avaliadas. Um exemplo claro é o caso de GA2p e GA2pE com 100 gerações, cujas soluções apresentaram o mesmo valor da função objetivo e

violaram o mesmo número de restrições, mas demandaram maior tempo de execução.

O BRKGA obteve melhores resultados do que o RKGA nos dois tipos de *crossover* analisados. Observou-se que, embora o RKGA não tenha conseguido encontrar soluções viáveis, o RKGAm, com 50 gerações e 100 indivíduos, gerou soluções com poucas restrições rígidas e flexíveis violadas. Esse resultado destaca a importância de não descartar imediatamente soluções inviáveis, pois elas ainda podem contribuir para melhorias.

O *crossover* de múltiplos pontos se mostrou mais eficiente e robusto do que o *crossover* de 2 pontos durante os experimentos. Esse comportamento indica que considerar um maior número de pontos de *crossover* ao combinar cromossomos pode levar a soluções mais promissoras e diversificadas. Outra vantagem do *crossover* de múltiplos pontos é sua capacidade de gerar maior variabilidade, devido ao aumento dos pontos de cruzamento entre os pais.

Entre os métodos utilizados no estudo, o BRKGA apresentou a melhor performance, com resultados válidos e satisfatórios e custo computacional aceitável. Algumas melhorias sugeridas para o algoritmo incluem o uso de métodos de *crossover* alternativos e a aplicação diferenciada desses métodos, como a combinação de múltiplos operadores de *crossover* e o uso de *crossover* adaptativo, onde os operadores são ajustados dinamicamente durante a execução.

3.4 COMPARAÇÕES

Os trabalhos analisados, assim como a proposta apresentada, abordaram uma subcategoria do problema de criação de grades de horários utilizando algoritmos genéticos. Os cenários abordados incluem a criação de grades de horários em universidades, fábricas (*job shop*) e academias de treino físico. Cada um desses contextos impõe restrições específicas, exigindo adaptações na estrutura tradicional dos algoritmos genéticos para que possam atender adequadamente às particularidades de cada problema.

Em relação à inicialização, no trabalho proposto, ela é realizada de forma aleatória, garantindo o respeito às restrições do problema. Em (Dias et al., 2024) e (Zhao; Kong, 2016), os detalhes da implementação não foram especificados, sendo informado apenas que a inicialização também foi feita aleatoriamente. Por outro lado, (Wong et al., 2022) empregou uma abordagem mais sofisticada, utilizando busca tabu com amostragem e perturbação.

Quanto aos métodos de *crossover* adotados, (Dias et al., 2024) utilizou *crossover* de dois pontos, *crossover* de múltiplos pontos e *crossover* de dois pontos combinado com busca cíclica, conforme também adotado por (Zhao; Kong, 2016). Já (Wong et al., 2022) não utilizou operador de *crossover*, restringindo-se ao operador

de mutação. Na presente proposta, foi empregado o *crossover* de um ponto e uma implementação própria de *crossover* de múltiplos pontos.

No trabalho proposto e em (Wong et al., 2022), o operador de mutação é implementado como a troca entre dois elementos de um bloco da grade de horários, sendo que (Wong et al., 2022) adiciona ainda a possibilidade de transferir um elemento para um bloco vazio. Uma peculiaridade desse trabalho é o uso de um número fixo e relativamente elevado de mutações por geração, sendo este o único operador empregado para gerar uma nova população, já que o *crossover* não foi implementado. Em (Zhao; Kong, 2016), a mutação consiste em destacar uma sequência de inteiros do cromossomo, representando uma ordem de eventos, e inverter a ordem dessa sequência. Por sua vez, (Dias et al., 2024) utiliza a abordagem usual de mutação, que consiste na permutação aleatória de um gene do indivíduo, devido à estrutura de suas soluções em forma de vetores.

As pesquisas de (Wong et al., 2022) e (Dias et al., 2024), assim como a proposta apresentada, adotaram um modelo semelhante para a função de *fitness*, baseado no número de violações das restrições impostas, com pesos distintos atribuídos a diferentes restrições. No entanto, (Dias et al., 2024) permitiu a presença de soluções inválidas na população, buscando maior diversidade genética. Nesse caso, o caráter de invalidez das soluções resultava em uma penalização significativa na avaliação de *fitness* do indivíduo. Para (Zhao; Kong, 2016), a função *fitness* teve como parâmetro o tempo de processamento total para que todas as máquinas finalizem suas tarefas. Na proposta apresentada, a restrição considerada envolve a ocorrência simultânea da mesma máquina no treino de diferentes alunos em um mesmo horário, além do número de alunos que não puderam ser alocados. A penalidade aplicada cresce de forma exponencial com esses valores, conferindo maior severidade às soluções que infringem tais critérios.

Os critérios de parada adotados foram baseados em tempo de execução (predeterminado) e também no controle do número de gerações. No artigo de (Zhao; Kong, 2016), o critério de parada utilizado não foi especificado. Assim como em (Dias et al., 2024), na presente proposta adotou-se o critério de número fixo de gerações, pois permite analisar o comportamento do algoritmo em um intervalo de tempo customizável, sendo o parâmetro ideal para a realização de experimentos no trabalho proposto.

Na tabela 1 são demonstradas as características dos algoritmos estudados junto com o trabalho proposto para comparação. De modo geral, observa-se que, apesar das variações entre os trabalhos analisados, há uma convergência em torno de certos princípios, como a representação das soluções, o uso de penalidades para violações de restrições e a importância da diversidade na população. A proposta aqui discutida se insere nesse contexto, contribuindo com uma abordagem adaptada ao

domínio específico do problema.

Tabela 1 – Comparação entre trabalhos relacionados

Categoria	(Wong et al., 2022)	(Zhao; Kong, 2016)	(Dias et al., 2024)	Trabalho Proposto
Contexto	Criação de grades de horários em universidade	Problema de agendamento em <i>job shop</i> (máquinas e tarefas com tempos variados)	Alocação de disciplinas e professores em instituições de ensino superior	Agendamento de alunos em horários com base na disputa por máquinas em uma academia
Escopo	Aulas, professores e salas considerando restrições rígidas e flexíveis	Alocação eficiente de tarefas em máquinas com busca por minimizar o tempo total de trabalho	Alocação de horários sem conflitos, considerando preferências e restrições	Minimizar o tempo de espera dos alunos, considerando treinos e disponibilidade de máquinas
Inicialização	Busca tabu com amostragem e perturbação para criar soluções iniciais válidas	Inicialização aleatória (sem detalhes técnicos)	Inicialização aleatória (com cromossomos do tipo vetor de chaves)	Inicialização aleatória respeitando restrições de alocação e disponibilidade dos alunos
<i>Crossover</i>	Não utilizado (evita gerar soluções inválidas, usa apenas mutação)	<i>Crossover</i> de dois pontos com busca cíclica para melhorar aptidão local	<i>Crossover</i> de dois e múltiplos pontos com variações BRKGA e RKGA	<i>Crossover</i> de um ponto e múltiplos pontos
Mutação	Troca de eventos entre blocos ou transferência para blocos vazios; principal operador do algoritmo	Inversão de sub-sequência de genes representando a ordem de execução	Permutação aleatória de genes; inserção de mutantes aleatórios na população	Troca entre horários de alunos compatíveis
<i>Fitness</i>	Avalia apenas violações de restrições flexíveis (pontuação decrescente por violação)	Tempo de processamento total para que todas as máquinas finalizem suas tarefas	Avalia restrições rígidas e flexíveis com forte penalidade para soluções inválidas	Penaliza ocorrências simultâneas da mesma máquina em um horário e alunos não alocados (penalidade exponencial)
Critério de Parada	Tempo fixo de execução ou quando não há mais violações nas soluções	Não especificado	Número de gerações fixo (10 a 400), analisando impacto em desempenho e diversidade	Número fixo de gerações (150)

3.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou uma análise crítica do estado da arte em algoritmos genéticos aplicados ao problema de agendamento. Os resultados desta revisão sistemática fornecem uma base sólida para o desenvolvimento da solução proposta, que será detalhada no capítulo seguinte.

4 PROPOSTA

Neste capítulo, propõe-se o uso de um algoritmo genético para otimização de horários em academias, detalhando sua arquitetura e funcionalidade. O algoritmo proposto contribui para a otimização dos recursos da academia ao realizar uma triagem dos alunos que solicitaram reserva de um horário para treino, agrupando-os com alunos de treinos distintos, reduzindo assim o tempo de espera por equipamentos.

Para isso, os dados dos alunos — seus treinos e horários disponíveis — são fornecidos ao sistema previamente à execução do algoritmo. Cada aluno é representado por uma tupla contendo duas sub-tuplas de inteiros. A primeira indica os horários em que o aluno está disponível para treinar, por exemplo: [14, 16, 17]. Todos os alunos devem possuir disponibilidade em ao menos um horário, mas não há limite máximo, podendo declarar disponibilidade total dentre os horários ofertados. A segunda sub-tupla representa o treino do aluno, isto é, as máquinas que ele utilizará durante o exercício. Essa sub-tupla contém 10 inteiros entre 0 e 9, cada um correspondendo a um dos 10 tipos de máquinas disponíveis na academia. Um exemplo de treino seria [3, 1, 3, 4, 5, 2, 3, 9, 9, 0]. O exercício específico que o aluno fará na máquina não é relevante para a implementação, pois seu foco está na ocupação da máquina em que o exercício é feito. Portanto, os exercícios feitos são abstraídos nesta implementação, denotando somente a máquina que é utilizada no treino. Para manter a coerência com treinos reais, foi definido um limite de até três ocorrências de uma mesma máquina por treino. Essa restrição foi implementada porque, na prática, é incomum que um treino real utilize predominantemente um único tipo de equipamento.

Com a lista de alunos — denominada neste trabalho como *lista origem* — o algoritmo genético inicializa a primeira geração de soluções através da alocação aleatória dos alunos dentre os horários em que os mesmos têm disponibilidade. Em seguida, as soluções obtidas são avaliadas pela função *fitness* do algoritmo. A função implementada avalia a qualidade de cada solução com base na contagem total de ocorrências de cada tipo de máquina nos treinos dos alunos alocados em um mesmo horário. Desta forma, a ordem em que as máquinas estão declaradas nos treinos não tem influência na execução. Esta implementação é justificada pela possibilidade de um aluno utilizar outra máquina do seu treino quando a que ele iria utilizar está ocupada. Assim, o algoritmo tem como objetivo reduzir o tempo em que um aluno fica ocioso pois todas as máquinas de seu treino estão ocupadas, com a comparação da ordem dos treinos dos alunos em um horário estando fora de seu escopo. Ao fim da execução do algoritmo, o resultado é a alocação dos alunos nos horários em que terão menor competição por máquinas do seu treino.

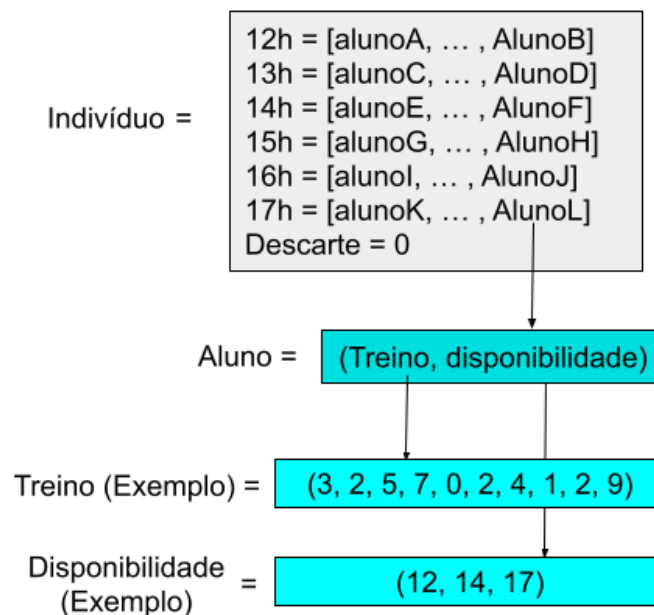
Além disso, o modelo contempla situações em que a demanda excede a ca-

pacidade da academia. Nesses casos, o algoritmo pode descartar alunos que não puderem ser alocados a nenhum horário disponível. Cada solução carrega uma variável chamada *descarte*, um inteiro inicialmente zero que é incrementado para cada aluno não alocado. Desta forma, os alunos descartados não são mantidos na estrutura final da solução, apenas a contagem dos mesmos. Essa modelagem visa representar com fidelidade um cenário real de alocação com recursos limitados e garantir que as soluções obtidas sejam viáveis e aplicáveis. A implementação detalhada de cada etapa do algoritmo — incluindo a codificação dos indivíduos, operadores genéticos utilizados, função *fitness* e critério de parada — será apresentada nas seções seguintes deste capítulo.

4.1 MODELAGEM DO CROMOSSOMO

O cromossomo representa uma possível solução para o problema proposto, ou seja, representa a alocação dos alunos em horários para realizarem seu treino. Ele é composto por 7 genes: 6 representando horários e 1 referente ao valor de *descarte*. Cada horário é uma lista de alunos com um limite máximo de 20. O valor de *descarte* é um número inteiro positivo que indica a quantidade de alunos não alocados nesse cromossomo. Os horários estão distribuídos nas posições 0 a 5, correspondendo, respectivamente, às 12h, 13h, 14h, 15h, 16h e 17h. Assim, o cromossomo representa uma possível distribuição dos alunos nos horários disponíveis. A figura 4 ilustra a modelagem do cromossomo neste algoritmo genético.

Modelagem do Cromossomo

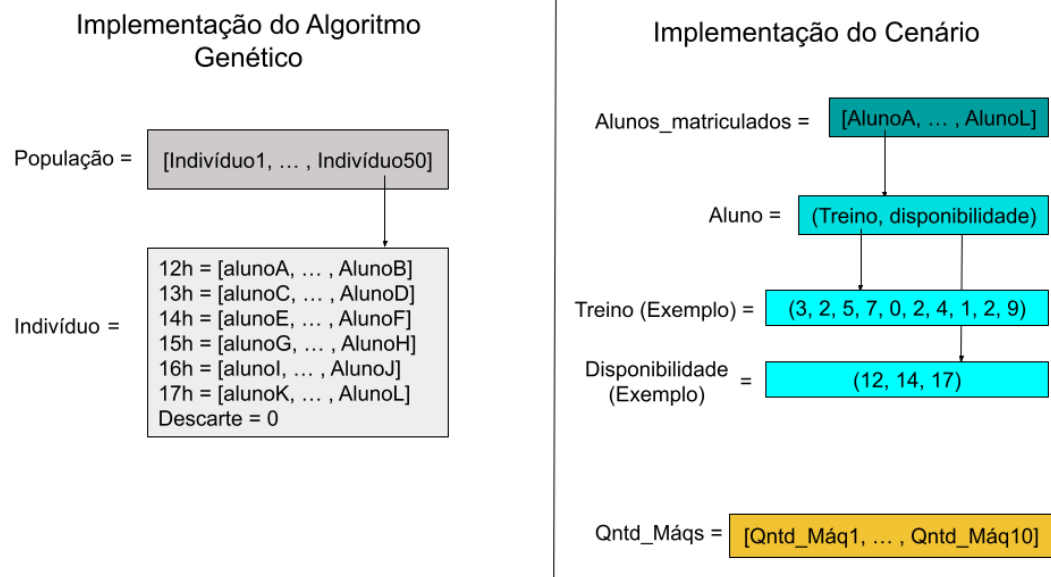


Fonte: autoria própria

Figura 4 – Modelagem do Cromossomo

4.2 ARQUITETURA DO ALGORITMO GENÉTICO

A arquitetura do algoritmo genético é apresentada na Figura 5. O algoritmo genético opera com uma população de 50 indivíduos, cada um representando uma solução possível para o problema em questão. Cada indivíduo é composto por seis listas, que representam horários e os alunos alocados neles, e um número inteiro positivo que indica o total de alunos descartados durante a distribuição. Os alunos são representados por uma tupla contendo duas sub-tuplas de inteiros: a primeira descreve o treino (onde cada valor representado por um número inteiro corresponde a uma máquina/equipamento), e a segunda sub-tupla indica os horários em que o aluno está disponível para ir à academia. Estes horários podem ser mapeados para as listas do indivíduo ao subtrair 12 de cada valor, relacionando-os diretamente às posições das listas de horários. Além disso, há uma lista suplementar para representar o número de instâncias de cada máquina, onde cada posição corresponde a uma máquina específica. Por exemplo, a quantidade de máquinas 1 está armazenada na posição 1 desta lista.



Fonte: autoria própria

Figura 5 – Arquitetura do software

4.3 CICLO DE EXECUÇÃO DO ALGORITMO

O ciclo de execução do algoritmo segue o modelo tradicional de um algoritmo genético. Primeiro, a população é inicializada aleatoriamente. Em seguida, os indivíduos são ordenados em ordem crescente com base em suas notas de *fitness*, posicionando o melhor indivíduo no topo. Uma nova estrutura é então criada para armazenar a próxima geração, na qual os melhores indivíduos da geração atual, definidos pela taxa de elitismo, são copiados. Os demais indivíduos são gerados por meio de ciclos de seleção de pais, *crossover* e mutação, até que a população atinja seu tamanho final. Com a nova geração instanciada, a anterior é descartada e o processo recomeça. Esse ciclo é repetido até que o critério de parada seja alcançado, e o melhor indivíduo da última geração é considerado a solução encontrada.

4.3.1 Inicialização

A inicialização da população é feita utilizando a lista de matriculados ("lista origem") para distribuir os alunos em horários que atendam os critérios. Para cada indivíduo, os alunos são removidos de uma cópia da lista origem e alocados a horários com espaço disponível e compatíveis com sua disponibilidade. Essa distribuição ocorre de forma aleatória até que todos os alunos sejam processados. Caso, durante a inicialização, um aluno só tenha disponibilidade em horários lotados, ele será descartado, e o valor de descarte da solução será incrementado em 1. Para esta implementação, o tamanho da população é estático em 50 indivíduos.

4.3.2 Seleção

O método de seleção adotado neste trabalho foi a seleção por *ranking*. Nesse procedimento, os indivíduos da população são ordenados de forma crescente com base em suas respectivas notas de *fitness*. A partir dessa ordenação, atribui-se a cada indivíduo um valor de *ranking* proporcional à sua posição.

A conversão do *ranking* em probabilidade de seleção é realizada da seguinte forma: subtrai-se o *ranking* do indivíduo r_i do tamanho total da população N acrescido de uma unidade. O resultado dessa operação é então dividido pela soma total dos rankings de todos os indivíduos da população. Dessa forma, obtém-se a probabilidade de seleção associada a cada indivíduo. Assim, a fórmula utilizada é descrita na equação 1:

$$P(i) = \frac{N+1-r_i}{\sum_{j=1}^N (N+1-r_j)} \quad (1)$$

onde:

- $P(i)$ é a probabilidade de seleção do indivíduo i ,
- N é o tamanho da população,
- r_i é o ranking do indivíduo i (sendo 1 o melhor).

Essa estratégia promove uma distribuição mais equilibrada das probabilidades de seleção, reduzindo o impacto de valores de *fitness* extremamente elevados e contribuindo para a manutenção da diversidade genética ao longo das gerações.

4.3.3 Crossover entrelaçado

Devido à implementação do cenário extender além do teste *fitness* pela inco-mum representação cromossomial, a operação de *crossover* foi adaptada para contemplar o uso da lista origem de alunos matriculados e sua distribuição em horários. Uma possível implementação foi denominada *crossover* entrelaçado (Figuras 6 e 7), pois intercala entre os dois indivíduos pais para determinar a posição do próximo aluno nos indivíduos novos. Primeiramente, a função faz uma cópia da lista origem e cria estruturas para armazenar os dois filhos e suas respectivas listas de repescagem. Em seguida, ela percorre a lista origem, processando cada elemento (representando um aluno). Para cada aluno, identifica-se seu horário em um dos pais, e ele é alocado ao mesmo horário em um dos filhos. O mesmo processo é repetido com o outro pai e filho, completando a alocação. Em cada nova iteração, alterna-se o pai utilizado, promovendo um entrelaçamento das características dos pais nos filhos.

Caso o horário em que o aluno deveria ser alocado esteja cheio, ele é adicionado à lista de repescagem correspondente (uma para cada filho). Após concluir a iteração pela lista origem, tenta-se alocar os alunos da repescagem em outros horários em que estes tenham disponibilidade. Se todos os horários estiverem ocupados, o aluno é descartado, e a variável de descarte do respectivo filho é incrementada em 1. Essa variável será posteriormente utilizada no cálculo da nota de *fitness*.

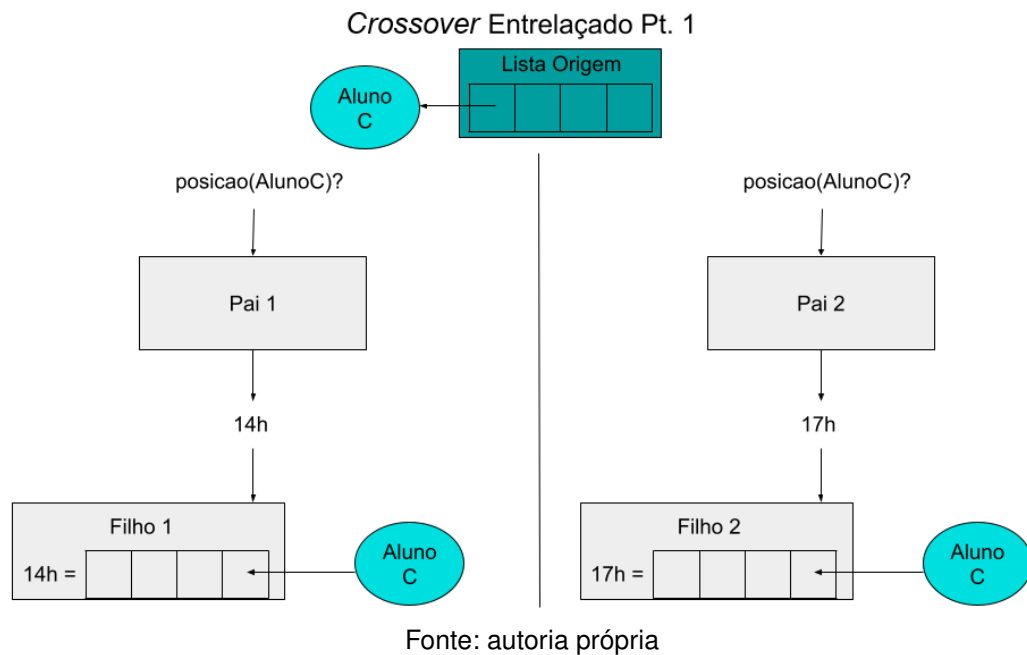


Figura 6 – Crossover entrelaçado parte 1

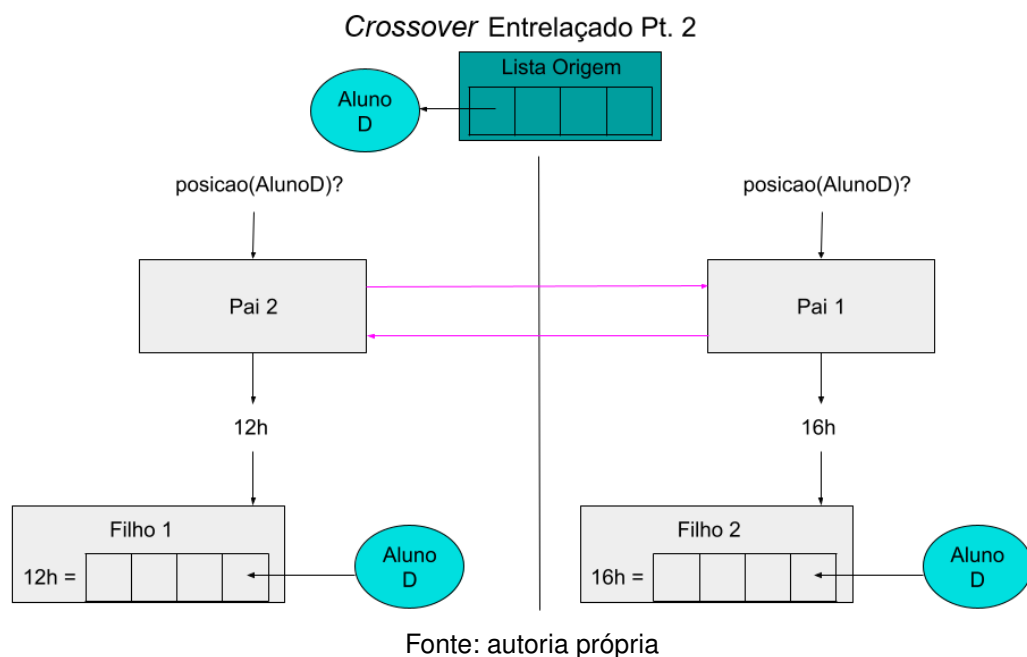


Figura 7 – Crossover entrelaçado parte 2

4.3.4 Crossover um ponto

Como alternativa, foi implementado o *crossover* de um ponto, adaptado ao contexto do problema. Esse método divide o cromossomo dos pais em duas partes, intercalando suas metades para gerar dois novos filhos. O ponto de corte é definido no meio da lista de origem. Na primeira metade, os alunos são alocados nos filhos com base em suas posições no pai correspondente. Para a segunda metade, o processo é repetido, invertendo os papéis dos pais. Além disso, aplica-se a mesma lógica de repescagem do *crossover* entrelaçado, com incremento na variável de descarte do filho quando não há espaço para um aluno.

4.3.5 Mutação

Devido à estrutura do cromossomo, a mutação usual, que atribui um valor aleatório do espaço solução a um gene, não é aplicável, pois implicaria remover um aluno para inserir outro já processado. Por isso, a mutação foi adaptada ao contexto. Quando ocorre, dois alunos trocáveis — isto é, com disponibilidade nos horários um do outro — são selecionados e têm seus horários permutados. Essa abordagem atinge o objetivo de promover diversidade genética na população. A taxa de mutação adotada foi de 20%. Optou-se por este valor devido ao grande tamanho dos genes presentes em cada indivíduo. Como a probabilidade de mutação é aplicada uma única vez por indivíduo — e não individualmente para cada aluno em um gene/horário — uma taxa elevada ajuda a garantir uma frequência adequada de alterações ao longo das gerações.

4.3.6 Teste *Fitness*

A função *fitness* implementada é calculada com base na distribuição de uso das máquinas em cada horário. Quanto mais uniforme essa distribuição, melhor o valor de *fitness*. Além disso, o número de alunos descartados também influencia negativamente o resultado, aplicando uma penalidade exponencial. Para esta implementação, valores menores de *fitness* indicam soluções melhores.

Para cada horário, realiza-se o seguinte processo: contabiliza-se o número de vezes que cada uma das 10 máquinas é utilizada. Para cada máquina n , eleva-se 2 ao número de usos x_n , e os resultados são somados, conforme mostrado na Equação 2:

$$h = \sum_{n=1}^{10} 2^{x_n} \quad (2)$$

Este processo é repetido para os seis horários do indivíduo, somando-se os respectivos valores h_i . Em seguida, aplica-se uma penalidade exponencial ao número de alunos descartados na solução, dada por 8^d , onde d representa o número total de

alunos descartados. O fator 8 foi escolhido por ser equivalente a 2^{3d} , o que faz com que o peso dos descartes cresça exponencialmente mais rápido que o peso associado a conflitos de máquinas, já que corresponde ao cubo de 2^x . Essa escolha visa evitar que alunos com possibilidade de alocação em horários com vagas disponíveis sejam descartados apenas para reduzir artificialmente o valor de *fitness*. A penalidade associada ao uso das máquinas é então somada à penalidade pelos descartes, resultando no valor final de *fitness*, descrito na Equação 3.

$$F = \sum_{i=1}^6 h_i + 8^d \quad (3)$$

4.3.7 Critério de Parada

O critério de parada implementado baseia-se no número de gerações, encerrando a execução após a 150ª geração. Este critério foi selecionado pois permite analisar o comportamento do algoritmo em um intervalo de tempo maior, sendo o parâmetro ideal para a realização da pesquisa proposta.

4.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou a proposta de um algoritmo genético adaptado para otimização da alocação de alunos em horários de treino na academia, buscando reduzir a competição por máquinas e o tempo de espera. Foram detalhadas a modelagem do cromossomo, a arquitetura do algoritmo, os operadores genéticos aplicados, a função de *fitness* e o critério de parada, evidenciando a adaptação das técnicas clássicas para o contexto específico do problema.

Na sequência, o capítulo Análise de experimentos trará a avaliação prática do desempenho do algoritmo, por meio de testes e simulações, permitindo validar sua efetividade e apontar oportunidades de aprimoramento.

5 ANÁLISE DE EXPERIMENTOS

Neste capítulo, são apresentados os experimentos realizados para avaliar o desempenho do algoritmo genético proposto. Para isso, foi desenvolvida uma simulação que reproduz o funcionamento de uma academia, com o objetivo de medir, principalmente, o tempo de espera dos alunos durante o uso dos equipamentos. Foram testados diferentes cenários, variando métodos de seleção, operadores de *crossover*, número de gerações e comparando os resultados com uma heurística de busca local, o algoritmo guloso (Cormen et al., 2022). A análise leva em conta tanto a nota de *fitness* quanto os resultados obtidos na simulação e no tempo de execução, buscando entender os efeitos práticos de cada configuração sobre a qualidade das soluções geradas.

5.1 SOBRE A SIMULAÇÃO

Para avaliar a eficácia das soluções geradas pelo algoritmo, foi desenvolvida uma simulação que representa o funcionamento prático de uma academia. O objetivo da simulação é observar o comportamento realista do agendamento proposto, mensurando principalmente o tempo total de espera dos alunos — ou seja, o tempo que cada um passa ocioso aguardando a liberação de máquinas.

A academia simulada possui 10 tipos diferentes de equipamentos. Cada tipo é representado por um semáforo, cujo contador interno define a quantidade disponível daquele equipamento. Esses semáforos são armazenados em uma lista, sendo identificados pelo índice correspondente ao tipo da máquina. Por exemplo, se há três instâncias da máquina 0, o semáforo presente no índice 0 da lista é inicializado com o valor 3. Essa abordagem permite simular a competição entre os alunos de forma direta, através da disputa por recursos compartilhados.

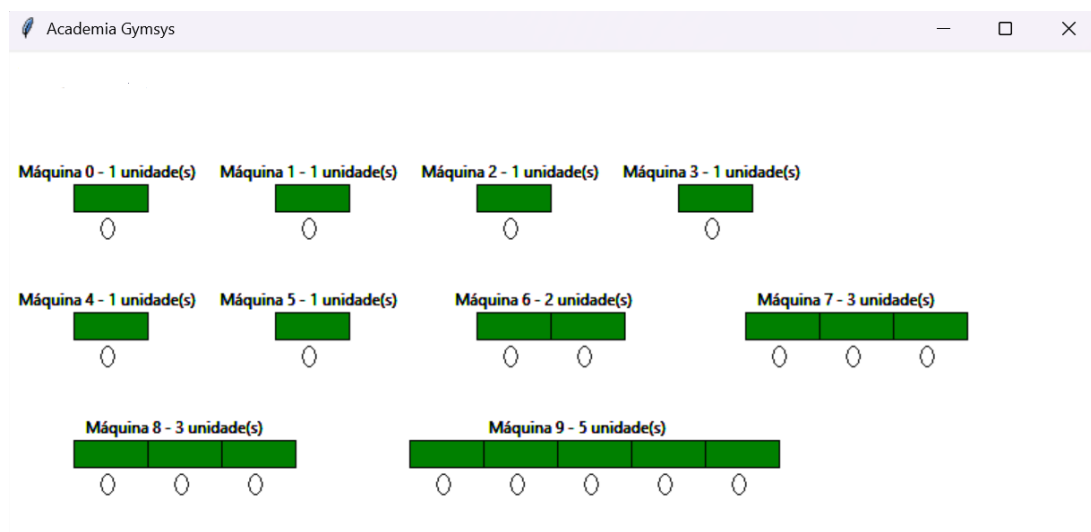
Os alunos são representados por *threads*, e cada *thread* executa a função *treinar*, responsável por simular o comportamento do aluno durante seu horário na academia. O treino de cada aluno é uma lista com 10 números entre 0 e 9, representando a sequência de máquinas que o aluno precisa utilizar.

Durante a execução da simulação, a *thread* de cada aluno tenta adquirir uma das máquinas listadas em seu treino. Se ao menos uma delas estiver disponível no momento, a *thread* adquire o semáforo correspondente e utiliza a máquina pelo tempo fixo definido para aquele equipamento, simulando o tempo de uso. Caso nenhuma das máquinas esteja disponível, a *thread* aguarda 0.1 segundos, adiciona esse valor ao tempo de espera do aluno, e tenta novamente. Esse processo se repete até que uma das máquinas desejadas possa ser utilizada. Ao final do treino, a *thread* adiciona o

tempo total de espera individual a uma variável global que acumula o tempo de espera total da simulação.

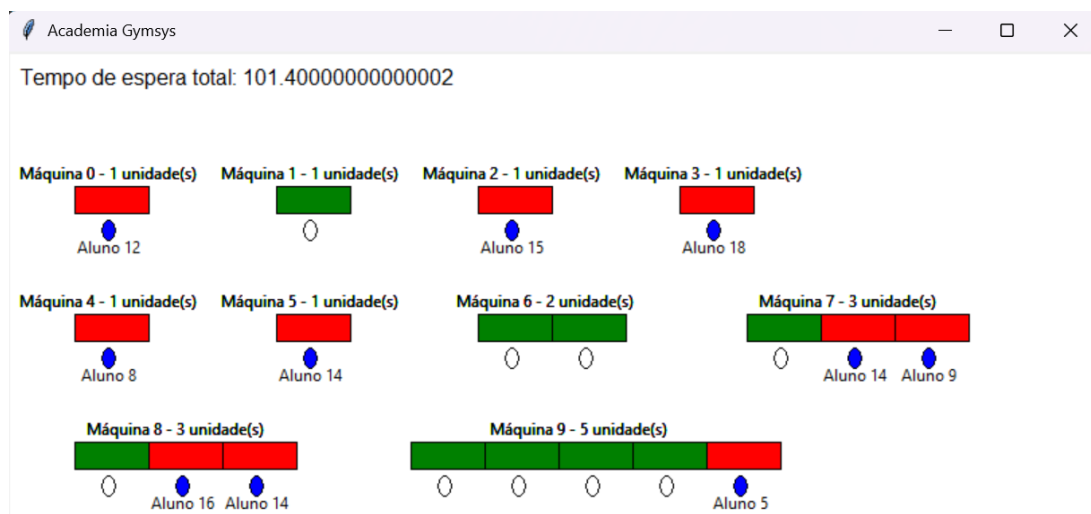
A simulação é executada separadamente para cada um dos seis horários da academia, com os alunos alocados no mesmo horário competindo entre si pelas máquinas, de forma a simular os conflitos reais causados pela demanda simultânea pelos equipamentos entre os alunos presentes.

Além da lógica da simulação, foi implementada uma interface gráfica para permitir o acompanhamento visual da academia durante a execução. A Figura 8 mostra o estado inicial da academia antes do início da simulação, enquanto a Figura 9 apresenta a academia em funcionamento, com os alunos utilizando os equipamentos.



Fonte: autoria própria

Figura 8 – Interface gráfica da simulação com academia vazia



Fonte: autoria própria

Figura 9 – Interface gráfica da simulação com academia em uso

Os parâmetros da simulação foram definidos com o intuito de refletir uma academia realista. A quantidade de instâncias para cada tipo de máquina foi: 1, 1, 1, 1, 1, 1, 2, 3, 3 e 5 (para os tipos 0 a 9, respectivamente). Já os tempos de uso definidos foram 0.3, 0.3, 0.4, 0.4, 0.5, 0.5, 0.7, 1.0, 1.2 e 1.5 segundos. A escolha desses valores se baseou na observação de que equipamentos com tempo de uso mais longo, como esteiras e bicicletas, tendem a estar disponíveis em maior número, enquanto aparelhos usados rapidamente, como máquinas de treino para grupos musculares isolados, costumam ser mais escassos.

Por meio dessa simulação, torna-se possível avaliar de maneira prática e visual a eficiência das soluções propostas pelo algoritmo, especialmente no que diz respeito ao impacto do agendamento no tempo de espera dos usuários.

5.2 ESPECIFICAÇÕES DO SISTEMA

Todos os testes foram realizados no software *Visual Studio Code* (Microsoft, 2024), utilizando *Python* versão 3.12.6. A Tabela 2 apresenta a configuração do PC utilizado nas simulações.

Tabela 2 – Especificações detalhadas do sistema utilizado nos testes.

Categoria	Detalhes
Processador	
Processador	13ª Geração Intel(R) Core(TM) i9-13900HX
Número de Núcleos	32
Velocidade	5.4 GHz
Memória	
RAM	32 GB
Placa de Vídeo 1	
Placa de Vídeo	NVIDIA GeForce RTX 4080 Laptop GPU
Memória Total	28 GB
Memória Dedicada	12 GB
Placa de Vídeo 2	
Placa de Vídeo	Intel(R) UHD Graphics
Memória Total	16 GB
Memória Dedicada	128 MB
Sistema Operacional	
Tamanho	64 bits
Sistema Operacional	Windows 11

5.3 RESULTADOS

Para calibrar os parâmetros do algoritmo e validar sua eficácia, foram definidos quatro cenários distintos para a realização dos testes. As métricas utilizadas na avaliação do desempenho do algoritmo em cada cenário foram: a nota de *fitness* do melhor indivíduo da geração final, o tempo total de espera dos alunos durante a simulação implementada e o tempo de execução do algoritmo. A partir dessas métricas, é possível verificar se o objetivo proposto — implementar um algoritmo genético para a otimização de grades de horários em academias de treino físico que produza soluções satisfatórias em tempo hábil — foi alcançado.

Para cada cenário, foram realizadas 20 execuções independentes, cada uma utilizando uma lista distinta de alunos a serem alocados. Quando algum parâmetro era alterado para fins de comparação — como o método de seleção, tipo de *crossover* ou número de gerações — a nova configuração era também executada 20 vezes, utilizando exatamente as mesmas listas de alunos das execuções anteriores. Essa abordagem garante que as variações observadas nos resultados sejam atribuídas às alterações de parâmetro, e não às diferenças nos dados de entrada.

No cenário A, avalia-se o desempenho do algoritmo utilizando os métodos de seleção por torneio, *ranking* e roleta. O cenário B compara o uso dos métodos de *crossover* entrelaçado e de um ponto. O cenário C analisa os resultados obtidos na 100ª e na 150ª geração de uma mesma execução do algoritmo, com critério de parada fixado em 150 gerações, permitindo observar os ganhos decorrentes da execução prolongada. Já no cenário D, compara-se o desempenho da melhor configuração do algoritmo genético, definida com base nos cenários anteriores, com uma abordagem gulosa desenvolvida para o mesmo problema de agendamento.

Exceto quando o parâmetro em questão está sendo deliberadamente alterado para o experimento, os valores utilizados foram: tamanho da população igual a 50, elitismo de 2 indivíduos, taxa de mutação de 20%, método de seleção por *ranking*, método de *crossover* entrelaçado e critério de parada em 150 gerações. A Tabela 3 resume os cenários testados, seus escopos e os parâmetros adotados em cada um deles.

Tabela 3 – Cenários avaliados nos experimentos realizados

	Cenário A	Cenário B	Cenário C	Cenário D
Escopo	Métodos de seleção	Métodos de <i>crossover</i>	Número de gerações	Comparação com outros algoritmos de otimização
Parâmetros utilizados	Torneio, <i>ranking</i> e roleta	<i>Crossover</i> entrelaçado e <i>crossover</i> um ponto	Execução por 100 gerações e 150 gerações	Algoritmo proposto e algoritmo guloso

5.3.1 Cenário A - Métodos de Seleção

No cenário A, foi analisado o desempenho do algoritmo utilizando três métodos distintos de seleção. Esses métodos são responsáveis por definir quais indivíduos da população serão escolhidos como pais, cujas características serão herdadas pela próxima geração.

Os métodos avaliados foram: seleção por torneio, seleção por *ranking* e seleção por roleta. Na seleção por torneio, uma pequena amostra da população é escolhida aleatoriamente (nesta análise, a amostra foi composta por três indivíduos), e os dois com melhor valor de *fitness* dentro dessa amostra são selecionados. Entre suas principais vantagens estão a simplicidade de implementação, a rapidez na execução e a robustez, já que tende a apresentar bom desempenho em uma ampla variedade de problemas de otimização. Contudo, esse método pode ser superado por abordagens mais sofisticadas, especialmente em contextos específicos de otimização.

A seleção por roleta atribui a cada indivíduo uma probabilidade de seleção proporcional ao seu valor de *fitness* em relação ao total da população. Ou seja, quanto melhor o *fitness*, maior a chance de ser escolhido. Esse método se mostra mais eficaz quando as diferenças de *fitness* entre os indivíduos são pequenas. Caso contrário, os melhores indivíduos tendem a ser selecionados com frequência excessiva, o que pode levar à convergência precoce da população.

Já a seleção por *ranking* ordena os indivíduos de acordo com seu valor de *fitness*, atribuindo probabilidades com base nessa classificação. Esse método ignora os valores absolutos do *fitness*, utilizando-os apenas para ranqueamento, o que resulta em perda da escala real. Essa característica pode ser benéfica ou prejudicial, a depender do contexto. Em problemas com grande variação entre os *fitness*, por exemplo, essa abordagem tende a evitar o domínio absoluto de poucos indivíduos, contribuindo para uma maior diversidade na seleção.

A Tabela 4 apresenta os resultados obtidos nos experimentos realizados com cada um desses métodos de seleção.

Tabela 4 – Resultados dos experimentos para o cenário A.

	Nota <i>Fitness</i>			Tempo de espera total (segundos)			Tempo de execução (segundos)		
Execução	Torneio	<i>Ranking</i>	Roleta	Torneio	<i>Ranking</i>	Roleta	Torneio	<i>Ranking</i>	Roleta
Execução 1	162.171	112.804	212.343	160.4	144.3	173.5	9.495	39.682	55.322
Execução 2	44.261	43.540	63.472	128.6	122.4	134.4	15.443	27.641	31.955
Execução 3	43.766	38.773	62.647	92.2	89.2	120.3	16.939	44.756	32.126
Execução 4	204.579	143.753	297.185	223.7	228.4	224.7	24.070	42.024	48.820
Execução 5	274.416	245.039	319.472	202.2	206.8	205.6	40.636	49.506	43.966
Execução 6	82.815	56.296	82.019	156.9	142.7	151.4	8.766	28.105	19.879
Execução 7	38.955	35.344	50.422	160.9	146.8	161.1	11.014	16.346	19.294
Execução 8	60.484	50.092	84.937	149.1	148.9	145.2	10.744	22.162	18.399
Execução 9	35.156	24.246	52.111	137.7	121.4	161.9	10.913	21.140	20.642
Execução 10	49.258	34.428	67.459	153.1	133.4	153.4	10.115	21.311	21.202
Execução 11	127.126	116.847	176.628	152.7	140.6	172.5	10.366	19.723	21.172
Execução 12	105.462	59.200	143.167	144.5	127.9	133.3	9.806	18.125	19.123
Execução 13	73.978	45.860	102.165	154.2	132.3	157.2	9.175	19.840	19.419
Execução 14	129.847	71.312	141.312	172.6	155.8	184.7	9.929	22.228	21.453
Execução 15	296.477	266.146	317.047	204.0	202.5	226.8	13.052	24.623	24.685
Execução 16	126.822	96.807	156.666	201.2	178.1	219.3	11.007	35.290	34.512
Execução 17	101.593	92.033	131.646	197.5	214.6	232.7	18.938	36.428	35.009
Execução 18	85.218	57.278	115.655	165.7	152.9	170.0	17.311	30.940	32.888
Execução 19	116.396	85.332	136.409	173.1	164.7	179.6	29.711	39.515	35.270
Execução 20	132.166	110.776	131.555	184.3	183.4	174.6	20.598	37.262	55.668

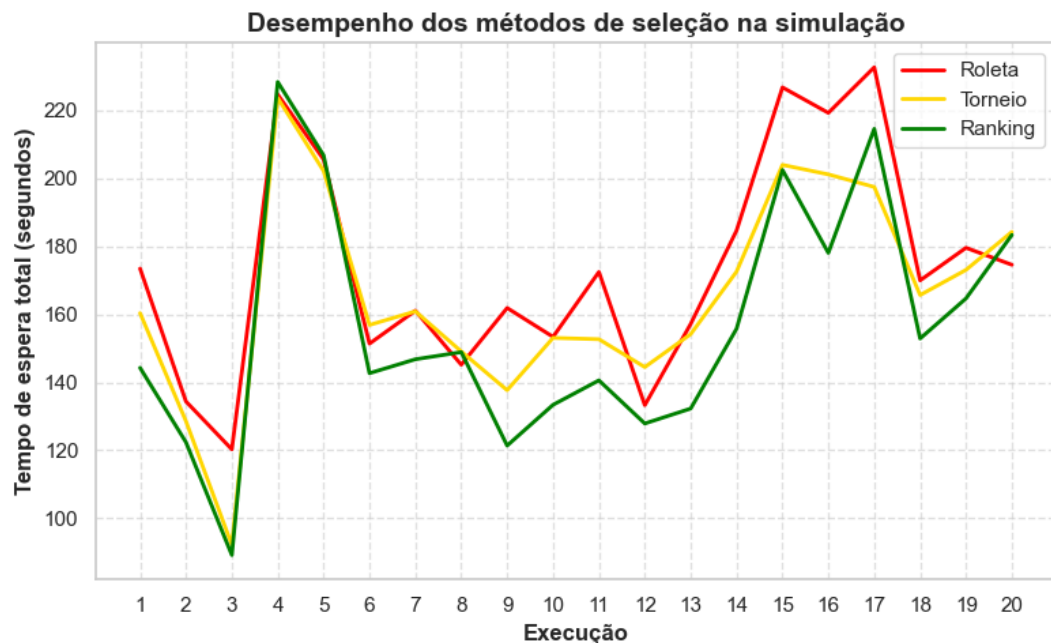
Melhor resultado
 Resultado intermediário
 Pior Resultado

A seleção por *ranking* apresentou o melhor desempenho em todas as execuções em termos de nota *fitness*. A seleção por torneio ficou em segundo lugar de forma consistente. Nas poucas execuções em que a roleta superou o torneio, a diferença foi mínima, indicando que o desempenho da roleta esteve próximo, mas ainda inferior, na maior parte dos casos.

Nos testes de simulação, o comportamento foi semelhante: a estratégia por *ranking* se destacou na maioria das execuções, e mesmo quando não apresentou o melhor resultado, a diferença em relação à melhor alternativa foi pequena. A seleção por torneio manteve um desempenho estável, sendo novamente a segunda melhor. Já a roleta teve, na maioria das vezes, o pior desempenho. Quando não foi a pior, sua performance ainda ficou próxima da mais baixa observada.

Com relação ao tempo de execução, a seleção por torneio foi a mais rápida, o que era esperado dada a sua simplicidade de implementação. As estratégias por *ranking* e roleta apresentaram tempos de execução semelhantes, já que ambas exigem o cálculo de probabilidades individuais de seleção para cada elemento da população.

O gráfico na Figura 10 representa o desempenho dos três métodos de seleção na simulação durante as 20 execuções realizadas.



Fonte: autoria própria

Figura 10 – Desempenho dos métodos de seleção na simulação do cenário A

Diante dos resultados, a seleção por *ranking* foi adotada como parâmetro no algoritmo, uma vez que obteve os melhores resultados globais tanto na nota *fitness* quanto nas simulações. Embora não tenha sido a mais rápida, seu tempo de execução ainda é compatível com os objetivos do trabalho. A estratégia por roleta, por outro lado, apresentou o pior desempenho geral, o que é consistente com sua conhecida limitação em cenários com alta disparidade entre os valores de *fitness* dos indivíduos, situação que tende a favorecer excessivamente aqueles com notas melhores.

5.3.2 Cenário B - *Crossover*

No cenário B, foi analisado o desempenho do algoritmo utilizando os dois métodos de *crossover* descritos no Capítulo 4 deste trabalho: o *crossover entrelaçado* e o *crossover um ponto*. O *crossover entrelaçado* distribui os alunos da lista de origem de forma alternada entre os indivíduos filhos, enquanto o *crossover um ponto* realiza essa distribuição de maneira contínua, alternando apenas a partir da metade da lista. Os resultados obtidos nos experimentos para este cenário estão apresentados na Tabela 5.

Tabela 5 – Resultados dos experimentos para o cenário B.

Execução	Nota <i>Fitness</i>		Tempo de espera total (segundos)		Tempo de execução (segundos)	
	Entrelaçado	Um Ponto	Entrelaçado	Um Ponto	Entrelaçado	Um Ponto
Execução 1	122.052	144.794	199.7	216.0	16.501	30.893
Execução 2	42.619	49.159	129.9	123.7	31.452	23.460
Execução 3	40.714	59.941	144.1	162.7	36.217	30.239
Execução 4	67.318	55.887	177.2	150.1	30.550	31.375
Execução 5	1221.927	1188.364	183.8	165.9	41.565	22.342
Execução 6	239.993	227.541	193.1	201.4	35.380	46.312
Execução 7	109.958	153.354	177.0	178.3	28.334	36.041
Execução 8	256.999	241.533	188.9	203.7	47.098	30.044
Execução 9	38.740	41.056	124.0	138.0	19.671	16.898
Execução 10	38.458	53.993	129.3	135.4	16.985	16.962
Execução 11	96.600	105.775	174.4	156.1	18.870	22.053
Execução 12	226.148	236.356	199.8	217.3	23.561	23.036
Execução 13	60.883	70.714	134.4	156.2	20.075	19.329
Execução 14	41.095	47.817	127.9	136.2	18.552	17.856
Execução 15	37.578	50.670	127.8	149.3	14.911	14.805
Execução 16	69.469	64.442	178.4	160.0	17.097	16.742
Execução 17	103.274	77.734	177.7	189.5	17.262	36.288
Execução 18	84.025	96.686	200.3	182.7	46.681	41.569
Execução 19	41.230	62.231	164.0	186.1	27.146	27.566
Execução 20	138.928	125.780	170.0	157.9	38.261	37.454

 Melhor resultado  Pior Resultado

Os resultados obtidos ao longo dos testes apontam uma vantagem geral para o *crossover* entrelaçado. Em termos de *fitness*, este método superou o *crossover* de um ponto em 65% das execuções, com diferenças que variaram de 5% (execução 9) até 33% (execução 19). Já nas execuções em que o *crossover* de um ponto foi superior, as melhorias variaram de 3% (execução 5) a 24% (execução 17).

Na simulação, o desempenho do *crossover* entrelaçado também foi superior em 65% das execuções. Dentre esses casos, 76% coincidiram com as melhores notas de *fitness*, o que indica uma forte correspondência entre o valor de aptidão obtido e o desempenho prático na simulação.

Quanto ao tempo de execução, não houve um claro vencedor. As diferenças foram, em sua maioria, mínimas, e os casos com vantagem significativa apareceram em quantidade semelhante para ambos os métodos.

Diante desses dados, o *crossover* entrelaçado foi escolhido como parâmetro

final. Seu desempenho mais consistente tanto na nota de *fitness* quanto na simulação levou a esta decisão. O tempo de execução, por sua vez, teve pouca influência na escolha, já que os dois métodos apresentaram tempos semelhantes ao longo dos testes.

5.3.3 Cenário C - Número de Gerações

No cenário C, foi avaliado o desempenho do algoritmo ao estender a execução em 50 gerações adicionais, totalizando 150 gerações. Para isso, comparou-se a solução obtida na 100ª geração com aquela gerada ao final da 150ª. Como consequência direta da continuidade da execução, métricas como a nota de *fitness* tendem, obrigatoriamente, a apresentar algum grau de melhora. Por isso, a análise deve considerar a proporção dessa melhora em relação ao aumento no tempo de execução. Já a avaliação das soluções na simulação não requer ressalvas adicionais, uma vez que não há garantias de que a solução obtida após 150 gerações terá desempenho superior à da 100ª geração. A seguir, na tabela 6, são expostos os resultados obtidos nos experimentos para este cenário:

Tabela 6 – Resultados dos experimentos para o cenário C.

Execução	Nota <i>Fitness</i>			Tempo de espera total (segundos)			Tempo de execução (segundos)		
	100 Ge.	150 Ge.	Mudança %	100 Ge.	150 Ge.	Mudança %	100 Ge.	150 Ge.	Mudança %
Execução 1	44.545	41.578	-6.66%	126.9	109.0	-14.11%	12.126	22.138	+82.57%
Execução 2	86.997	72.593	-16.56%	174.9	170.7	-2.40%	16.406	29.033	+76.97%
Execução 3	102.198	88.034	-13.86%	200.9	199.3	-0.80%	14.150	24.333	+71.96%
Execução 4	31.687	30.114	-4.96%	112.1	116.2	+3.66%	13.123	18.289	+39.37%
Execução 5	86.264	73.804	-14.44%	135.6	127.7	-5.83%	10.834	16.259	+50.07%
Execução 6	96.291	77.128	-19.90%	158.9	151.7	-4.53%	10.835	16.324	+50.66%
Execução 7	34.761	32.097	-7.66%	111.2	122.3	+9.98%	13.115	17.946	+36.84%
Execução 8	250.741	244.974	-2.30%	184.6	177.1	-4.06%	15.536	21.689	+39.61%
Execução 9	152.404	134.398	-11.81%	168.6	189.0	+12.10%	35.373	54.621	+54.42%
Execução 10	82.756	56.496	-31.73%	156.0	156.2	+0.13%	26.339	39.003	+48.08%
Execução 11	134.967	105.505	-21.83%	187.4	168.6	-10.03%	17.207	33.992	+97.55%
Execução 12	142.033	133.546	-5.98%	159.8	168.8	+5.63%	22.393	39.730	+77.42%
Execução 13	48.071	45.450	-5.45%	152.2	137.4	-9.72%	15.748	28.157	+78.79%
Execução 14	114.950	103.072	-10.33%	203.6	179.0	-12.08%	22.303	39.444	+76.86%
Execução 15	109.582	74.959	-31.60%	156.0	134.9	-13.53%	19.765	34.430	+74.20%
Execução 16	1213.927	1188.315	-2.11%	200.0	207.7	+3.85%	32.379	49.600	+53.18%
Execução 17	1196.950	1180.107	-1.41%	201.1	219.4	+9.10%	30.654	47.770	+55.84%
Execução 18	1279.463	1256.358	-1.81%	259.7	253.8	-2.27%	25.258	43.251	+71.23%
Execução 19	117.240	79.062	-32.56%	159.7	137.4	-13.96%	15.624	26.642	+70.52%
Execução 20	166.691	143.360	-14.00%	259.4	239.1	-7.83%	28.498	42.733	+49.95%

■ Melhor resultado
 ■ Pior Resultado
 ■ Mudança favorável
 ■ Mudança desfavorável

Quanto ao desempenho *fitness*, os resultados apresentaram melhoras variando entre -1% e -32%, evidenciando que o ganho não segue uma relação linear com o aumento de 50% no número de gerações. Esse comportamento é consistente com o princípio de convergência em algoritmos evolutivos, onde a população tende a se estabilizar em soluções ótimas locais após um determinado número de gerações.

A simulação revelou um cenário misto, com casos de melhora (-1% a -14%) e piora (1% a 12%) no desempenho. No entanto, houve um claro predomínio de melhorias: em cinco execuções, os ganhos superaram 10%, enquanto apenas uma execução apresentou piora equivalente. Essa distribuição aponta para a eficácia da modificação implementada.

O tempo de execução aumentou entre 40% e 80%, superando a previsão teórica de 50% associada à mudança de 100 para 150 gerações. Apesar disso, o tempo final permaneceu em níveis aceitáveis, já que a duração original já era curta.

Embora os resultados tenham ficado abaixo das expectativas iniciais, as melhorias no desempenho foram significativas. O aumento no tempo de processamento, ainda que relevante, não inviabilizou a operação do algoritmo. Diante disso, a decisão de manter 150 gerações mostrou-se adequada, pois equilibrou ganhos de performance com um tempo de execução ainda dentro do esperado – especialmente considerando que a maioria dos casos testados apresentou resultados positivos.

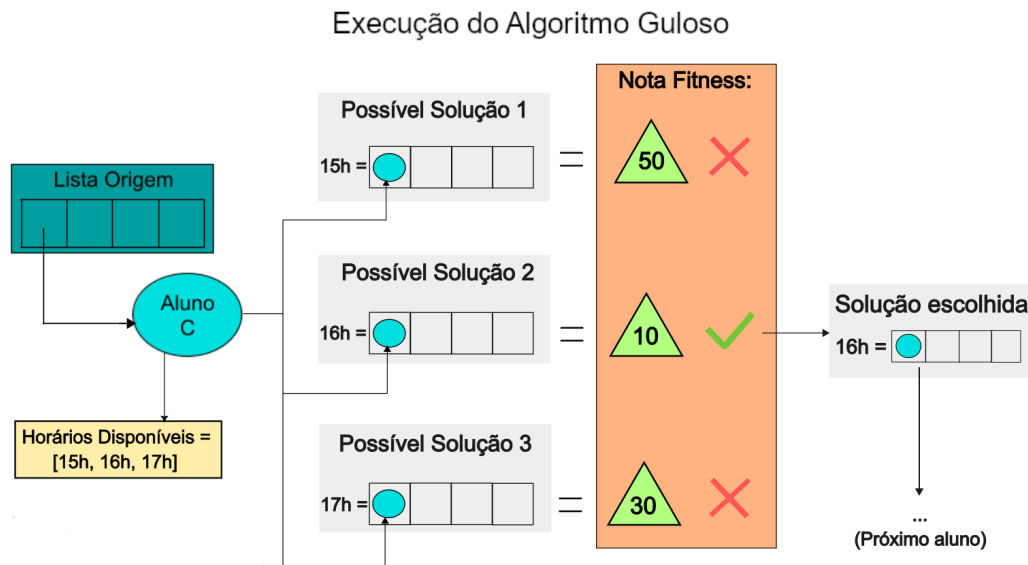
5.3.4 Cenário D - Algoritmo Genético Vs. Algoritmo Guloso

No Cenário D, realizou-se uma avaliação comparativa entre o algoritmo genético desenvolvido e uma abordagem gulosa para alocação de alunos em horários de academia. Um algoritmo guloso faz, em cada etapa, a escolha que parece mais vantajosa no momento, buscando assim uma solução globalmente ótima a partir de decisões localmente ótimas. Embora nem sempre leve à melhor solução possível, costuma produzir resultados satisfatórios e, em muitos casos, encontra a solução ótima (Cormen et al., 2022).

Para o experimento, utilizou-se com o algoritmo genético os parâmetros que obtiveram melhor desempenho nos testes preliminares, garantindo assim sua configuração ideal na comparação com a abordagem gulosa desenvolvida para o problema proposto.

O algoritmo guloso implementado segue uma estratégia direta e determinística. Inicialmente, o método recebe como entrada a lista completa de alunos a serem alocados. Para cada aluno, o algoritmo avalia todas as possíveis alocações nos horários disponíveis, selecionando aquela que proporciona o menor valor imediato para a função *fitness* - a mesma função utilizada pelo algoritmo genético. Este processo de avaliação e alocação é repetido sequencialmente para todos os alunos até que a so-

lução completa seja construída. O ciclo de execução do algoritmo guloso é destacado na figura 11.



Fonte: autoria própria

Figura 11 – Execução do algoritmo guloso

A comparação entre essas duas abordagens é motivada pela necessidade de entender se a complexidade extra do algoritmo genético realmente traz benefícios em relação a uma alternativa mais direta. O algoritmo guloso tem a vantagem de ser mais simples e rápido, porém espera-se que o algoritmo genético consiga encontrar soluções melhores, já que explora o espaço de busca de forma mais ampla e pode evitar cair em ótimos locais que limitariam a qualidade da solução final.

A seguir, na tabela 7, são expostos os resultados obtidos nos experimentos para o cenário D:

Tabela 7 – Resultados dos experimentos para o cenário D.

Execução	Nota <i>Fitness</i>		Tempo de espera total (segundos)		Tempo de execução (segundos)	
	Genético	Guloso	Genético	Guloso	Genético	Guloso
Execução 1	43.860	64.205	153.3	168.5	14.473	0.119
Execução 2	103.506	263.556	140.8	186.3	17.134	0.164
Execução 3	49.381	76.513	151.0	145.7	19.082	0.173
Execução 4	1202.545	1278.542	156.0	155.4	47.094	0.425
Execução 5	60.822	139.235	146.5	195.7	25.651	0.174
Execução 6	42.127	70.715	106.1	111.5	18.777	0.151
Execução 7	109.642	143.164	193.7	221.3	20.156	0.184
Execução 8	45.647	98.086	125.0	144.3	18.889	0.112
Execução 9	37.073	56.359	130.2	145.5	18.589	0.138
Execução 10	41.862	69.177	127.6	148.8	19.214	0.138
Execução 11	47.779	159.541	168.7	246.7	29.091	0.378
Execução 12	57.717	97.411	154.5	189.9	43.754	0.390
Execução 13	52.070	124.340	157.8	205.8	35.720	0.345
Execução 14	304.050	553.099	226.7	241.8	56.774	0.542
Execução 15	102.699	186.683	153.9	158.2	52.756	0.439
Execução 16	120.029	158.474	193.4	198.7	38.102	0.455
Execução 17	137.638	251.343	192.9	220.1	45.354	0.623
Execução 18	42.430	288.613	136.2	182.2	33.248	0.444
Execução 19	164.667	179.651	168.0	172.6	46.397	0.384
Execução 20	117.363	234.863	215.6	222.8	44.422	0.431

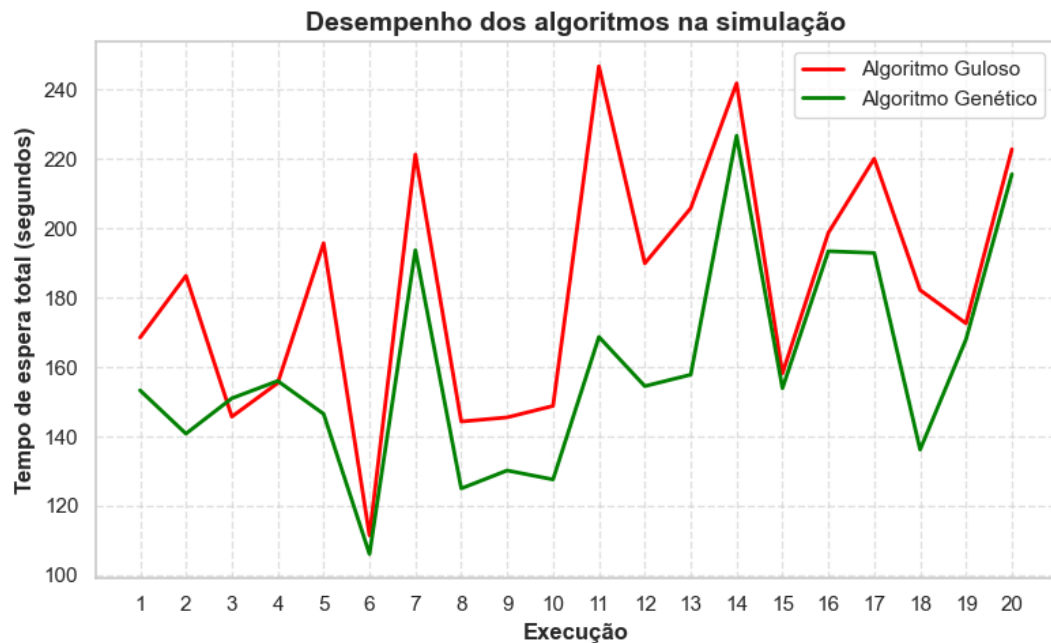
 Melhor resultado
  Pior Resultado

Os resultados mostraram que o algoritmo genético teve um desempenho consistentemente melhor em relação ao algoritmo guloso no critério de *fitness*. Em todas as execuções, o algoritmo genético apresentou notas mais baixas, indicando soluções de melhor qualidade. Em alguns casos, a diferença foi bastante significativa — como na execução 18, em que o *fitness* do algoritmo genético foi cerca de 80% menor do que o do guloso.

Na simulação, a maioria das execuções também favoreceu o algoritmo genético. Mesmo nas poucas vezes em que o algoritmo guloso apresentou desempenho superior, a diferença foi pequena. Isso reforça que, no geral, o algoritmo genético obteve resultados mais eficientes também nesse aspecto.

Já no tempo de execução, o comportamento esperado foi confirmado: o al-

goritmo guloso foi consideravelmente mais rápido. Em todas as execuções, superou o algoritmo genético com ampla vantagem, apresentando tempos ao menos 7500% menores. O gráfico na Figura 12 representa o desempenho de ambos os algoritmos na simulação durante as 20 execuções realizadas.



Fonte: autoria própria

Figura 12 – Desempenho dos algoritmos na simulação do cenário D

O algoritmo genético entregou soluções de melhor qualidade tanto no *fitness* quanto na simulação, com desempenho mais consistente ao longo das execuções. O algoritmo guloso destacou-se exclusivamente pelo tempo de execução, cumprindo seu papel como uma alternativa mais rápida, porém obteve resultados inferiores na maioria dos casos.

6 CONCLUSÕES

Este trabalho apresentou o desenvolvimento e a avaliação de um algoritmo genético para a otimização de agendamentos em academias, com o objetivo de reduzir o tempo de espera dos alunos por equipamentos durante seus treinos. A proposta demonstrou ser eficaz ao equilibrar a alocação de alunos em horários disponíveis, minimizando conflitos pelo uso das máquinas e melhorando a utilização dos recursos da academia.

Os resultados obtidos nos experimentos confirmaram a viabilidade do algoritmo genético para o problema proposto. A seleção por *ranking* destacou-se como o método mais eficiente, produzindo soluções com melhores notas de *fitness* e desempenho superior nas simulações. O *crossover* entrelaçado também se mostrou mais adequado do que o *crossover* de um ponto, reforçando a importância da escolha estratégica dos operadores genéticos. Além disso, a extensão do número de gerações para 150 permitiu melhorias significativas na qualidade das soluções, mesmo com um aumento moderado no tempo de execução.

A comparação com o algoritmo guloso evidenciou as vantagens do algoritmo genético, que obteve soluções de qualidade consistentemente superior, embora com um tempo de processamento maior. Essa diferença justifica o uso do algoritmo genético em cenários onde a otimização detalhada é prioritária, enquanto o algoritmo guloso pode ser uma alternativa viável para situações que demandam respostas rápidas.

Em síntese, o trabalho alcançou seu objetivo geral ao desenvolver um algoritmo genético capaz de gerar soluções satisfatórias para o problema de agendamento em academias, contribuindo tanto para o campo acadêmico, com a aplicação de técnicas de computação evolucionária, quanto para o setor prático, ao oferecer uma ferramenta que pode melhorar a experiência dos usuários e a gestão dos recursos.

6.1 SUGESTÃO PARA TRABALHOS FUTUROS

Como possibilidades para trabalhos futuros, destaca-se a ampliação do modelo atual para contemplar treinos com durações variadas, uma vez que, na implementação desenvolvida, todos os treinos são assumidos como tendo exatamente uma hora de duração. Consequentemente, também seria interessante considerar a alocação de treinos em horários mais flexíveis, permitindo que os alunos iniciem seus treinos em intervalos distintos (por exemplo, às 08:15 ou 09:45), ao invés de restringi-los a horários fixos terminados em ":00".

Outra sugestão é permitir treinos com quantidades variáveis de exercícios,

uma vez que o modelo atual impõe, de forma abstrata, um total fixo de 10 exercícios por treino. Essa flexibilidade permitiria representar melhor a diversidade de rotinas reais dos usuários de academias.

Além disso, é relevante considerar situações em que alunos optam por frequentar a academia sem participar do processo formal de agendamento. Nesse cenário, uma funcionalidade que avalie os treinos desses alunos em relação à distribuição atual dos horários poderia ser utilizada para recomendar os períodos com menor competição pelo uso dos equipamentos. Tal funcionalidade agregaria valor ao sistema ao contribuir para a redução de conflitos e melhorar a experiência dos usuários que ingressam dinamicamente no ambiente.

Outra possibilidade promissora é a utilização do paradigma orientado a agentes para o refinamento da simulação desenvolvida. Esse paradigma permitiria modelar os alunos e equipamentos como agentes autônomos, capazes de interagir e tomar decisões de forma distribuída e dinâmica. Tal abordagem pode ampliar a fidelidade do modelo, capturando comportamentos emergentes e tornando a simulação mais realista e adaptativa a diferentes cenários de uso da academia.

Por fim, sugere-se a exploração de técnicas híbridas, combinando algoritmos genéticos com outras abordagens de otimização, como programação dinâmica ou *machine learning*, para aprimorar ainda mais a qualidade e a eficiência das soluções geradas.

REFERÊNCIAS

- AFFENZELLER, M.; WAGNER, S.; WINKLER, S.; BEHAM, A. **Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications**. 1st edition. ed. Boca Raton, FL: CRC Press, 2009. ISBN 9781584886297.
- ARPACI-DUSSEAU, A. C.; ARPACI-DUSSEAU, R. H. **Operating Systems: Three Easy Pieces**. 1st. ed. [S.l.]: Arpaci-Dusseau Books, 2018. ISBN 978-0996016070.
- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to Algorithms**. 4. ed. Cambridge, MA: The MIT Press, 2022.
- DARWIN, C. **On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life**. London: John Murray, 1859. Edição original.
- DIAS, R. F.; MAGALHÃES DIAS FRINHANI, R. de; ANDRADE, C. E. de. Comparative study of genetic algorithms for the school timetabling problem. *In: Anais do LVI Simpósio Brasileiro de Pesquisa Operacional*. Fortaleza: Galoá, 2024. Acesso em: 08 dez. 2024. Disponível em: <https://proceedings.science/sbpo/sbpo-2024/trabalhos/comparative-study-of-genetic-algorithms-for-the-school-timetabling-problem?lang=pt-br>.
- EIBEN, A. E.; SMITH, J. E. **Introduction to Evolutionary Computing**. 2nd edition. ed. Berlin, Germany: Springer, 2015. ISBN 9783662448731.
- GOH, S. L.; KENDALL, G.; SABAR, N. R. Improved local search approaches to solve the post enrolment course timetabling problem. **European Journal of Operational Research**, v. 261, n. 1, p. 17–29, 2017.
- Gymetrix. **The Importance of Equipment Availability to Gym Customers**. s.d. Disponível em: <https://www.gymetrix.co.uk>, acesso em: junho de 2025.
- DE JONG, K. A. **The Handbook of Evolutionary Computation**. Bristol, UK: Institute of Physics Publishing, Oxford University Press, 1997. ISBN 9780262041942.
- LAWRENCE, S. Resource constrained project scheduling : an experimental investigation of heuristic scheduling techniques (supplement). **Graduate School of Industrial Administration, Carnegie-Mellon University**, 1984. Disponível em: <https://cir.nii.ac.jp/crid/1571980073974705920>.
- LINDEN, R. **Algoritmos Genéticos**. 3ª. ed. Rio de Janeiro, Brasil: Editora Ciência Moderna, 2012. ISBN 9788539901951.
- MICHALEWICZ, Z. **Genetic Algorithms + Data Structures = Evolution Programs**. 3rd edition. ed. Berlin, Germany: Springer, 1996. ISBN 9783642082337.
- MICROSOFT. **Visual Studio Code**. 2024. Disponível em: <https://code.visualstudio.com/>.

MUTH, J.; THOMPSON, G. **Industrial Scheduling**. Prentice-Hall, 1963. (Prentice-Hall International series in management). Disponível em: <https://books.google.com.br/books?id=A5AgAAAAMAAJ>.

PALACH, J. **Parallel Programming with Python**. [S.l.]: Packt Publishing, 2014. ISBN 978-1783288397.

REICHHELD, F. F. The one number you need to grow. **Harvard Business Review**, Harvard Business Publishing, v. 81, n. 12, p. 46–55, 2003.

WITTMAN, B.; MATHUR, A.; KORB, T. **Start Concurrent: An Introduction to Problem Solving in Java with a Focus on Concurrency**. [S.l.]: Purdue University Press, 2013. ISBN 978-1626710092.

WONG, C. H.; GOH, S. L.; LIKOH, J. A genetic algorithm for the real-world university course timetabling problem. *In*: **2022 IEEE 18th International Colloquium on Signal Processing Applications (CSPA)**. Selangor, Malaysia: [s.n.], 2022. p. 46–50.

ZHAO, H.; KONG, F. Research and applications of shop scheduling based on genetic algorithms. **Brazilian Archives of Biology and Technology**, Instituto de Tecnologia do Paraná - Tecpar, v. 59, n. spe, p. e16160545, 2016. ISSN 1516-8913. Disponível em: <https://doi.org/10.1590/1678-4324-2016160545>.

A ARTIGO ACADÊMICO

GYMSYS: Algoritmo Genético para Otimização de Agendamentos em Academias

Arthur Machado Capaverde¹, Luciana de Oliveira Rech¹

¹Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brazil

arthur.capaverde@grad.ufsc.br, luciana.rech@ufsc.br

Abstract. *Genetic algorithms are an efficient solution for complex optimization problems, such as scheduling. Although well-established in many fields, their application in gym management remains underexplored. This project proposes a method to schedule gym sessions by minimizing equipment conflicts, reducing wait times, and optimizing resource usage. The developed genetic algorithm generates efficient timetables in feasible time, improving user experience and facility utilization.*

Resumo. *Algoritmos genéticos destacam-se como uma solução eficiente para problemas complexos de otimização, como o agendamento de horários. Apesar de seu uso consolidado em diversas áreas, sua aplicação em academias ainda é pouco explorada. Este projeto propõe um método para alocar alunos em horários de forma a minimizar conflitos no uso de equipamentos, reduzindo filas e otimizando recursos. O algoritmo genético desenvolvido busca gerar grades de horários eficientes em tempo viável, melhorando a experiência dos usuários e o aproveitamento da infraestrutura.*

1. Introdução

A Computação Evolucionária, um ramo da Inteligência Artificial inspirado na evolução biológica, busca resolver problemas complexos por meio de processos como seleção, crossover e mutação. Dentro desse campo, os Algoritmos Genéticos se destacam por sua capacidade de explorar grandes espaços de soluções e encontrar respostas satisfatórias em tempo hábil, sendo amplamente utilizados em problemas de otimização e combinação.

Este trabalho propõe aplicar algoritmos genéticos na otimização de grades de horários em academias, um cenário pouco explorado na literatura. A ideia é alocar alunos em horários disponíveis considerando os equipamentos utilizados por cada um, reduzindo a concorrência por máquinas e otimizando o uso dos recursos. A proposta envolve a análise da disponibilidade de cada aluno e a geração de horários otimizados que minimizem o uso simultâneo de equipamentos por diferentes alunos.

Além do valor prático para a gestão de academias, o projeto contribui cientificamente ao explorar a adaptação e eficácia dos algoritmos em um novo domínio. Durante seu desenvolvimento, são abordadas técnicas da computação evolucionária, estruturas de dados, parametrização de algoritmos e a modelagem de problemas reais para soluções computacionais, reforçando a relevância acadêmica e prática da pesquisa.

2. Fundamentação Teórica

Os Algoritmos Genéticos são uma técnica pertencente à computação evolucionária, inspirada nos princípios da seleção natural [Darwin 1859], na qual uma população de soluções evolui ao longo do tempo para resolver problemas complexos de otimização. Cada indivíduo da população representa uma solução candidata, codificada em forma de genes. O algoritmo avalia a qualidade de cada solução por meio de uma função de fitness, e utiliza operadores como crossover e mutação para gerar novas soluções a partir das mais promissoras. Entre suas aplicações destacam-se problemas em que a busca por soluções ótimas ou quase ótimas em espaços muito grandes seria inviável por métodos tradicionais. A estrutura dos algoritmos genéticos permite lidar com múltiplas soluções simultaneamente, o que favorece a exploração do espaço de busca e contribui para a sua robustez em cenários variados [Linden 2012].

O ciclo de execução de um algoritmo genético é composto por diversas etapas interdependentes. Inicialmente, é gerada uma população de soluções aleatórias que representem possíveis respostas ao problema. Em seguida, cada indivíduo é avaliado por meio de uma função de fitness, que mede sua adequação ao problema. A seleção escolhe os indivíduos mais aptos para a reprodução, e o operador de crossover combina suas características para formar descendentes. A mutação introduz variações aleatórias nos genes, promovendo diversidade genética e evitando a convergência prematura. Esse ciclo é repetido por várias gerações até que se atinja um critério de parada, como o número máximo de gerações ou a estagnação na melhoria das soluções. Estratégias como elitismo e diferentes métodos de seleção — torneio, roleta ou ranking — são empregadas para melhorar o desempenho do algoritmo [Linden 2012].

A programação paralela é uma abordagem que visa aumentar a eficiência computacional ao permitir a execução simultânea de múltiplas tarefas por diferentes núcleos de processamento. Dois conceitos centrais nesse modelo são o paralelismo — a execução efetiva de operações em paralelo — e a concorrência — a coordenação entre tarefas que compartilham recursos. Em ambientes com múltiplas threads, a programação paralela permite simular o comportamento de diversos agentes atuando de forma simultânea, como ocorre em sistemas reais com múltiplas interações concorrentes. No entanto, essa simultaneidade traz desafios, como as condições de corrida, que ocorrem quando duas ou mais threads acessam uma mesma variável compartilhada de forma descoordenada. Para evitar tais conflitos, utilizam-se mecanismos de exclusão mútua, como mutexes e semáforos, que controlam o acesso a regiões críticas do código. Dessa forma, a programação paralela torna-se essencial para simulações que buscam representar com fidelidade o comportamento simultâneo de múltiplos agentes em cenários complexos [Wittman et al. 2013].

3. Proposta

A abordagem proposta realiza a alocação otimizada de alunos em horários disponíveis para treino em academias, visando reduzir o tempo de espera por equipamentos. Utiliza-se um algoritmo genético para agrupar alunos com treinos distintos em horários comuns, minimizando a competição simultânea por máquinas e promovendo melhor aproveitamento dos recursos da academia.

Cada aluno é representado por uma tupla composta por duas sub-tuplas de inteiros. A primeira sub-tupla define os horários em que o aluno tem disponibilidade para treinar,

por exemplo, [14, 16, 17]. Todos os alunos devem indicar pelo menos um horário, não havendo limite máximo. A segunda sub-tupla representa o treino do aluno, ou seja, os tipos de máquinas que ele utilizará. Essa sub-tupla contém exatamente 10 inteiros entre 0 e 9, cada um correspondendo a um dos 10 tipos de equipamentos da academia. Um exemplo de treino seria [3, 1, 3, 4, 5, 2, 3, 9, 9, 0]. Para refletir práticas realistas, foi estabelecido um limite de até três ocorrências de uma mesma máquina por treino, evitando representações com uso excessivo de um único equipamento. A ordem dos exercícios declarados não é relevante, visto que a implementação abstrai os detalhes do plano de treino, considerando apenas a frequência de uso de cada máquina por horário. Assume-se que, na prática, o aluno pode adaptar a ordem de execução de acordo com a disponibilidade dos aparelhos.

A lista de alunos, denominada lista origem, serve de base para a geração inicial da população de soluções. Cada indivíduo representa uma possível alocação dos alunos nos horários em que estão disponíveis, construída de forma aleatória. A função de avaliação (fitness) mede a qualidade de cada solução com base no número total de ocorrências de cada tipo de máquina por horário. Soluções com alta sobreposição no uso de máquinas por alunos em um mesmo horário tendem a receber notas de fitness inferiores, já que indicam maior probabilidade de espera pelos equipamentos.

O modelo também contempla cenários em que a demanda excede a capacidade de alocação. Caso não seja possível alocar um aluno em nenhum dos horários disponíveis, ele é descartado da solução. Cada indivíduo armazena a contagem total de descartes em uma variável específica, sem manter os dados desses alunos na estrutura final. Essa modelagem permite representar ambientes com recursos limitados, como academias com alta demanda, e buscar soluções viáveis mesmo sob restrição. A solução incorpora a definição da estrutura dos indivíduos, aplicação de operadores genéticos de seleção, crossover e mutação, além de uma função de avaliação baseada na competição por máquinas e um critério de parada baseado no número de gerações.

3.1. Modelagem do Cromossomo

O cromossomo representa uma solução candidata à alocação dos alunos. Ele é composto por sete genes: seis correspondem aos horários disponíveis (das 12h às 17h) e o sétimo registra a quantidade de alunos não alocados (valor de descarte). Cada horário é modelado como uma lista de até 20 alunos. A posição dos genes de horário vai de 0 a 5, refletindo a sequência das faixas horárias. A Figura 1 ilustra essa modelagem.

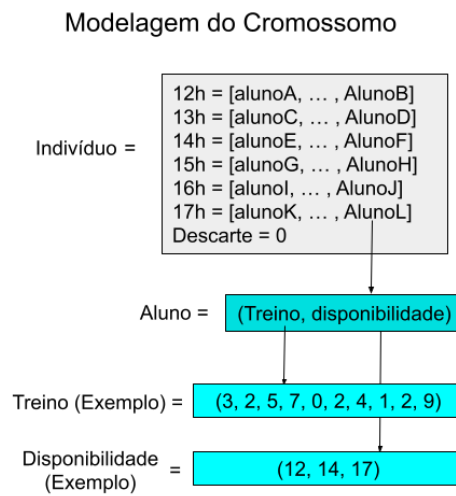


Figure 1. Modelagem do cromossomo. Elaborado pelo autor.

3.2. Arquitetura Geral do Algoritmo

O algoritmo trabalha com uma população de 50 indivíduos, cada um contendo seis listas de horários e uma variável de descarte. Os alunos são mapeados para essas listas com base em sua disponibilidade, ajustada pela subtração de 12 (por exemplo, o horário 14 mapeia para a posição 2). Além disso, uma lista complementar armazena o número de instâncias de cada tipo de máquina disponível. A arquitetura geral é apresentada na Figura 2.

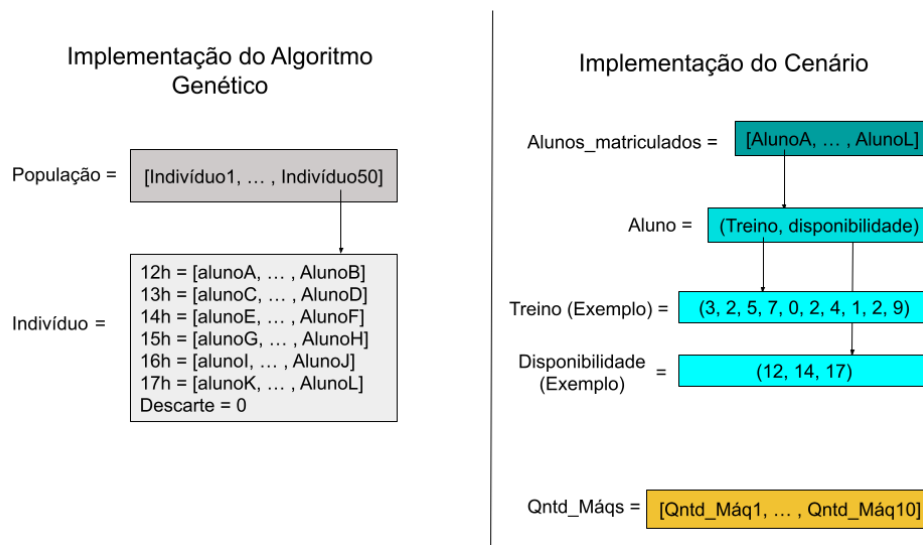


Figure 2. Arquitetura do algoritmo. Elaborado pelo autor.

3.3. Ciclo Evolutivo

O processo de evolução segue as etapas clássicas de algoritmos genéticos: inicialização da população, avaliação por função de aptidão, seleção, cruzamento, mutação e substituição. A cada geração, os melhores indivíduos são preservados por elitismo, enquanto novos indivíduos são gerados por operadores genéticos até que o critério de parada — fixado em 150 gerações — seja atingido.

3.3.1. Inicialização

A população inicial é construída distribuindo alunos da lista de entrada em horários compatíveis, respeitando a capacidade máxima de 20 alunos por horário. Caso não haja vagas em horários compatíveis, o aluno é descartado, incrementando a variável de descarte do indivíduo. Essa distribuição é feita aleatoriamente a partir de cópias da lista original.

3.3.2. Seleção

A seleção é feita por ranking: os indivíduos são ordenados por aptidão, e cada um recebe uma probabilidade de seleção proporcional ao seu ranking. A fórmula utilizada é:

$$P(i) = \frac{N + 1 - r_i}{\sum_{j=1}^N (N + 1 - r_j)} \quad (1)$$

onde $P(i)$ é a probabilidade de seleção do indivíduo i , N é o tamanho da população, e r_i é o ranking do indivíduo.

3.3.3. Operadores de Crossover

Dois operadores de crossover foram testados:

Entrelaçado: alterna entre os pais na ordem da lista de alunos. Para cada aluno, tenta alocá-lo no mesmo horário em que aparece no pai. Caso o horário esteja cheio, o aluno é direcionado a uma lista de repescagem e, se não puder ser realocado, é descartado. As Figuras 3 e 4 ilustram o processo.

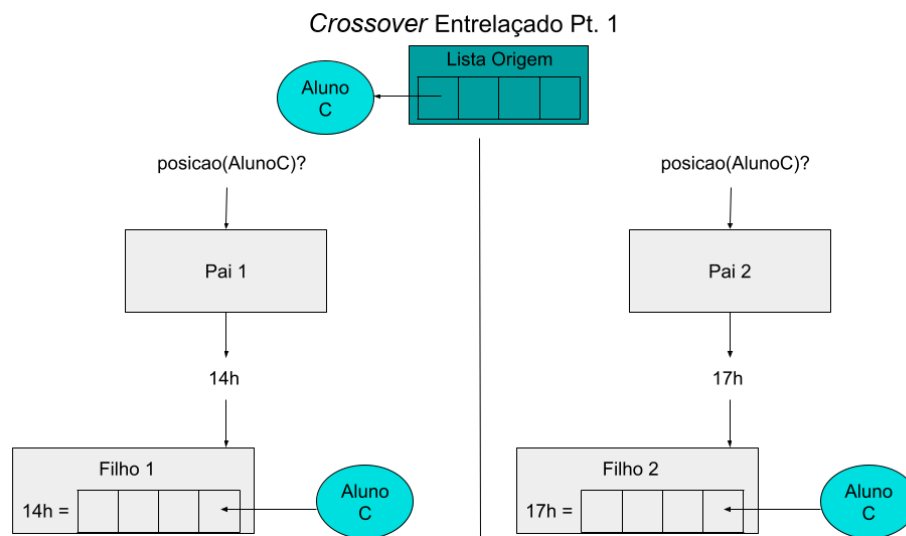


Figure 3. Crossover entrelaçado — parte 1. Elaborado pelo autor.

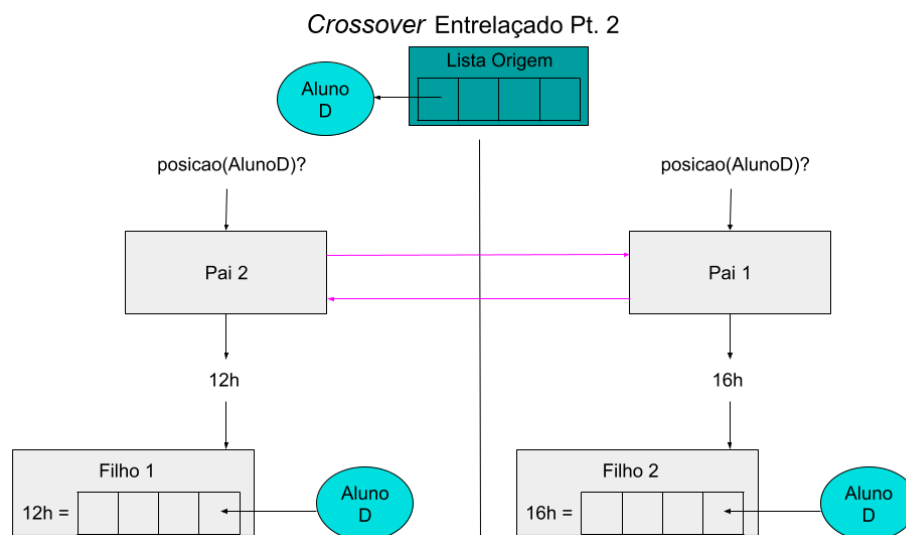


Figure 4. Crossover entrelaçado — parte 2. Elaborado pelo autor.

Um ponto: divide a lista de alunos ao meio. A primeira metade é herdada diretamente do pai correspondente, e a segunda, do outro. O mesmo mecanismo de repescagem e descarte é aplicado quando necessário.

3.3.4. Mutação

A mutação troca os horários de dois alunos que possuam disponibilidade mútua. Essa troca respeita as restrições de capacidade dos horários e visa manter a viabilidade da solução. A taxa de mutação adotada foi de 20%, valor considerado adequado para promover variações significativas em populações com genes extensos.

3.4. Função de Avaliação

A função de aptidão penaliza soluções com uso desbalanceado de máquinas e alto número de alunos descartados. Para cada horário, calcula-se:

$$h = \sum_{n=1}^{10} 2^{x_n} \quad (2)$$

onde x_n representa o número de vezes que a máquina n é utilizada. A penalidade por descarte é:

$$8^d \quad (3)$$

em que d é o número de alunos não alocados. A soma total que define a nota de fitness do indivíduo é:

$$F = \sum_{i=1}^6 h_i + 8^d \quad (4)$$

Esse modelo prioriza soluções que alocam o maior número possível de alunos com o menor conflito possível por equipamentos.

3.5. Critério de parada

O critério de parada adotado neste trabalho baseia-se em um número fixo de gerações. O algoritmo é executado por 150 gerações, encerrando-se ao final desse ciclo. Essa escolha permite que o algoritmo explore o espaço de soluções de forma suficientemente ampla, garantindo tempo hábil para que melhorias significativas ocorram sem comprometer a eficiência da execução.

4. Análise de Experimentos

Com o objetivo de avaliar a qualidade das soluções geradas pelo algoritmo genético, foi desenvolvida uma simulação que representa o funcionamento de uma academia real. A simulação permite mensurar o impacto das diferentes configurações do algoritmo no tempo total de espera dos alunos durante o uso dos equipamentos. Foram testadas diversas variações, incluindo diferentes métodos de seleção, operadores de crossover,

número de gerações e comparação com uma heurística de busca local, o algoritmo guloso [Cormen et al. 2022]. A análise considera tanto os valores da função de avaliação quanto os resultados empíricos da simulação, incluindo o tempo de execução.

4.1. Descrição da Simulação

A simulação modela uma academia com 10 tipos distintos de equipamentos, cada um representado por um semáforo. O valor inicial de cada semáforo corresponde à quantidade de instâncias disponíveis daquele equipamento, permitindo simular a competição entre usuários em tempo real. Os semáforos são armazenados em uma lista, sendo o índice da lista equivalente ao identificador do equipamento. Por exemplo, se há três instâncias do equipamento 0, o semáforo na posição 0 da lista é inicializado com valor 3.

Cada aluno é representado por uma thread que executa a função treinar, simulando o comportamento do usuário durante seu horário de treino. O treino consiste em uma lista de 10 números inteiros entre 0 e 9, indicando a sequência de equipamentos que o aluno pretende utilizar.

Durante a simulação, a thread tenta adquirir um dos equipamentos indicados em seu treino. Caso pelo menos um esteja disponível, o semáforo correspondente é adquirido, e a thread permanece utilizando o equipamento por um tempo fixo associado àquele tipo de máquina. Se nenhuma das opções estiver disponível, a thread aguarda 0,1 segundo antes de tentar novamente, adicionando esse intervalo ao tempo de espera acumulado do aluno. Esse processo se repete até que todas as máquinas necessárias tenham sido utilizadas.

A simulação é executada separadamente para cada um dos seis horários disponíveis, com os alunos alocados em cada horário competindo entre si pelos recursos da academia. Ao término do treino de todos os alunos, a simulação acumula o tempo total de espera global, permitindo comparações quantitativas entre diferentes agendamentos.

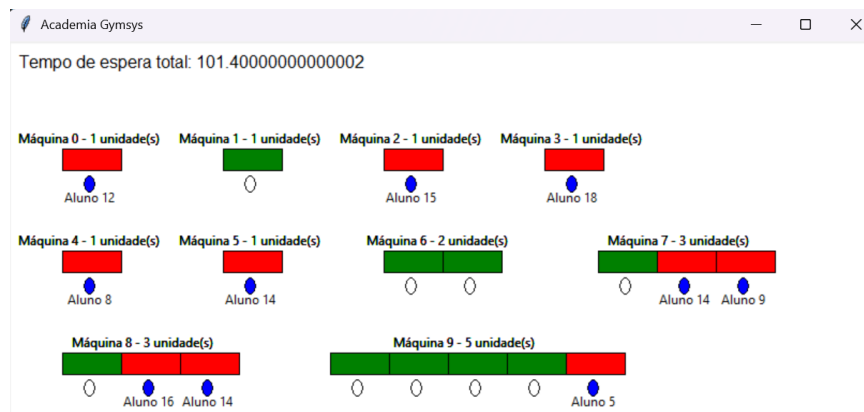


Figure 5. Interface da simulação com academia em uso. Elaborado pelo autor.

4.2. Parâmetros da Simulação

Os parâmetros da simulação foram definidos para refletir um ambiente realista de academia. A quantidade de instâncias por tipo de equipamento foi configurada da seguinte

forma: 1, 1, 1, 1, 1, 1, 2, 3, 3 e 5, para os tipos de equipamento de 0 a 9, respectivamente. Os tempos de uso atribuídos a cada tipo foram: 0,3, 0,3, 0,4, 0,4, 0,5, 0,5, 0,7, 1,0, 1,2 e 1,5 segundos.

A escolha desses valores foi baseada na observação prática de que equipamentos com maior tempo de uso — como esteiras e bicicletas — geralmente estão disponíveis em maior quantidade, enquanto equipamentos utilizados por períodos mais curtos — como máquinas de treino localizado — tendem a ser mais escassos.

Essa abordagem permitiu uma avaliação prática e visual do desempenho do algoritmo em condições realistas, destacando o impacto das estratégias de alocação no tempo de espera dos alunos e na utilização eficiente dos equipamentos.

5. Resultados

Quatro cenários experimentais foram definidos para calibrar os parâmetros do algoritmo e avaliar seu desempenho. As métricas consideradas em cada cenário foram: a nota de *fitness* do melhor indivíduo da geração final, o tempo total de espera dos alunos na simulação e o tempo de execução do algoritmo. A combinação dessas métricas permite mensurar a eficácia do método na geração de soluções viáveis dentro de um intervalo de tempo aceitável.

Cada cenário foi executado 20 vezes de forma independente, utilizando listas distintas de alunos em cada execução. Em comparações paramétricas — como troca de método de seleção, tipo de *crossover* ou número de gerações — a mesma sequência de listas foi mantida para todas as variações. Essa estratégia assegura que as diferenças observadas entre os resultados sejam atribuídas exclusivamente às mudanças nos parâmetros, e não a variações nos dados de entrada.

No **Cenário A**, avalia-se o impacto do método de seleção, comparando as abordagens por torneio, *ranking* e roleta. O **Cenário B** investiga o efeito do operador de recombinação, contrastando os métodos de *crossover* entrelaçado e de um ponto. O **Cenário C** analisa o impacto da extensão do número de gerações, comparando os resultados obtidos na 100ª e na 150ª geração de uma mesma execução, mantendo o critério de parada em 150 gerações. Por fim, o **Cenário D** compara o desempenho da melhor configuração do algoritmo genético — definida a partir dos cenários anteriores — com uma heurística gulosa implementada para o mesmo problema de agendamento.

Salvo nos casos em que o parâmetro estava sendo alterado intencionalmente para avaliação, os valores-padrão utilizados foram: população de 50 indivíduos, elitismo de 2 indivíduos, taxa de mutação de 20%, método de seleção por *ranking*, *crossover* entrelaçado e critério de parada em 150 gerações.

5.1. Cenário A: Métodos de Seleção

Neste cenário, avaliou-se o desempenho do algoritmo ao empregar três métodos distintos de seleção de pais: seleção por torneio, por *ranking* e por roleta. Esses métodos determinam quais indivíduos da população serão responsáveis pela geração da próxima população, influenciando diretamente a convergência do algoritmo.

A seleção por *ranking* apresentou o melhor desempenho em todas as execuções, gerando indivíduos com menor nota de *fitness*. A seleção por torneio ficou consis-

temente em segundo lugar. Embora a roleta tenha ocasionalmente superado o torneio, a diferença foi marginal, e seu desempenho médio permaneceu inferior nas demais execuções.

Nos testes de simulação, os resultados acompanharam o padrão observado na *fitness*: a seleção por ranking resultou, na maioria dos casos, nos menores tempos de espera total. Quando não obteve o melhor resultado absoluto, a diferença para o melhor método foi pequena. A seleção por torneio manteve desempenho intermediário, enquanto a roleta obteve os piores resultados em tempo de espera na maioria das execuções.

Quanto ao tempo de execução do algoritmo, a seleção por torneio foi a mais eficiente, devido à sua menor complexidade computacional. As estratégias por ranking e roleta apresentaram tempos de execução semelhantes, dado que ambas exigem o cálculo de probabilidades específicas para cada indivíduo da população.

Diante dos resultados, a seleção por ranking foi adotada como configuração padrão do algoritmo, considerando seu desempenho superior nas métricas avaliadas. A seleção por roleta, por sua vez, foi a menos eficaz, o que está alinhado com sua limitação em cenários com alta disparidade entre os valores de *fitness*, onde tende a concentrar as chances em poucos indivíduos e comprometer a diversidade da população.

5.2. Cenário B: Crossover

No cenário B, avaliou-se o desempenho do algoritmo genético utilizando dois métodos distintos de crossover: o crossover entrelaçado e o crossover de um ponto. O primeiro método distribui os alunos da lista de origem de forma alternada entre os indivíduos filhos, enquanto o segundo realiza a distribuição de maneira contínua, alternando apenas a partir da metade da lista.

Os resultados demonstraram uma vantagem geral do crossover entrelaçado. Em termos de *fitness*, este método superou o crossover de um ponto em 65% das execuções, com diferenças variando entre 5% e 33%. Nas execuções em que o crossover de um ponto obteve melhor desempenho, as melhorias variaram de 3% a 24%. Na simulação, o crossover entrelaçado também apresentou melhor desempenho em 65% dos casos, sendo que 76% dessas execuções coincidiram com as melhores notas de *fitness*, indicando uma forte correlação entre o valor de aptidão calculado e o desempenho prático.

Quanto ao tempo de execução, não foram observadas diferenças significativas entre os métodos, com variações mínimas na maioria dos casos e vantagens pontuais distribuídas de forma equilibrada. Diante desses resultados, optou-se pelo *crossover* entrelaçado como método padrão, devido ao seu desempenho mais consistente tanto em termos de *fitness* quanto em simulações práticas. O tempo de execução, por não apresentar diferenças relevantes, não foi um fator determinante na escolha.

5.3. Cenário C: Número de Gerações

No cenário C, avaliou-se o desempenho do algoritmo ao estender a execução em 50 gerações adicionais, totalizando 150 gerações. Comparou-se a solução obtida na 100ª geração com aquela gerada ao final da 150ª, observando que métricas como a nota de *fitness* tendem obrigatoriamente a apresentar alguma melhora devido à continuidade da execução. Por esse motivo, a análise considerou a proporção dessa melhora em relação ao

aumento no tempo de execução. Já a avaliação das soluções na simulação não apresenta ressalvas metodológicas, pois não há garantias de que a solução após 150 gerações terá desempenho superior à da 100ª geração.

Os resultados de fitness apresentaram melhoras variando entre -1% e -32%, demonstrando que o ganho não segue uma relação linear com o aumento de 50% no número de gerações. Esse comportamento é consistente com o princípio de convergência em algoritmos evolutivos, onde a população tende a se estabilizar em soluções ótimas locais após determinado número de gerações. A simulação revelou um cenário misto, com casos de melhora (-1% a -14%) e piora (1% a 12%) no desempenho, porém com claro predomínio de melhorias: em cinco execuções os ganhos superaram 10%, enquanto apenas uma execução apresentou piora equivalente.

O tempo de execução aumentou entre 40% e 80%, superando a previsão teórica de 50% associada à mudança de 100 para 150 gerações. Apesar disso, o tempo final permaneceu em níveis aceitáveis, considerando que a duração original já era reduzida. Embora os resultados tenham ficado abaixo das expectativas iniciais, as melhorias no desempenho foram significativas. O aumento no tempo de processamento, ainda que relevante, não inviabilizou a operação do algoritmo. Conclui-se que a decisão de manter 150 gerações mostrou-se adequada, pois equilibrou ganhos de performance com um tempo de execução dentro do esperado, especialmente considerando que a maioria dos casos testados apresentou resultados positivos.

5.4. Cenário D: Algoritmo Genético Vs. Algoritmo Guloso

No Cenário D, realizou-se uma avaliação comparativa entre o algoritmo genético desenvolvido e uma abordagem gulosa para alocação de alunos em horários de academia. O algoritmo guloso, que em cada etapa faz a escolha localmente ótima [Cormen et al. 2022], foi implementado seguindo uma estratégia determinística: para cada aluno, avaliam-se todas as possíveis alocações nos horários disponíveis, selecionando aquela que minimiza imediatamente a função fitness (a mesma utilizada pelo algoritmo genético), conforme ilustrado na Figura 6.

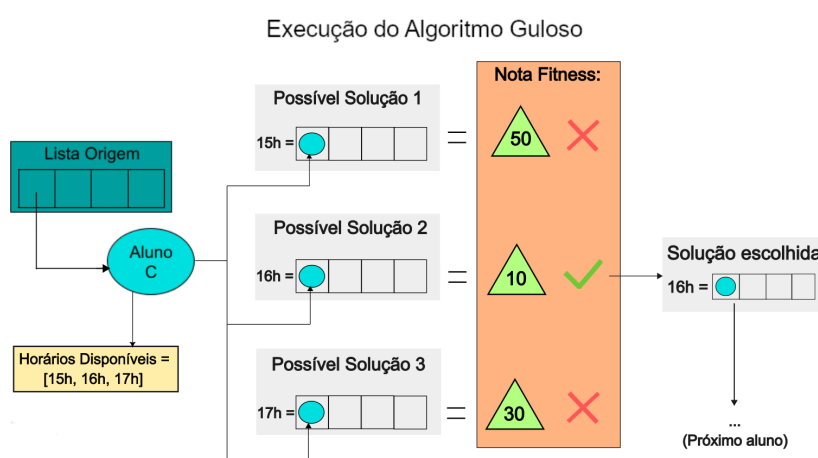


Figure 6. Execução do algoritmo guloso. Elaborado pelo autor.

Para o algoritmo genético, utilizaram-se os parâmetros que obtiveram melhor desempenho nos testes preliminares. Os resultados demonstraram que o algoritmo genético superou consistentemente a abordagem gulosa em qualidade de solução. Em todas as execuções, apresentou valores de fitness inferiores, com diferenças significativas em alguns casos. Na simulação, o algoritmo genético também obteve melhor desempenho na maioria dos casos, com as raras vantagens do método guloso sendo marginais.

Quanto ao tempo de execução, o algoritmo guloso confirmou sua principal vantagem, sendo consideravelmente mais rápido em todas as execuções, com tempos pelo menos 7500% menores. Contudo, essa vantagem operacional não se traduziu em melhor qualidade de solução. Os resultados indicam que, para este problema específico, a complexidade adicional do algoritmo genético justifica-se pela obtenção de soluções significativamente melhores, tanto em termos de fitness quanto no desempenho prático na simulação, embora com um custo computacional substancialmente maior.

6. Conclusões

Os experimentos demonstraram que a abordagem é eficaz ao distribuir melhor os alunos nos horários disponíveis, minimizando conflitos pelo uso de máquinas. Entre os métodos utilizados, a seleção por ranking apresentou o melhor desempenho, gerando soluções mais

eficientes, enquanto o crossover entrelaçado superou o crossover de um ponto. O aumento do número de gerações para 150 também contribuiu para melhorar significativamente a qualidade das soluções, com impacto moderado no tempo de execução.

A comparação com o algoritmo guloso evidenciou a superioridade do algoritmo genético em termos de qualidade das soluções, ainda que com maior tempo de processamento. Isso torna o algoritmo genético mais indicado para cenários onde a qualidade da alocação é prioritária, enquanto o algoritmo guloso pode atender bem a situações que exigem rapidez na resposta.

Como propostas para trabalhos futuros, sugere-se a adaptação do modelo para treinos com durações variadas e horários mais flexíveis, refletindo melhor a realidade dos usuários. Também seria relevante permitir quantidades variáveis de exercícios por treino e incorporar alunos que não utilizam o sistema de agendamento, oferecendo a esses usuários sugestões baseadas na ocupação da academia. Além disso, a adoção do paradigma orientado a agentes pode enriquecer a simulação ao permitir interações dinâmicas entre alunos e equipamentos. Por fim, a combinação do algoritmo genético com outras técnicas, como programação dinâmica ou aprendizado de máquina, representa um caminho promissor para aprimorar a qualidade e a eficiência das soluções geradas.

References

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2022). *Introduction to Algorithms*. MIT Press, Cambridge, MA, 4th edition.
- Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, London.
- Linden, R. (2012). *Algoritmos Genéticos*. Editora Ciência Moderna, Rio de Janeiro, Brasil.
- Wittman, B., Mathur, A., and Korb, T. (2013). *Start Concurrent: An Introduction to Problem Solving in Java with a Focus on Concurrency*. Purdue University Press.

B CÓDIGO FONTE

O código fonte desenvolvido para este trabalho está disponível no repositório público do GitHub, podendo ser acessado através do seguinte link:

https://github.com/Arthurcapa/Gym_sys