Revista Brasileira de Informática na Educação – RBIE
Brazilian Journal of Computers in Education
(ISSN online: 2317-6121; print: 1414-5685)
https://sol.sbc.org.br/journals/index.php/rbie

Submission: 31/Out/2024; 1st round notif.: 22/Jan/2025; New version: 10/Fev/2025; 2nd round notif.: 20/Mar/2025;
Camera ready: 06/Abr/2025; Edition review: 07/May/2025; Available online: 21/Aug/2025; Published: 21/Aug/2025;

Estratégias para Extração de Padrões Sequenciais no Ensino de Algoritmos: Uma Comparação entre Algoritmos Clássicos e Metaheurísticas Evolutivas

Title: Strategies for Extracting Sequential Learning Patterns in the Teaching of Algorithms: A Comparison between Classical Algorithms and Evolutionary Metaheuristics

Título: Estrategias para la Extracción de Patrones Secuenciales en la Enseñanza de Algoritmos: Una Comparación entre Algoritmos Clásicos y Metaheurísticas Evolutivas

Djefferson Smith Santos Maranhão
Universidade Federal do Maranhão (UFMA) ORCID: 0000-0002-8390-4878
djefferson.maranhao@discente.ufma.br

Carlos de Salles Soares Neto
Universidade Federal do Maranhão, campus São Luís
ORCID: 0000-0002-6800-1881
carlos.salles@ufma.br

Resumo

A descoberta de padrões sequenciais no comportamento dos estudantes em ambientes de ensino de programação online é essencial para a personalização do aprendizado e otimização do processo educacional. Com a transformação digital no ensino, plataformas voltadas ao aprendizado de algoritmos ganharam destaque, mas ainda carecem de um sequenciamento personalizado de atividades que auxilie na progressão educacional. Neste cenário, a aplicação da Mineração de Padrões Sequenciais (MPS) pode revelar padrões de aprendizagem e desafios recorrentes enfrentados pelos alunos, tornando-se uma ferramenta valiosa para personalização da experiência de aprendizagem. Assim, o presente estudo compara duas abordagens de MPS: algoritmos clássicos e metaheurísticas evolutivas, para avaliar suas eficácias na detecção de padrões. A pesquisa analisa as interações de 313 alunos em um ambiente de ensino de algoritmos, observando o desempenho das abordagens quanto à precisão, custo computacional e aplicabilidade. Os algoritmos clássicos mostram-se eficientes em bases menores, enquanto as metaheurísticas evolutivas revelam padrões complexos com maior eficácia em grandes volumes de dados. Os resultados indicam que ambas as abordagens podem beneficiar o ambiente de ensino, ajudando o professor a antecipar sinais de desmotivação e intervir de forma proativa para manter o aluno interessado e ativo na plataforma. Como contribuição, demonstra-se que, tanto os algoritmos clássicos quanto as metaheurísticas evolutivas, podem gerar insights valiosos, como a necessidade de revisar um problema específico ou fornecer mais exemplos práticos aos alunos.

Palavras-chave: *Mineração de Padrões Sequenciais; Metaheurísticas Evolutivas; Personalização do Ensino de Programação.*

Abstract

Discovering sequential patterns in student behavior in online programming teaching environments is essential for personalizing learning and optimizing the educational process. With the digital transformation in education, platforms focused on algorithm learning have gained prominence but still lack personalized sequencing of activities to assist in academic progression. In this scenario, the application of Sequential Pattern Mining (SPM) can reveal learning patterns and recurring challenges students face, becoming a valuable tool for personalizing the learning experience. Thus, the present study compares two SPM approaches: classical algorithms and evolutionary metaheuristics, to

Cite as: Maranhão, D. S. S., Soares Neto, C. S. (2025). Estratégias para Extração de Padrões Sequenciais no Ensino de Algoritmos: Uma Comparação entre Algoritmos Clássicos e Metaheurísticas Evolutivas. Revista Brasileira de Informática na Educação, 33, 912-942. https://doi.org/10.5753/rbie.2025.5119.

evaluate their effectiveness in detecting patterns. The research analyzes the interactions of 313 students in an algorithm teaching environment, observing the performance of the approaches in terms of accuracy, computational cost, and applicability. Classical algorithms prove efficient in smaller datasets, while evolutionary metaheuristics reveal complex patterns more effectively in large volumes of data. The results indicate that both approaches can benefit the teaching environment, helping the teacher anticipate signs of demotivation and intervene proactively to keep the student interested and active on the platform. As a contribution, it is demonstrated that both classical algorithms and evolutionary metaheuristics can generate valuable insights, such as the need to review a specific problem or provide more practical examples to students.
Keywords: *Sequential Pattern Mining; Evolutionary Metaheuristics; Personalization of Programming Education.*

Resumen

El descubrimiento de patrones secuenciales en el comportamiento de los estudiantes en entornos de enseñanza de programación en línea es esencial para la personalización del aprendizaje y la optimización del proceso educativo. Con la transformación digital en la enseñanza, las plataformas orientadas al aprendizaje de algoritmos han ganado protagonismo, pero aún carecen de un secuenciamiento personalizado de actividades que ayude en la progresión educativa. En este escenario, la aplicación de la Minería de Patrones Secuenciales (MPS) puede revelar patrones de aprendizaje y desafíos recurrentes enfrentados por los alumnos, convirtiéndose en una herramienta valiosa para la personalización de la experiencia de aprendizaje. Así, el presente estudio compara dos enfoques de MPS: algoritmos clásicos y metaheurísticas evolutivas, para evaluar sus eficacias en la detección de patrones. La investigación analiza las interacciones de 313 alumnos en un entorno de enseñanza de algoritmos, observando el desempeño de los enfoques en cuanto a precisión, costo computacional y aplicabilidad. Los algoritmos clásicos se muestran eficientes en bases de datos más pequeñas, mientras que las metaheurísticas evolutivas revelan patrones complejos con mayor eficacia en grandes volúmenes de datos. Los resultados indican que ambos enfoques pueden beneficiar el entorno de enseñanza, ayudando al profesor a anticipar señales de desmotivación e intervenir de forma proactiva para mantener al alumno interesado y activo en la plataforma. Como contribución, se demuestra que tanto los algoritmos clásicos como las metaheurísticas evolutivas pueden generar insights valiosos, como la necesidad de revisar un problema específico o proporcionar más ejemplos prácticos a los alumnos.
Palabras clave: *Minería de Patrones Secuenciales; Metaheurísticas Evolutivas; Personalización de la Enseñanza de Programación*

1 Introdução

Nos últimos anos, o ensino de programação vem passando por uma transformação significativa, saindo do formato tradicional em sala de aula para ambientes de ensino *online*. Isso pode ser atribuído, em grande parte, à mudança global para o ensino à distância provocada pela pandemia da COVID-19, que acelerou a adoção de ferramentas e recursos de aprendizagem *online*. O surgimento de plataformas especializadas na preparação para entrevistas técnicas também tem contribuído para difundir a importância dos algoritmos e da resolução de problemas como um meio para alcançar melhores posições no mercado de trabalho.

A facilidade de acesso a essas plataformas vem democratizando a educação, permitindo que alunos de todo o mundo consigam adquirir novas competências em programação de qualquer lugar ligado à Internet. Com a multiplicação das plataformas *online*, o conteúdo ofertado aumentou significativamente, assim como o volume de dados gerado pela navegação dos alunos. No entanto, ainda se observa uma lacuna no quesito da personalização do aprendizado. Nesse sentido, acredita-se que a chave para melhorar a experiência dos estudantes está na análise cuidadosa de suas interações com a plataforma.

Uma abordagem promissora para esse tipo de análise é a Mineração de Padrões Sequenciais (MPS), que permite identificar padrões frequentes de comportamento dos alunos ao longo do tempo. Essa abordagem pode ser utilizada para prever o desempenho dos estudantes, detectar dificuldades no aprendizado e sugerir intervenções personalizadas. Padrões de comportamento frequentes revelam como os alunos interajem com os recursos didáticos, contribuindo para a melhoria contínua do ensino online (Zhang & Paquette, 2023; Zhou et al., 2010)

Os algoritmos clássicos de MPS, como PrefixSpan (Pei et al., 2001), GSP (Srikant & Agrawal, 1996) e SPADE (Zaki, 2000), utilizam métodos determinísticos para explorar sequências frequentes, aplicando estratégias de poda para reduzir a quantidade de padrões candidatos e aumentar a eficiência. Esses algoritmos garantem exaustividade e precisão na descoberta de padrões, sendo altamente eficazes em bases de dados menores. No entanto, podem enfrentar desafios de escalabilidade quando aplicados a grandes volumes de dados, nos quais a quantidade de possíveis sequências cresce exponencialmente, impactando negativamente o desempenho computacional.

Por outro lado, metaheurísticas evolutivas oferecem uma abordagem promissora para a tarefa de MPS. Inspiradas em processos naturais e biológicos, técnicas como Algoritmos Genéticos (AG), Otimização por Colônia de Formigas (OCF), Otimização por Enxame de Partículas (OEP) e Algoritmo de Busca de Harmonia (ABH) apresentam uma forma robusta de explorar grandes espaços de busca. Essas metaheurísticas têm a capacidade de escapar de ótimos locais, proporcionando soluções quase ótimas em um tempo computacional aceitável, mesmo em cenários onde a aplicação de algoritmos determinísticos se torna inviável por causa do custo computacional.

Em trabalhos anteriores, investigou-se a aplicação isolada de algoritmos clássicos (Maranhão et al., 2023) e de metaheurísticas evolutivas (Maranhão & Neto, 2024) na tarefa de MPS. Nesse estudo, realiza-se um comparativo qualitativo entre ambas as abordagens, evidenciando que os algoritmos clássicos são mais adequados em cenários que exigem exaustividade, enquanto as metaheurísticas se destacam em problemas de maior escalabilidade e ampla exploração de soluções. Além de avaliar o desempenho de ambas as abordagens, discute-se a relevância dos padrões descobertos para a melhoria das estratégias de ensino e a personalização do aprendizado.

Por fim, optou-se por realizar um estudo de caso em um ambiente real de aprendizado, denominado Cosmo, desenvolvido e mantido pelo Laboratório de Sistemas Multimídia, da Universidade Federal do Maranhão (UFMA). Essa decisão reflete tanto o caráter exploratório da pesquisa, que busca revelar padrões de aprendizado pouco mapeados, quanto sua dimensão explicativa, ao investigar causas e implicações desses padrões na motivação e na aprendizagem. Ao focar na exploração desse ambiente, pode-se observar com mais profundidade a interação dos alunos, permitindo identificar possíveis relações entre as sequências de interação e possíveis dificuldades.

O presente artigo está organizado como segue: a Seção 2 apresenta a fundamentação teórica acerca da mineração de padrões sequenciais usando a abordagem clássica e as metaheurísticas investigadas; a Seção 3 apresenta os trabalhos que se relacionam a esta pesquisa; a Seção 4 descreve a metodologia de análise dos dados e modelagem do problema; a Seção 5 discute os resultados obtidos e compara as abordagens; e a Seção 6 apresenta a conclusão do trabalho.

2 Mineração de Padrões Sequenciais (MPS)

A MPS consiste em encontrar subsequências interessantes em um banco de dados de sequências (Agrawal & Srikant, 1995). O interesse de uma subsequência pode ser definido de várias maneiras, como sua frequência ou seu comprimento. Uma sequência é formada por um conjunto não vazio de itens i_j, formalmente representada como $\{i_1, i_2, \cdots, i_n\}$, em que i_j é chamado de *itemset*. No ensino de algoritmos, um item representa uma ação realizada por um aluno ao interagir com a plataforma, seja acessando um conteúdo teórico seja submetendo a solução de um determinado problema.

A Tabela 1 apresenta um conjunto de sequências para alunos hipotéticos. A sequência do aluno 1 possui duas interações. O primeiro *itemset* contém dois itens, *a* e *b*, indicando que o aluno interagiu com os objetos de aprendizagem *a* e *b* na sessão 1, enquanto a sessão 2 indica que o aluno interagiu com o problema *c*, depois, na sessão 3, com o problema *d* e, finalmente, na sessão 4 com o problema *e*. A ordem dos itens em um *itemset* não é relevante. Todas as sessões do aluno formam uma sequência.

Tabela 1: Exemplo de Banco de Sequências.

Id do Aluno	Sequência
1	$\langle\{a,b\},\{c\},\{d\},\{e\}\rangle$
2	$\langle\{a,c\},\{e,f\}\rangle$
3	$\langle\{b\},\{c,d,e\},\{f,b\},\{c\}\rangle$
4	$\langle\{b,c\},\{d\},\{f\},\{g\}\rangle$

Formalmente, uma sequência $\{i_1, i_2, \cdots, i_n\}$ contém uma subsequência $\{p_1, p_2, \cdots, p_m\}$ se houver números inteiros $k_1 < k_2 < \cdots < k_n$ tais que $p_1 \subseteq i_{k_1}, p_2 \subseteq i_{k_2}, p_m \subseteq i_{k_n}$. Por exemplo, para avaliar se as sequências da Tabela 1 contêm a subsequência $S = \langle\{b\},\{c\}\rangle$, verifica-se se a sequência possui algum item contendo $\{b\}$ e, posteriormente, outro contendo $\{c\}$, respeitando a ordem em que aparecem na sequência. Sendo assim, pode-se observar que:

- a sequência 1 contém *S*, porque, respectivamente, $\{b\}$ aparece no primeiro item e $\{c\}$ aparece no segundo item;
- a sequência 2 não contém *S*, pois não há itens contendo $\{b\}$;

- a sequência 3 contém S, pois $\{b\}$ aparece no primeiro item para e $\{c\}$ aparece tanto no segundo quanto no quarto itens, mantendo a ordem exigida;
- a sequência 4 não contém S, pois, apesar de $\{b\}$ e $\{c\}$ aparecerem juntos no primeiro item, não há segundo item apenas com $\{c\}$ para satisfazer a subsequência em dois passos.

Muitos algoritmos permitem definir o *gap* máximo para reduzir padrões ruidosos (padrões identificados como frequentes em virtude de erros aleatórios) e limitar a quantidade de padrões retornados (Srikant & Agrawal, 1996; Zaki, 2000). Por exemplo, se o intervalo máximo for 1, o algoritmo inferirá que a sequência do aluno 1 não contém $\langle\{b\},\{e\}\rangle$, pois $\{b\}$ aparece na primeira posição, em $\{a,b\}$, e $\{e\}$ está na quarta posição, resultando em um gap de $4-1=3$. Os ruídos devem ser removidos porque eles podem não fornecer informações confiáveis sobre os processos que geraram as sequências.

A sequência de um aluno suporta uma subsequência se ela estiver contida na sequência do aluno. O valor de suporte de uma subsequência é definido como a proporção de sequências que contém essa subsequência (Agrawal & Srikant, 1995). Ou seja, para obter o valor de suporte, deve-se contar em quantas sequências cada subsequência aparece e dividir pelo total de sequências analisadas. Se o valor de suporte de uma subsequência não for menor que um limite pré-especificado (chamado de suporte mínimo), essa subsequência é considerada um padrão sequencial frequente.

Se o suporte mínimo é definido como $sup\_min = 0,5$, então todas as sequências com suporte mínimo acima desse limiar serão consideradas frequentes. Tomando-se como exemplo as subsequências $S_1 = \langle\{b\},\{c\}\rangle$, $S_2 = \langle\{b\},\{d\}\rangle$, $S_3 = \langle\{c\},\{d\}\rangle$ e $S_4 = \langle\{a\},\{c\}\rangle$, observa-se que, pelo critério do suporte mínimo, S_1, S_2 e S_3 são padrões frequentes, enquanto S_4 não. Isso ocorrre porque o valor de suporte de:

- S_1 é $2/4 = 0,5 \geq sup\_min$, pois S_1 aparece em duas (alunos 1 e 3) das quatro possíveis sequências;
- S_2 é $3/4 = 0,75 \geq sup\_min$, pois S_2 aparece em três (alunos 1, 3 e 4) das quatro possíveis sequências;
- S_3 é $2/4 = 0,5 \geq sup\_min$, pois S_3 aparece em duas (alunos 1 e 4) das quatro possíveis sequências;
- S_4 é $1/4 = 0,25 < sup\_min$, pois S_4 aparece em uma (aluno 1) das quatro possíveis sequências.

Os algoritmos clássicos, como o PrefixSpan (Pei et al., 2001), GSP (*Generalized Sequential Patterns*) (Srikant & Agrawal, 1996) e SPADE (*Sequential PAttern Discovery using Equivalent Class*) (Zaki, 2000), utilizam abordagens determinísticas para explorar o espaço de busca de sequências frequentes. Para aumentar a eficiência da mineração, esses métodos frequentemente empregam técnicas de poda para reduzir a quantidade de padrões candidatos a serem avaliados.

Embora bem estabelecidos e amplamente utilizados por causa da sua capacidade de garantir exaustividade e precisão na descoberta de padrões sequenciais, os algoritmos clássicos enfrentam desafios de escalabilidade quando aplicados a bases de dados muito grandes ou complexas, onde a quantidade de possíveis sequências cresce exponencialmente (Zaki, 2001). É nesse cenário que a utilização de metaheurísticas pode ser promissora. Inspiradas em processos naturais e biológicos, elas têm sido aplicadas com sucesso para resolver problemas complexos de otimização.

2.1 Algoritmos Clássicos para Mineração de Padrões Sequenciais

Vários algoritmos têm sido aplicados a dados educacionais, como GSP (*Generalized Sequential Patterns*) (Srikant & Agrawal, 1996); SPADE (Zaki, 2000); FreeSpan (Han, Pei, Mortazavi-Asl et al., 2000), PrefixSpan (Pei et al., 2001); SPAM (*Sequential Pattern Mining*) (Ayres et al., 2002) e LAPIN (*Last Position Induction*) (Yang et al., 2007). Esses algoritmos podem ser divididos em duas abordagens: Geração de Candidatos e Crescimento de Padrões.

2.1.1 Geração de Candidatos

Essa abordagem se baseia na execução de múltiplos passos sobre os dados. Em cada passo, começa-se com um conjunto semente de uma quantidade grande de sequências, chamadas de sequências candidatas. O suporte para essas sequências candidatas é computado durante a passada pelos dados. Ao final do passo, são determinadas as maiores sequências candidatas. Tais sequências compõem a semente para o próximo passo.

Como exemplos, pode-se citar os algoritmos GSP e SPADE. O algoritmo GSP (*Generalized Sequential Patterns*) escala linearmente com a quantidade de sequências de dados, respeitando a quantidade de transações por sequência de dados e a quantidade de itens por transação (Srikant & Agrawal, 1996). O SPADE decompõe o problema original em subproblemas menores usando propriedades combinatoriais. Os subproblemas são resolvidos na memória principal utilizando técnicas de busca eficientes e operações de junção simples. Normalmente, são realizadas apenas três varreduras na base de dados – uma para sequências frequentes de tamanho 1, outra para sequências de tamanho 2 e mais uma para a geração das demais sequências (Chiu et al., 2004).

2.1.2 Crescimento de Padrões

Essa abordagem usa a estratégia de dividir para conquistar, começando com um conjunto de padrões frequentes de tamanho 1. Em seguida, deriva, para cada padrão p, um banco de dados projetado de p e o explora recursivamente. Como o conjunto de dados é decomposto progressivamente em um conjunto de bancos de dados muito menores, o método de crescimento de padrões reduz o espaço de pesquisa e leva a alta eficiência e escalabilidade.

A abordagem de geração de candidatos atinge um bom desempenho pela redução da quantidade de candidatos gerado. Entretanto, quando o suporte mínimo é pequeno ou o tamanho dos padrões gerados é grande, o algoritmo pode continuar tendo custos não triviais, independentemente das técnicas de implementação detalhadas (Han, Pei & Yin, 2000).

Para vencer essas dificuldades, foi desenvolvida uma nova abordagem denominada de crescimento de padrões frequentes. A ideia geral do FreeSpan (*Frequent pattern-projected Sequential pattern mining*) é usar itens frequentes para projetar recursivamente bancos de dados de sequências em um conjunto menor de bancos de dados e crescer os fragmentos de subsequências em cada banco de dados projetado (Han, Pei, Mortazavi-Asl et al., 2000).

Por outro lado, a concepção do PrefixSpan (*Prefix-projected Sequential Pattern Mining*) é de que, em vez de projetar sequências considerando todas as ocorrências possíveis de subsequências frequentes, a projeção seja baseada apenas em prefixos frequentes porque qualquer subsequência pode sempre ser encontrada a partir do crescimento de um prefixo (Pei et al., 2001).

2.2 Metaheurísticas para Mineração de Padrões Sequenciais

As metaheurísticas são estratégias de otimização de propósito geral especialmente úteis em situações em que métodos exatos de busca são inviáveis por causa da alta dimensionalidade ou da natureza complexa do espaço de busca. Elas não necessariamente vão encontrar a solução ótima, mas podem encontrar boas soluções em um tempo razoável. Outra vantagem das metaheurísticas é sua flexibilidade e adaptabilidade, permitindo ajustes finos e hibridizações para diferentes tipos de problemas (Parpinelli & Lopes, 2011).

2.2.1 Algoritmo Genético

O Algoritmo Genético (AG) foi proposto por J. H. Holland em 1975, inspirado na Teoria da Evolução de Darwin (Holland, 1992). Os conceitos evolutivos são usados para definir uma técnica de busca para resolver problemas de otimização. Nessa técnica, uma população de soluções candidatas (inicializada aleatoriamente) evolui por meio da seleção dos indivíduos mais aptos (melhores soluções candidatas), *crossover* e mutação.

Conforme mostra o Algoritmo 1, a metaheurística começa com um conjunto de indivíduos (população), onde cada indivíduo representa uma solução candidata para o problema. Cada indivíduo (ou cromossomo) corresponde a uma sequência de nós, $C = (n_1, n_2, n_3, \cdots, n_k)$, com tamanho fixo k, onde n_i é um nó do grafo e a sequência deve seguir arestas válidas entre os nós. A aptidão (*fitness*) de um cromossomo é determinada pela soma dos pesos das arestas entre os nós que compõem a sequência, penalizando caminhos inválidos com uma redução na pontuação.

Durante a seleção dos pais, um conjunto de indivíduos é escolhido para reprodução. Os métodos mais comuns de seleção incluem roleta, classificação e torneio. Os indivíduos selecionados são recombinados por meio de *crossover* para gerar novos indivíduos. Uma das abordagens mais simples é o cruzamento de pontos, onde um ou mais pontos de cruzamento são escolhidos aleatoriamente e o filho é criado pela troca dos genes entre os pais.

A mutação envolve pequenas modificações nas soluções candidatas geradas. Nessa etapa, um ou mais bits são alterados de acordo com uma probabilidade de mutação. A mutação é fundamental para manter a diversidade genética da população e evitar a convergência prematura para soluções subótimas.

Por fim, as novas soluções candidatas geradas pelo *crossover* e pela mutação são então avaliadas em relação à função objetivo, e a população atual é atualizada. Muitas vezes, os melhores indivíduos da população são mantidos, e até 10% das piores soluções candidatas podem ser preservadas para manter a diversidade. O algoritmo termina quando o critério de parada é alcançado.

2.2.2 Otimização por Enxame de Partículas

A Otimização por Enxame de Partículas (OEP), proposta por Russell Eberhart e James Kennedy, é inspirada nos comportamentos coletivos observados em bandos de pássaros e cardumes de peixe (Eberhart & Kennedy, 1995). Esse algoritmo utiliza informações tanto locais quanto globais para guiar o movimento e melhorar a qualidade das soluções candidatas representadas pelas partículas.

Algoritmo 1: Pseudocódigo do Algoritmo Genético.

```
Result: Sequência de materiais de aprendizagem
População ← InicializarPopulação(TP, |M|);
// TP: Tamanho da população
// M: Conjunto de materiais de aprendizagem, |M| é o
   número total de materiais
Objetivo ← CalcularObjetivo(População);
S_melhor ← ObterMelhorSolução(Objetivo);
while ¬CondiçãoDeParada() do
    Pais ← SelecionarPais(População, TT);
    // TT: Tamanho do torneio de seleção (número de
       indivíduos que competem para se tornar pais)
    Filhos ← ∅;
    for i ← 0 to |Pais| − 1 do
        (Filho_1, Filho_2) ← Cruzamento(Pais[i], Pais[i+1]);
        // Cruzamento: Combinação de dois pais para gerar
           dois filhos
        Filho_1 ← Mutação(Filho_1, PM);
        Filho_2 ← Mutação(Filho_2, PM);
        // PM: Probabilidade de mutação aplicada aos filhos
        Filhos ← Filhos ∪ {Filho_1, Filho_2};
        i ← i + 2;
    Objetivo ← CalcularObjetivo(Filhos);
    S_melhor ← ObterMelhorSolução(Objetivo);
    População ← Substituir(População, Filhos);
    // Substituir a população antiga pela nova geração de
       filhos
```

Conforme mostra o Algoritmo 2, cada partícula no OEP é caracterizada por três componentes principais: velocidade, um vetor de valores contínuos que indica a direção e a magnitude do movimento da partícula; posição, um vetor de valores contínuos que representa a solução candidata atual; e melhor local (P_{best}), a melhor posição encontrada pela partícula até o momento. A metaheurística também armazena o Melhor Global (G_{best}), que é a melhor posição encontrada por qualquer partícula no enxame.

Cada partícula representa uma solução candidata modelada como um caminho no grafo. Assim como nos outros algoritmos, a solução é expressa como uma sequência de nós, $C = (n_1, n_2, n_3, \cdots, n_k)$, com tamanho fixo k, onde n_i é um nó do grafo e a sequência deve respeitar conexões válidas entre os nós. A aptidão (*fitness*) de uma partícula é determinada pela soma dos pesos das arestas no caminho percorrido, penalizando soluções inválidas.

O movimento das partículas é influenciado por suas experiências individuais (P_{best}) e pela experiência coletiva do enxame (G_{best}). A cada iteração, as velocidades das partículas são atualizadas considerando três componentes: a velocidade anterior de cada partícula, a diferença entre a posição atual de cada partícula e sua melhor posição individual (P_{best}), e a diferença entre a posição atual

de cada partícula e a melhor posição global (G_{best}) do enxame. Além disso, são aplicados fatores de inércia e coeficientes que definem a influência das experiências individual e coletiva.

O processo de atualização continua até que um critério de parada seja atingido, que pode ser uma quantidade máxima de iterações ou a convergência para uma solução aceitável.

Algoritmo 2: Pseudocódigo do Algoritmo de Otimização por Enxame de Partículas.

```
Result: Sequência de materiais de aprendizagem
População ← InicializarPartículas(TP, |M|);
// TP: Tamanho da População
// M: Conjunto de materiais de aprendizagem, |M| é o
   número total de materiais
while ¬CondiçãoDeParada() do
    foreach x ∈ População do
        v_i(x) ← AtualizarVelocidade(v_i(x), pb(x), gb, c1, c2, c3);
        // Atualizar a velocidade da partícula com base em
           parâmetros e melhores posições locais e globais
        p_i(x) ← Avaliar(v_i(x), EM);
        // EM: Espaço de Materiais; calcular a nova posição
           da partícula com base na velocidade
        pb(x) ← Melhor(pb(x), p_i(x));
        // Atualizar a melhor posição local da partícula
        gb ← Melhor(gb, pb(x));
        // Atualizar a melhor posição global
```

2.2.3 Otimização por Colônia de Formigas

A Otimização por Colônia de Formigas (OCF) foi proposta por Marco Dorigo em 1992, inspirada no comportamento de busca de alimentos das formigas (Dorigo, 1992). As formigas depositam feromônio ao longo de seu caminho, criando trilhas que outras formigas seguem, com a intensidade do feromônio influenciando a probabilidade de escolha das trilhas.

No OCF (Algoritmo 3), uma população de formigas é utilizada para construir soluções para um problema de otimização. Cada formiga constrói uma solução ao adicionar componentes à solução parcial com base na quantidade de feromônio e em uma medida heurística da qualidade do componente. O algoritmo inicia com a definição dos parâmetros, como a quantidade de formigas, o coeficiente de evaporação do feromônio (ρ), e os parâmetros de controle (α e β). Além disso, a quantidade de feromônio em todas as arestas é inicializada com um valor inicial τ_0.

Cada solução construída por uma formiga corresponde a um caminho no grafo, representado como uma sequência de nós de tamanho fixo k, assim como no AG. Dessa forma, a estrutura da solução pode ser descrita como $C = (n_1, n_2, n_3, \cdots, n_k)$, onde n_i representa um nó do grafo, e a sequência deve seguir conexões válidas entre os nós. A aptidão da solução (*fitness*) é determinada pela soma dos pesos das arestas percorridas, sendo que caminhos inválidos são penalizados.

Enquanto constroem suas soluções, as formigas escolhem os componentes de maneira

probabilística, considerando tanto a quantidade de feromônio presente nas arestas (τ_{ij}) quanto a informação heurística associada a elas (η_{ij}). A probabilidade P_{ij}^k de uma formiga k escolher um determinado componente i após outro componente j é calculada com base em uma combinação desses fatores, ajustada pelos parâmetros de controle.

Depois que todas as formigas completam suas soluções, a quantidade de feromônio nas arestas é atualizada para refletir a qualidade das soluções encontradas. Esse processo de atualização leva em conta a evaporação natural do feromônio e a quantidade depositada por cada formiga, reforçando as arestas que fazem parte de soluções melhores. A evaporação é essencial para evitar a convergência prematura, garantindo que a exploração continue ao longo das iterações.

O processo de construção de soluções e atualização de feromônio é repetido até que um critério de parada seja atingido, como uma quantidade máxima de iterações ou a convergência para uma solução aceitável.

Algoritmo 3: Pseudocódigo do Algoritmo de Otimização por Colônia de Formigas.

$\alpha, \beta, \rho, \tau_0 \leftarrow$ InicializarParâmetros;
$\tau_{ij} \leftarrow \tau_0, \forall ij$;
// Inicializar todas as arestas com feromônio τ_0
while $\neg$*CondiçãoDeParada()* **do**
  foreach *formiga* $k \in$ *colônia* **do**
    $S_k \leftarrow$ ConstruirSolução(P_{ij}^k);
    // Construir uma solução usando a regra de probabilidade P_{ij}^k
    $P_{ij}^k \leftarrow \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in N_i} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta}$
    // P_{ij}^k: probabilidade de a formiga k escolher o componente j após i
  foreach *aresta* i, j **do**
    $\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k$;
    // Atualizar o feromônio na aresta ij com evaporação e depósito
return MelhorSolução();

2.2.4 Algoritmo de Busca de Harmonia

O Algoritmo de Busca de Harmonia (ABH) é uma heurística de busca baseada no processo de improvisação dos músicos de jazz (Geem et al., 2001). No jazz, os músicos tentam ajustar os tons de seus instrumentos, de forma que as harmonias gerais sejam otimizadas em virtude dos aspectos estéticos. Começando com algumas harmonias, eles tentam alcançar melhores harmonias utilizando a improvisação.

Essa analogia pode ser usada para derivar heurísticas de busca, que podem otimizar uma determinada função objetivo em vez de harmonias. Nesse contexto, os músicos são identificados com as variáveis de decisão e as harmonias correspondem às soluções. Tal como os músicos de

jazz criam novas harmonias por meio da improvisação, o algoritmo ABH cria iterativamente novas soluções baseadas em soluções passadas e em modificações aleatórias.

Conforme mostra o Algoritmo 4, a metaheurística ABH inicializa a *Harmony Memory* (HM) com soluções geradas aleatoriamente. A quantidade de soluções armazenadas na memória HM é definido pelo *Harmony Memory Size* (HMS). Então, iterativamente, uma nova solução é criada como segue. Cada variável de decisão é gerada considerando a memória e uma possível modificação adicional ou por seleção aleatória.

Os parâmetros utilizados no processo de geração de uma nova solução são chamados *Harmony Memory Considering Rate* ($HMCR$) e *Pitch Adjusting Rate* (PAR). Cada variável de decisão é definida como o valor da variável correspondente de uma das soluções na HM com probabilidade de HMCR, e uma modificação adicional desse valor é realizada com probabilidade de PAR.

Caso contrário, com probabilidade de $1 - HMCR$, a variável de decisão é definida com um valor aleatório. Após a criação de uma nova solução, ela é avaliada e comparada com a pior solução da memória. Se o seu valor objetivo for melhor que o da pior solução, substitui a pior solução no HM. Esse processo é repetido até que um critério de parada seja satisfeito.

Algoritmo 4: Pseudocódigo do Algoritmo de Busca de Harmonia.

```
1  iterações ← 0;
2  for i = 0; i < HMS; i++ do
3      HM[i] = solução viável gerada estocasticamente;
4  Ordenar HM;
5  while iterações < IT do
6      for i = 0; i < n; i++ do
7          Escolher aleatoriamente r ∈ (0,1);
8          if r < HMCR then
9              H[i] = escolher aleatoriamente tom disponível na posição i em HM;
10             Escolher aleatoriamente k ∈ (0,1);
11             if k < PAR then
12                 α = bw aleatório ∈ (−1,1);
13                 H[i] = H[i] + α;
14         else
15             H[i] = escolher aleatoriamente tom disponível ;
16     if f(H) é melhor que f(HM[HMS − 1]) then
17         HM[HMS − 1] = H;
18         Ordenar HM;
19     iterações++;
20 return HM[0]
```

3 Trabalhos Relacionados

Para compor esta seção de trabalhos relacionados, foi realizado um mapeamento sistemático da literatura com o objetivo de identificar pesquisas que aplicassem metaheurísticas evolutivas para o sequenciamento de currículos e personalização de aprendizado em contextos educacionais, especialmente no ensino de algoritmos. As questões secundárias exploraram quais características dos alunos e dos materiais didáticos são consideradas, quais técnicas de computação evolutiva são aplicadas e como o sequenciamento de aprendizado foi realizado.

A estratégia de busca incluiu a construção de uma *string* baseada em termos como "*curriculum sequencing*", "*learning path generation*", "*personalized course generation*" e "*instructional planning*", usando algoritmos clássicos de MPS ou metaheurísticas evolutivas. As fontes de pesquisa selecionadas foram Google Scholar, IEEE, Scopus, Springer e ACM, abrangendo publicações de 2011 a 2023. Além disso, foram utilizadas as técnicas de "*backward snowballing*" e "*forward snowballing*" para expandir a rede de trabalhos relevantes.

Os artigos selecionados estão relacionados ao presente estudo por explorarem a aplicação dos algoritmos clássicos de MPS ou usarem metaheurísticas para analisar e melhorar a experiência de aprendizagem dos alunos. Embora existam trabalhos que aplicam os algoritmos clássicos e as metaheurísticas separadamente no contexto educacional, não foram encontrados trabalhos que façam um paralelo entre as duas abordagens.

A MPS tem sido empregada em pesquisas com propósitos variados na área educacional, incluindo a descoberta de comportamentos de aprendizagem, o enriquecimento de teorias educacionais e a filtragem de recursos didáticos para a construção de sistemas de recomendação. Inserido nesse contexto, o presente trabalho busca contribuir com o estado da arte ao comparar o uso dos algoritmos clássicos com metaheurísticas para a tarefa de mineração de padrões sequenciais, especificamente no contexto do ensino de algoritmos.

Os dados do processo de aprendizagem, como *logs* de eventos, registram informações detalhadas sobre as interações dos alunos com os ambientes de aprendizagem, colegas e instrutores. Nesse contexto, padrões sequenciais frequentes podem indicar comportamentos comuns entre os alunos (Zhou et al., 2010). Esses padrões de comportamento podem revelar como os alunos navegam em suas atividades dentro de um ambiente de aprendizagem e indicar como melhorar a experiência de aprendizagem (Mirzaei & Sahebi, 2019).

Em Kang et al. (2017), os autores aplicaram o algoritmo cSPADE aos *logs* de *Alien Rescue*, um jogo sério para ensinar habilidades científicas de resolução de problemas a alunos do ensino médio. O estudo teve como objetivo analisar como os padrões sequenciais de interação podem diferir ao longo de vários dias de uso do jogo educacional. Eles observaram como os padrões sequenciais nos primeiros dias representavam comportamentos de exploração, enquanto os padrões sequenciais nos dias restantes indicavam comportamentos científicos de resolução de problemas.

Alguns estudos mostram como a MPS pode ser utilizada para investigar trajetórias de grupos com desempenhos acadêmicos diferentes para tentar identificar padrões frequentes no grupo de alto desempenho, mas raros no grupo de baixo desempenho. Por exemplo, Slim et al. (2016) aplicaram a MPS em sequências de matrículas em cursos de graduação em engenharia elétrica. O resultado mostrou que os alunos que se formaram com uma média alta seguiram um padrão de matrícula distinto daqueles que obtiveram uma média mais baixa.

Por sua vez, as metaheurísticas AG, OCF, OEP e ABH têm sido aplicadas no contexto educacional para personalizar e otimizar a experiência de aprendizagem dos estudantes. Os estudos apresentados a seguir propõem abordagens para enfrentar os desafios de sobrecarga cognitiva e desorientação dos alunos, proporcionando aprendizado mais eficaz e personalizado.

Em Bhaskar et al. (2010), é descrito um sistema que utiliza AGs para criar esquemas de aprendizado adaptativos baseados no contexto específico de cada aluno. O método considera múltiplos parâmetros, como perfil, preferências e infraestrutura, para gerar conteúdos de ensino personalizados. A eficácia foi demonstrada com um curso de redes de computadores, onde o algoritmo gerou esquemas de aprendizagem adaptados às necessidades e preferências individuais dos alunos.

Em Sharma et al. (2012), é proposto um algoritmo baseado na metaheurística OCF que avalia o nível de um aprendiz e recomenda os conceitos apropriados para ele. Esse algoritmo é sensível às mudanças nos comportamentos de aprendizagem de cada aluno e ajusta suas estratégias para recomendar o próximo conceito de acordo com a necessidade. Os comportamentos dos alunos anteriores são capturados e utilizados para recomendar conteúdo a futuros alunos.

Em Li et al. (2012), é detalhado o uso de metaheurísticas para compor cursos personalizados, atendendo às necessidades individuais dos alunos. O processo utiliza as metaheurísticas AG e OEP na etapa de composição do curso personalizado. Os experimentos realizados mostraram que, com até 300 materiais de aprendizagem, o OEP se destaca pela eficiência em termos de tempo e quantidade de gerações necessárias para convergir para uma solução ótima. Com mais de 300 materiais, o AG foi mais eficiente.

Em Hnida et al. (2016), o algoritmo ABH foi sugerido como abordagem ao problema de sequenciamento curricular. Esse algoritmo é inspirado no processo de improvisação musical, em que um grupo de músicos improvisa o tom de seus instrumentos, buscando a harmonia perfeita. A pesquisa adapta o ABH para sequenciar objetos de aprendizagem de maneira a maximizar a relevância dos conteúdos apresentados aos alunos, considerando o nível de conhecimento dos alunos e as inter-relações de conteúdos.

Em Machado et al. (2019), o Algoritmo Presa-Predador (APP) é empregado na geração sequências curriculares adaptativas considerando características extrínsecas e intrínsecas dos alunos. Os experimentos foram conduzidos em um ambiente real de aprendizado, demonstrando que a abordagem personalizada melhora a qualidade da compreensão dos alunos e reduz a taxa de desistência, sugerindo um efeito motivacional.

Em Martins et al. (2021), são comparadas diferentes metaheurísticas, incluindo AG, OEP, APP e Evolução Diferencial (ED), aplicadas à geração de sequências curriculares personalizadas. Além de propor um procedimento para criar conjuntos de dados sintéticos para avaliação das abordagens, os experimentos indicaram que o algoritmo ED é mais eficaz em problemas menores (até 500 materiais didáticos), enquanto OEP se destacou em instâncias maiores.

Em Almeida et al. (2022), é definido um modelo de sequenciamento e recomendação de ações pedagógicas personalizadas em Ambientes Virtuais de Aprendizagem (AVA) que utiliza a Taxonomia de Bloom para modelar as ações pedagógicas e o método RASI para definir o perfil cognitivo do estudante. O estudo utiliza um AG multiobjetivo para recomendar ações pedagógicas alinhadas ao perfil do estudante, otimizando o processo de ensino.

Dessa forma, a presente pesquisa busca contribuir para o estado da arte ao comparar o uso de algoritmos clássicos de MPS com metaheurísticas para o sequenciamento de aprendizado, particularmente no contexto do ensino de algoritmos. A expectativa é que, após entender as limitações de cada abordagem, o uso combinado delas possibilite a criação de um modelo mais robusto e flexível, capaz de lidar com desafios como sobrecarga cognitiva e recomendação de conteúdo didático.

4 Metodologia

Esta seção descreve o processo de análise de dados adotado no presente estudo. A Seção 4.1 detalha o processo de obtenção, tratamento e visualização dos dados, proporcionando entendimento mais claro ao leitor. A Seção 4.2 apresenta e discute os resultados obtidos por meio de duas abordagens para a descoberta de padrões sequenciais: uma usando os algoritmos clássicos da MPS e outra usando metaheurísticas evolutivas.

4.1 Obtenção, Tratamento e Visualização dos Dados do Ambiente de Aprendizagem

Esta etapa consiste em entender um pouco da dinâmica do ambiente de aprendizagem de programação onde foram produzidos os dados em análise. A plataforma de ensino em questão é utilizada na disciplina de Algoritmos I do Curso de Ciência da Computação da Universidade Federal do Maranhão.

A disciplina é organizada para ser realizada em um semestre e abrange os seguintes tópicos:

- variáveis e atribuição: conceitos de armazenar e manipular dados;
- comandos condicionais: uso de estruturas como “se” e “senão” para tomada de decisões no código;
- laços de repetição: execução repetitiva de instruções com estruturas como “para”, “enquanto"e “faça ... enquanto”; e
- vetores e listas: manipulação de coleções de dados para armazenar múltiplos valores.

Entre 2021 e 2023, período em que os dados foram coletados, a plataforma disponibilizou materiais teóricos e atividades práticas acerca desses quatro tópicos. Durante esse tempo, os alunos tiveram liberdade para explorar os conteúdos sem suporte adicional à navegação, e as aulas foram ministradas sempre pelo mesmo professor.

A Figura 1 exibe a tela “Minhas Atividades”, que serve como ponto de partida para os alunos acessarem os conteúdos teóricos e atividades práticas disponibilizados na plataforma. Também, são exibidas as telas relativas aos conteúdos teóricos dos tópicos “Variáveis e Atribuição” (Figura 2) e “Comando Condicionais” (Figura 3).

A plataforma funciona internamente como um sistema de juiz *online*, sendo capaz de:

- compilar os códigos-fonte submetidos pelos alunos;
- executar esses códigos contra casos de teste previamente definidos; e
- avaliar o resultado das submissões, classificando-as em:

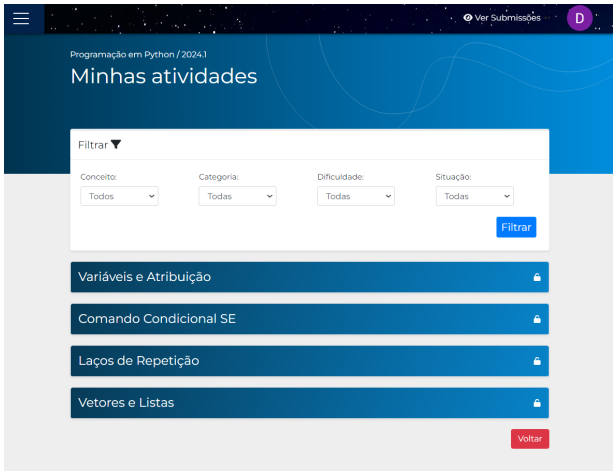

Figura 1: Tópicos disponibilizados na plataforma.

Figura 2: Conteúdo teórico do tópico “variáveis e atribuição”.

Figura 3: Conteúdo teórico do tópico “comandos condicionais”.

- sucesso: se nenhum caso de teste falhar;
- erro de compilação: se houver qualquer falha no processo de compilação, como erros de sintaxe;
- tempo limite excedido: se o código não gerar os resultados esperados no tempo previsto; e
- erro de execução: se algum caso de teste falhar durante a execução.

A Figura 4 ilustra a escolha da questão “Olá, Mundo!”. Nessa questão, como mostra a Figura 5, o aluno deve apenas imprimir a expressão que a intitula. Havendo sucesso na submissão (Figura 6) ou se identificados erros de execução (Figura 7) ou de compilação (Figura 8), a plataforma fornece *feedback* ao aluno no momento da submissão, facilitando a compreensão dos conceitos e o aperfeiçoamento das habilidades de programação dos estudantes.

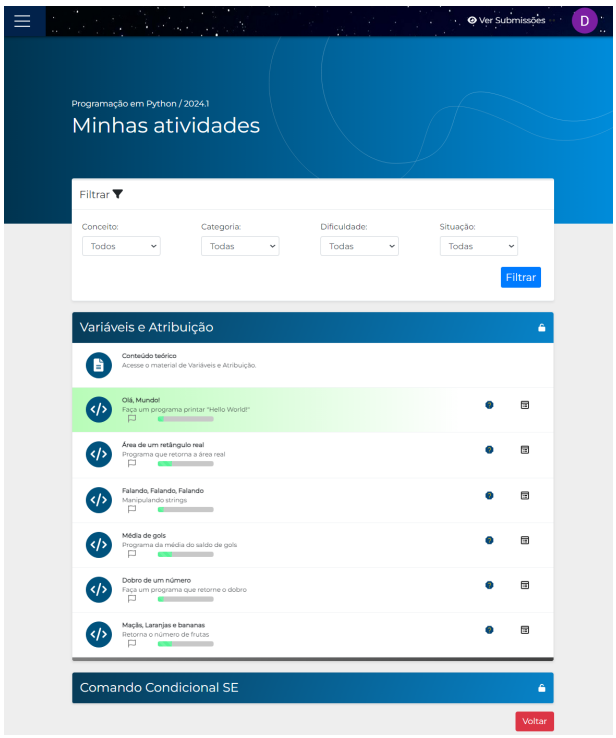

Figura 4: Tela “Minhas Atividades".

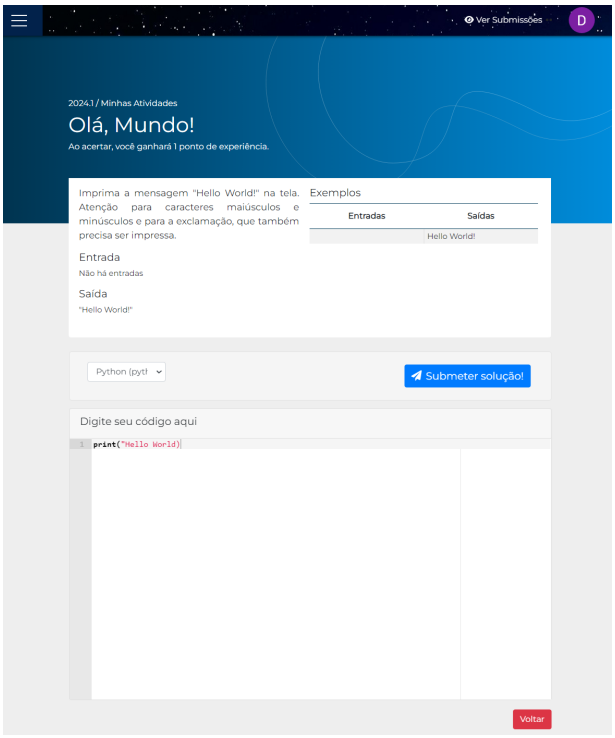

Figura 5: Tela exibindo a questão “Olá, Mundo!".

4.1.1 Obtenção dos Dados

A coleta foi realizada mediante consulta direta à base de dados da plataforma de ensino. O conjunto de dados foi obtido a partir do cruzamento entre as tabelas de questões e submissões, sendo composto pelas seguintes colunas: *usuario_id*, *questao_id*, *conceito_id*, *turma_id*, *tipo_resultado_id*, *tipo_linguagem_id*, *codigo*, *data_criacao*, *tempo_execucao* e *resultado*. A Tabela 2 exemplifica algumas das submissões contidas no conjunto de dados.

Tabela 2: Exemplos de Submissões Contidas no Conjunto de Dados

ID	Usuário ID	Questão ID	Turma ID	Tipo Resultado ID	Tipo Linguagem ID	Código	Tempo de Execução	Data de Criação	Resultado
121	36	3	1	1	5	print("Hello World!")	0,037	2021-03-03	Hello World!
122	19	3	1	2	5	print("Hello World!")	0,037	2021-03-03	Hello World!
124	36	3	1	1	5	print("Hello World!")	0,032	2021-03-03	Hello World!
125	37	6	1	1	5	base = float(input()) altura = float(input()) area = base * altura print(":.1f".format(area))	0,147	03/03/2021	9.5 9.9 11.1 2.0
126	29	3	1	1	5	print("Hello World!")	0,035	03/03/2021	Hello World!
127	13	3	1	1	5	base = float(input()) altura = float(input()) area = base * altura print(f"area:.2f")	NaN	03/03/2021	File "codigo.py", line 5 print(f"area:.2f") SyntaxError: invalid syntax

128	37	3	1	1	5	num_1 = int(input()) print(2*num_1)	0,169	03/03/2021	4 -6 2 8 -10
129	36	3	1	2	5	print("Hello World!")	0,032	03/03/2021	Hello World!
130	27	4	1	2	5	a = input("Digite um numero:") print(a*2)	0,181	03/03/2021	Digite um numero:22 Digite um numero:-3.3 Digite um numero:11 Digite um numero:44 Digite um numero:-5.5
131	16	3	1	1	5	print("Hello World!")	0,031	03/03/2021	Hello World!

O conjunto de dados brutos contém o total de 17.945 (dezessete mil, novecentas e quarenta e cinco) submissões realizadas por 329 alunos de 7 turmas (semestres) diferentes, todas escritas utilizando a linguagem de programação Python. Além disso, estão presentes o total de 68 (sessenta e oito) problemas, distribuídos em 4 (quatro) tópicos distintos.

4.1.2 Limpeza e Transformação dos Dados

A limpeza envolveu a remoção dos alunos com quantidades de interações muito discrepantes (*outliers*), identificados por meio do método do intervalo interquartílico [1]. Antes da limpeza, o primeiro quartil (Q_1) da distribuição de interações por aluno era de 21, enquanto o terceiro quartil (Q_3) era de 72, resultando em um $IQR = Q_3 - Q_1 = 72 - 21 = 51$. Foram considerados *outliers* quaisquer observações menores que $Q_1 - 1,5 \times IQR = 21 - 1,5 \times 51 = -55,5$ ou maiores que $Q_3 + 1,5 \times IQR = 72 + 1,5 \times 51 = 148,5$. Como resultado desse processo, 16 usuários e 4.029 submissões foram excluídos da análise.

Em seguida, os registros de submissão foram convertidos de um formato relacional para uma estrutura de lista encadeada, o que envolveu agrupar as submissões por usuário e, em seguida, ordenar cada grupo pela data e hora das interações, criando sequências temporais das atividades. O objetivo dessa transformação é simplificar a identificação de padrões de comportamento e outros fatores relevantes ao desempenho dos alunos, conforme apresentado nas seções seguintes.

4.1.3 Análise Exploratória e Visualização dos Dados

A etapa de análise exploratória e visualização tem como principal objetivo entender a estrutura dos dados, buscando identificar padrões e detectar anomalias, assim como resumir as principais características dos dados com a ajuda de métodos visuais. Após a etapa de limpeza, o conjunto de dados possui o total de 13.916 submissões realizadas por 313 alunos de 7 turmas (semestres) diferentes, todas escritas em Python.

Estão presentes no conjunto de dados o total de 68 (sessenta e oito) problemas, dos quais 7 são sobre variáveis e atribuição; 12 sobre comandos condicionais; 29 sobre laços de repetição; e 20 sobre vetores e listas, conforme apresentado na Tabela 3. As questões possuem níveis de dificuldade variados, partindo de um simples "Olá, Mundo!" até questões mais avançadas, como o cálculo dos números primos em um determinado intervalo.

[1] O intervalo interquartílico (*IQR - InterQuartile Range*) é a diferença entre o 3º Quartil (Q_3) e o 1º Quartil (Q_1), representando os 50% centrais dos dados.

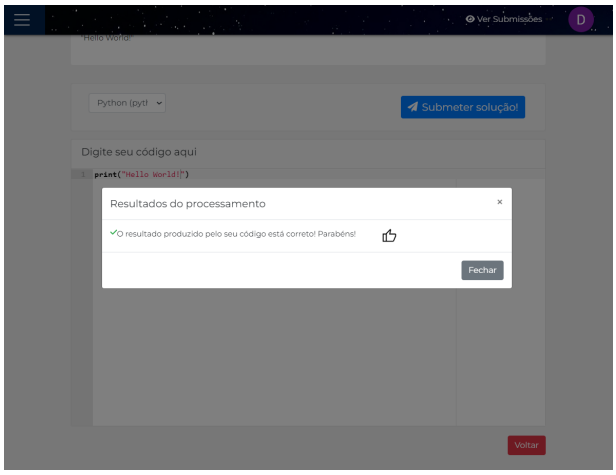

Figura 6: Tela com resultados do processamento, exibindo mensagem de “sucesso”.

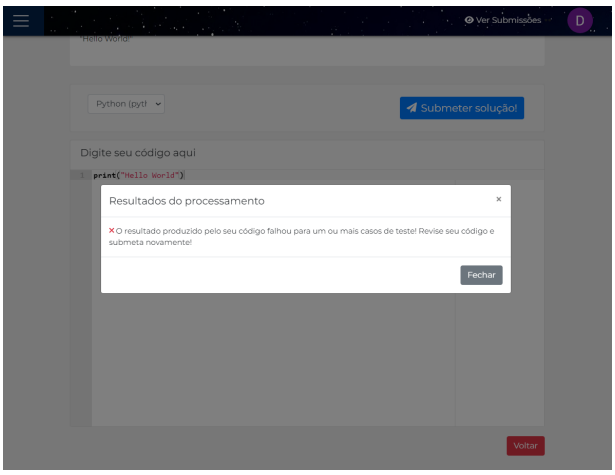

Figura 7: Tela com resultados do processamento, exibindo mensagem de “erro de execução”.

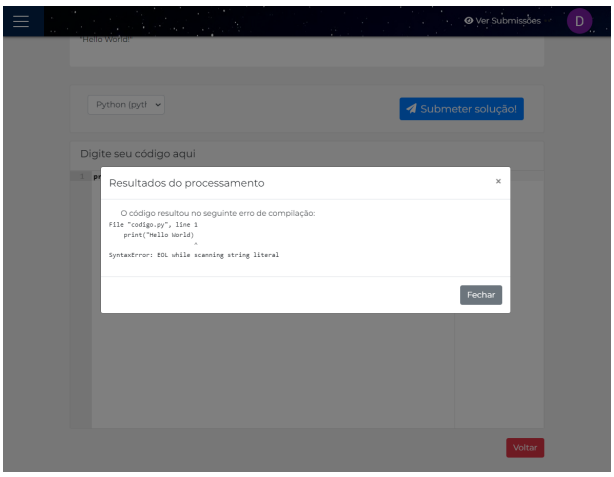

Figura 8: Tela com resultados do processamento, exibindo mensagem de “erro de compilação”.

Tabela 3: Distribuição das questões por tópico.

Tópico	**Identificadores**
Variáveis e Atribuição	3, 4, 6, 7, 8, 12, 210
Comandos Condicionais	5, 10, 11, 13, 14, 15, 18, 19, 30, 94, 106, 216
Laços de Repetição	16, 17, 26, 31, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 236, 238, 240, 242, 244, 246, 248, 250
Vetores e Listas	25, 27, 29, 56, 57, 58, 59, 60, 65, 67, 69, 71, 72, 73, 74, 75, 76, 254, 256, 262

Nesse momento, é pertinente apresentar exemplos de algumas questões analisadas. Isso proporcionará uma visão mais clara dos tipos de problemas que os alunos enfrentaram e facilitará a compreensão das análises subsequentes. A seguir, na Tabela 4, são apresentados os enunciados de sete questões selecionadas do conjunto de dados.

Ainda, podem ser encontradas no conjunto de dados cerca de 6.557 submissões bem-sucedidas; 5.155 submissões com erros de compilação; 2.140 submissões resultaram em erros de tempo de execução; e 64 submissões que excederam o limite permitido. Os tipos de resultado estão

distribuídos pelas questões conforme representado na Figura 9.

Conforme mostra a Figura 10, os dados revelam significativa desigualdade na distribuição de submissões: 80% delas estão concentradas em apenas 19 questões, que representam cerca de 27,9% dos problemas disponíveis na plataforma. Além disso, cerca de 50% de todas as submissões estão concentradas em apenas 9 questões. Esse desequilíbrio pode ser parcialmente explicado pelo cadastro contínuo de novos problemas a cada semestre, o que justifica a menor quantidade de submissões em questões mais recentes.

Tabela 4: Enunciados das Questões

Id	Enunciado	Tópico
3	Imprima a mensagem "Hello World!" na tela. Atenção para caracteres maiúsculos e minúsculos e para a exclamação, que também precisa ser impressa.	Variáveis e Atribuição
4	Escreva um programa que pede para o usuário digitar um inteiro. O programa deve exibir como resultado o dobro desse número inteiro.	Variáveis e Atribuição
5	Escreva um programa que pede para o usuário informar a medida da base e da altura de um retângulo (ambos inteiros). Depois, o programa imprime em uma linha isolada o valor da área desse retângulo.	Comandos Condicionais
6	Faça um programa que pede para o usuário informar a medida da base e da altura de um retângulo (ambos reais, com uma casa decimal). Depois, o programa imprime em uma linha isolada o valor da área desse retângulo, com uma casa decimal de precisão.	Variáveis e Atribuição
7	Crie um programa que receba dois números inteiros do usuário: o primeiro é o número de gols feitos por uma determinada equipe no campeonato e o segundo é o número de gols sofridos. Visto que o saldo de gols é a diferença entre o número de gols feitos e gols sofridos, e que o campeonato tem 38 jogos, calcule a média do saldo de gols de cada equipe por jogo.	Variáveis e Atribuição
11	Faça um programa em que o usuário informa dois inteiros, que chamaremos de SOMA e SUBTRAÇÃO. A SOMA é a soma de maçãs e bananas que o usuário comeu durante a semana. Já SUBTRAÇÃO corresponde à diferença entre a quantidade de bananas e de maçãs que aquele usuário comeu durante a semana. Dados os dois inteiros SOMA e SUBTRAÇÃO, seu programa deve exibir a quantidade de maçãs que o usuário comeu durante a semana.	Comandos Condicionais
12	Faça um programa em que o usuário informa três inteiros em três linhas isoladas. O primeiro inteiro corresponde à soma de maçãs e laranjas que ele comeu durante a semana. Já o segundo inteiro corresponde à soma de maçãs e bananas que ele comeu. Por fim, o terceiro inteiro corresponde à soma de laranjas e bananas que ele comeu. Dadas as três somas, imprima em três linhas isoladamente a quantidade de maçãs (primeira linha), laranjas (segunda linha) e bananas (terceira linha) que o usuário comeu durante aquela semana.	Variáveis e Atribuição

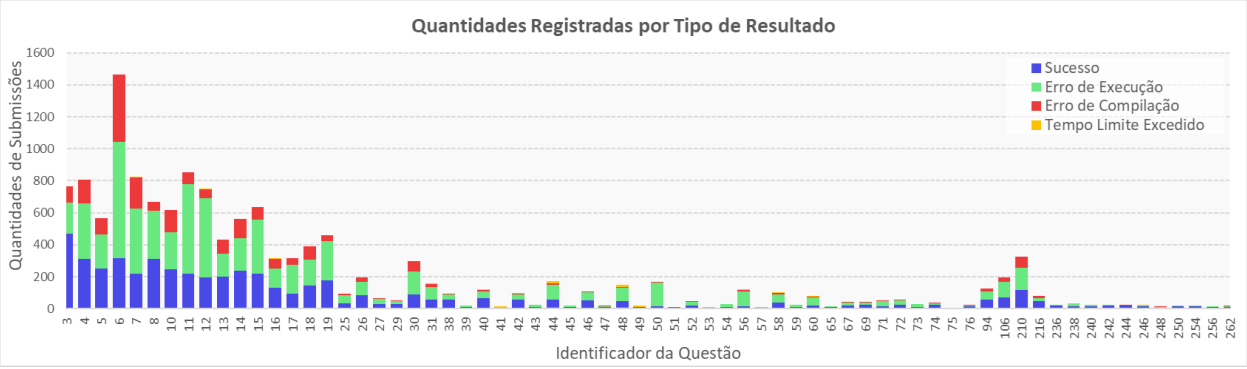

Figura 9: Distribuição dos tipos de resultado por questão.

Outro aspecto importante é a distribuição relativa das submissões por tópico e tipo de resultado. Conforme ilustra a Figura 11, os dados indicam que 25% das submissões estão concentradas no tópico de Variáveis e Atribuição, 24% em Comandos Condicionais, 33% em Laços de Repetição e 18% em Vetores e Listas. Quanto aos tipos de resultado, a Figura 12 mostra que 37% das

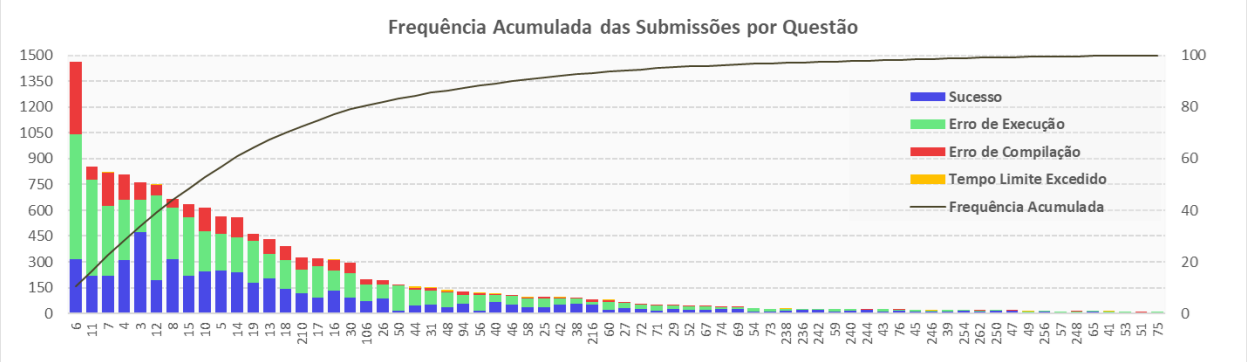

Figura 10: Distribuição dos tipos de resultado por questão.

submissões foram bem-sucedidas, enquanto 47% resultaram em Erro de Execução, 15% em Erro de Compilação e apenas 1% em Tempo Limite Excedido.

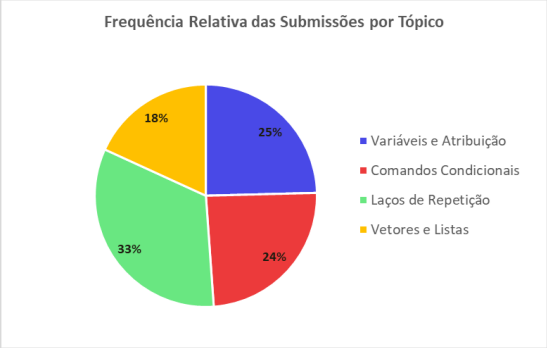

Figura 11: Distribuição Relativa das Submissões por Tópico.

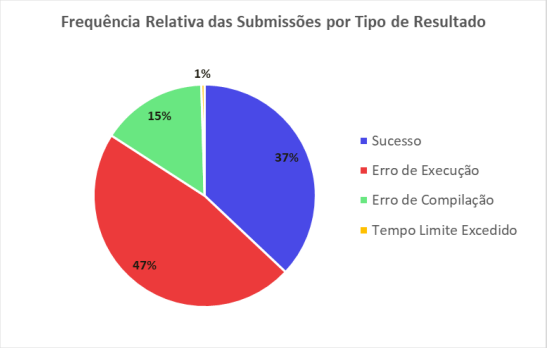

Figura 12: Distribuição Relativa das Submissões por Tipo de Resultado.

Por fim, a Tabela 5 mostra que, em média, cada aluno teve 44,46 interações, mas com uma grande variação, refletida no desvio padrão de 32,95. O aluno menos participativo teve apenas uma interação, enquanto o mais ativo teve 184. Além disso, a Tabela 6 mostra que cada questão recebeu em média 400,69 interações, com um desvio padrão ainda maior, de 404,27. A questão menos popular teve 16 submissões, enquanto a mais popular teve 1.801. Esses dados demonstram grande diversidade tanto na participação dos alunos quanto no interesse pelas questões.

4.2 Modelagem do Problema

A abordagem adotada neste trabalho é estruturada em torno de um modelo de grafo, onde cada nó representa uma questão específica com a qual os alunos interagiram na plataforma de ensino. Esse grafo é enriquecido com informações detalhadas sobre a interação dos alunos, incluindo a quantidade de tentativas feitas para resolver cada questão, proporcionando uma visão dos desafios enfrentados ao longo das trajetórias de aprendizado. As arestas entre os nós, por sua vez, representam as transições de uma questão para outra, refletindo as sequências de navegação e interação entre tópicos.

Os experimentos são organizados em duas abordagens distintas para facilitar a análise

Medida-resumo	Valor
Média	44,46
Desvio Padrão	32,95
Mínimo	1
1o. Quartil (25%)	20
2o. Quartil (50%)	33
3o. Quartil (75%)	63
Máximo	147

Tabela 5: Medidas-resumo da distribuição de interações por usuário.

Medida-resumo	Valor
Média	204,64
Desvio Padrão	289,16
Mínimo	4
1o. Quartil (25%)	23
2o. Quartil (50%)	70
3o. Quartil (75%)	299,5
Máximo	1464

Tabela 6: Medidas-resumo da distribuição de interações por questão.

comparativa: algoritmos clássicos de mineração de padrões sequenciais e metaheurísticas evolutivas. Na primeira abordagem, aplicam-se algoritmos clássicos para identificar padrões sequenciais de aprendizagem com base nas trajetórias mais recorrentes. Na segunda, explora-se o uso de quatro metaheurísticas evolutivas, especialmente vantajosas na detecção de padrões complexos que os algoritmos clássicos podem não captar em razão de sua natureza determinística e menos flexível.

Para avaliar a eficácia da modelagem proposta, os experimentos foram conduzidos em uma máquina com as seguintes especificações: processador Intel Core i7-8550U de 1.80 GHz (turbo até 1.99 GHz), 16 GB de memória RAM DDR4 e SSD de 512 GB, rodando o sistema operacional Windows 11 x64.

4.2.1 Algoritmos Clássicos

A primeira estratégia adotada neste trabalho utiliza a ferramenta SPMF (Fournier-Viger et al., 2014), uma biblioteca de código-fonte aberto composta por diversos algoritmos relacionados à tarefa de descoberta de padrões sequenciais em *datasets*. Para extração dos padrões, foram escolhidos os algoritmos GSP, SPADE, FreeSpan, SPAM e LAPIN. Esses algoritmos foram executados contra o *dataset* considerando valores de suporte mínimo variando no intervalo de $0,1$ a 1.

A quantidade de padrões detectados, o tempo de processamento e a memória consumida em KB estão representados, respectivamente, nas Figuras 13, 15 e 17. Como os algoritmos se comportam de forma similar a partir de um valor de suporte mínimo igual a 0,30, também foram analisados o comportamento de tais algoritmos no intervalo de 0,1 a 0,30, conforme detalham as Figuras 14, 16 e 18.

Os algoritmos clássicos apresentam desempenhos similares em termos de tempo de processamento, memória consumida e quantidade de padrões detectados. Nesse sentido, apenas o algoritmo SPAM foi selecionado para uma análise detalhada, pois é mais flexível que os demais, possibilitando a escolha do tamanho mínimo da sequência. Para um limiar de 0,2%, foram encontradas 1.202 sequências frequentes com cinco interações; 258 com seis interações; e 15 com sete interações, resultando no total de 1.475 padrões com pelo menos cinco interações. As Tabelas 7 e 8 detalham, respectivamente, as cinco sequências mais frequentes e as cinco maiores sequências identificadas.

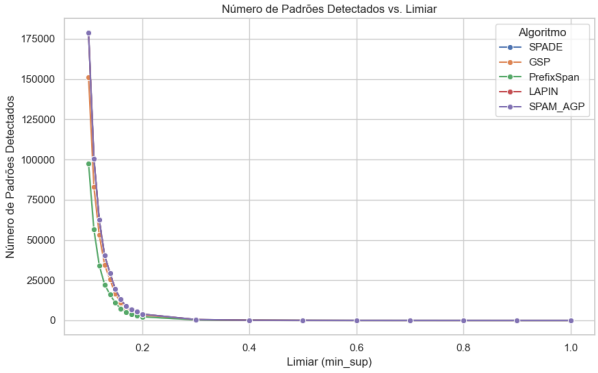

Figura 13: Quantidade de Padrões vs. Suporte Mínimo

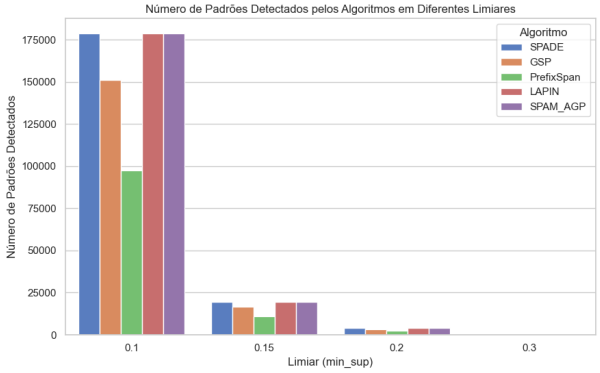

Figura 14: Detalhamento do Número de Padrões

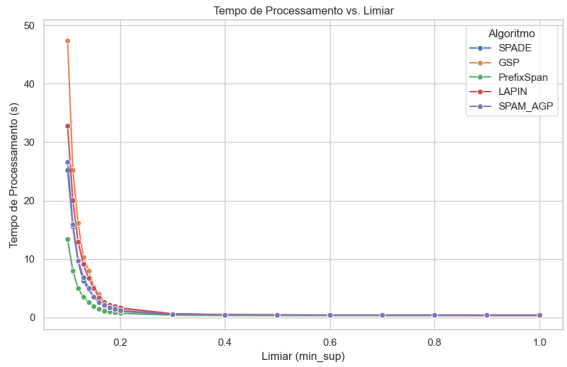

Figura 15: Tempo de Processamento vs. suporte mínimo

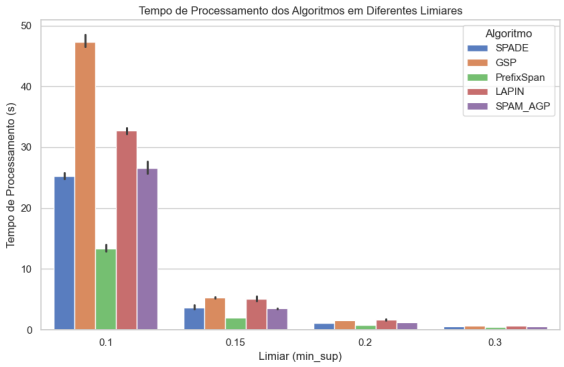

Figura 16: Detalhamento do Tempo de Processamento

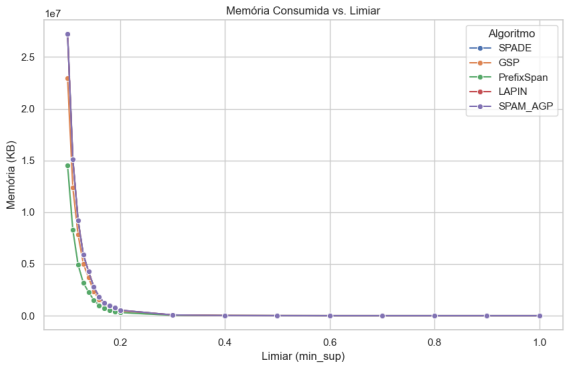

Figura 17: Memória Consumida vs. suporte mínimo

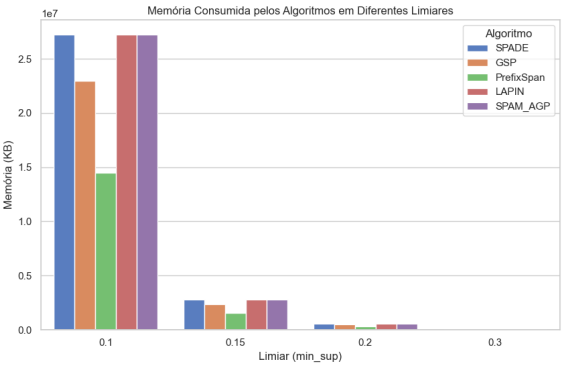

Figura 18: Detalhamento do Consumo de Memória

4.2.2 Metaheurísticas Evolutivas

A segunda estratégia consiste na geração de soluções candidatas utilizando quatro metaheurísticas distintas: AG, OEP, OCF e ABH. A função objetivo utilizada para avaliar a qualidade dessas

Sequência	Frequência
3_1,6_1,6_2,6_3,6_4	122
6_1,6_2,6_3,6_4,6_5	115
3_1,6_1,6_2,6_3,8_1	107
3_1,6_1,8_1,14_1,15_1	104
3_1,4_1,14_1,15_1,19_1	97

Tabela 7: Cinco sequências mais frequentes

Sequência	Frequência
3_1,6_1,6_2,6_3,6_4,6_5,6_6	91
6_1,6_2,6_3,6_4,6_5,6_6,6_7	87
3_1,6_2,6_3,6_4,6_5,6_6,6_7	84
3_1,6_1,6_2,6_3,6_4,6_5,6_7	83
3_1,6_1,6_2,6_3,6_4,6_6,6_7	83

Tabela 8: Cinco maiores sequências.

soluções foi a maximização do somatório das arestas do grafo. Essa função objetivo, quando aplicada a uma solução candidata, gera um valor chamado *fitness*, que reflete quão consistente e frequente é a trajetória de aprendizado identificada entre os alunos.

Cada metaheurística foi configurada para buscar sequências de tamanho 7, pois análises preliminares mostraram que esse valor corresponde ao tamanho máximo de padrões identificados pelos algoritmos clássicos. A quantidade de 10.000 gerações foi adotada após experimentos iniciais indicarem que, a partir desse ponto, as soluções tendiam a se estabilizar. Além disso, cada algoritmo foi executado 30 vezes para que a média amostral se aproximasse da normalidade e as estimativas fossem mais representativas. Os parâmetros de inicialização dos algoritmos estão detalhados na Tabela 9.

As Figuras 19, 20 e 21 detalham, respectivamente, o valor da função de *fitness* por geração, o tempo de execução (em segundos) e a memória consumida (em MB).

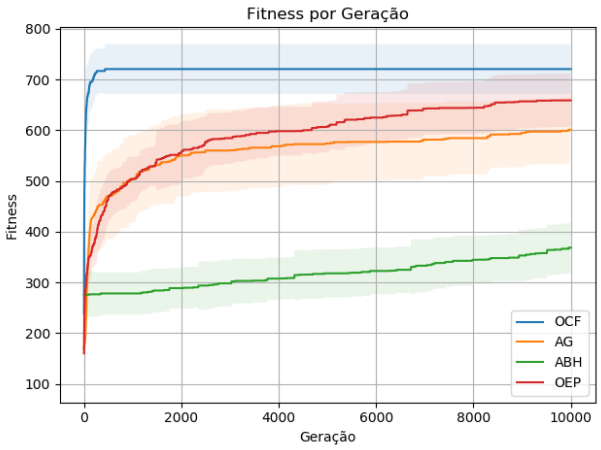

Figura 19: *Fitness* vs. Geração

Os resultados indicam que o algoritmo OCF apresenta desempenho superior em termos de *fitness* em comparação com os demais, para o conjunto de dados em análise. Na prática, isso significa que o algoritmo é capaz de descobrir a trajetória de aprendizagem mais adotada pelos alunos de forma mais rápida. O desempenho superior do OCF pode ser atribuído à sua capacidade natural de representar sequências. Como as soluções são construídas incrementalmente com base na probabilidade de transições entre estados e na influência dos feromônios depositados ao longo das iterações, o algoritmo explora de forma mais direcionada o espaço de busca, reforçando trajetórias promissoras.

Tabela 9: Comparação dos Parâmetros de Inicialização das Metaheurísticas Evolutivas

Parâmetro	Descrição	AG	ABH	OCF	OEP
Tamanho da Sequência	Número de componentes da solução ou dimensão do problema	7	7	7	7
Número de Iterações	Limite máximo de iterações (ou gerações)	10.000	10.000	10.000	10.000
População / Enxame / Colônia	Quantidade de soluções consideradas em cada iteração	100	–	100	100
Taxa de Mutação (*mutation_rate*)	Probabilidade de alteração de genes para garantir diversidade	0,2	–	–	–
Taxa de Elitismo (*elitism_rate*)	Percentual de indivíduos de melhor desempenho preservados intactos	0,3	–	–	–
Penalização à Diversidade (*diversity_penalty*)	Fator para desestimular convergência prematura, punindo soluções muito similares	0,2	–	–	–
Harmony Memory Size (*HMS*)	Quantidade de harmonias (soluções) armazenadas em memória.	–	500	–	–
Harmony Memory Considering Rate (*HMCR*)	Probabilidade de selecionar valores diretamente da memória ao criar nova harmonia.	–	0,9	–	–
Pitch Adjusting Rate (*PAR*)	Taxa de ajuste fino para modificar valores já selecionados da memória	–	0,4	–	–
Bandwidth (*BW*)	Intervalo (amplitude) de variação para ajuste de tom.	–	10	–	–
Taxa de Evaporação (*evaporation_rate*)	Taxa de dissipação do feromônio depositado pelas formigas	–	–	0,1	–
alpha	Fator de influência dos feromônios na escolha do caminho	–	–	1,0	–
beta	Fator de influência das informações heurísticas na escolha do caminho	–	–	1,0	–
Fator de Inércia	Controla a velocidade de deslocamento das partículas de uma iteração para outra	–	–	–	0,5
Fator de Influência Individual	Define a influência de atração em torno da melhor posição encontrada pela própria partícula, estimulando a busca local	–	–	–	1,5
Fator de Influência Coletiva	Determina a influência de atração em torno da melhor posição global do enxame, fomentando a busca global	–	–	–	1,5

Outros algoritmos que apresentaram bom desempenho foram o OEP e o AG, pois também conseguiram encontrar sequências relevantes dentro da quantidade de gerações anteriormente configurado. Em contrapartida, o algoritmo ABH apresentou desempenho consideravelmente inferior, necessitando cerca de 50.000 gerações para alcançar a convergência. O mecanismo de ajuste das soluções no ABH é baseado em perturbações aleatórias nas posições da harmonia, sem um mecanismo explícito para garantir a continuidade lógica das sequências geradas. Como resultado, o ABH pode gerar caminhos que não refletem padrões reais na base de dados, impactando negativamente sua eficiência na descoberta de sequências válidas.

Em termos de tempo de execução e consumo de memória, o algoritmo OCF apresenta

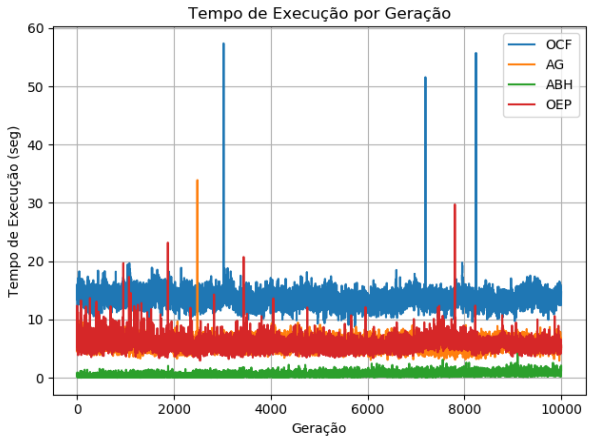

Figura 20: Tempo de Execução vs. Geração

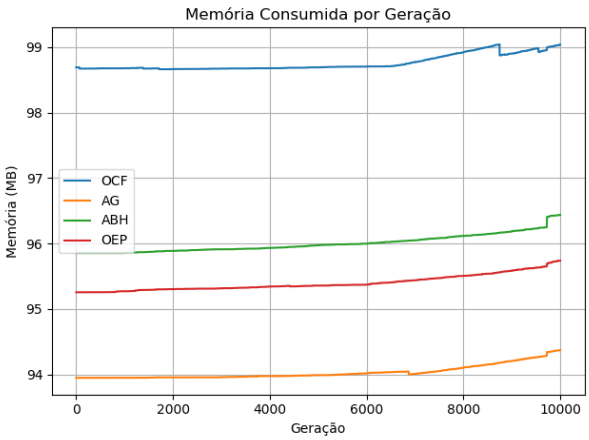

Figura 21: Memória Consumida vs. Geração

desvantagens em relação aos concorrentes. Seu tempo de execução é quase duas vezes maior que o dos outros algoritmos, e seu consumo de memória também é mais elevado. Por outro lado, os algoritmos OEP e AG conseguem alcançar resultados similares em termos de *fitness*, mas com menor uso de recursos. O algoritmo ABH destaca-se pelo menor tempo de processamento, pois realiza apenas um ajuste na população a cada geração, substituindo a pior harmonia por uma solução candidata melhor, quando essa é encontrada.

A Tabela 10 detalha os padrões mais frequentes identificados pelos algoritmos.

Tabela 10: Dez maiores sequências.

Id.	Padrão Sequencial	Soma	Id.	Padrão Sequencial	Soma
1	*inicio*,3_1,6_1,6_2,6_3,6_4,6_5	830	6	6_1,6_2,6_3,6_4,6_5,6_6,6_7	717
2	*inicio*,3_1,4_1,6_1,6_2,6_3,6_4	759	7	*inicio*,3_1,6_2,6_3,6_4,6_5,6_6	699
3	*inicio*,3_1,3_2,3_3,6_1,6_2,6_3	737	8	5_1,6_1,6_2,6_3,6_4,6_5,6_6	683
4	*inicio*,3_1,6_1,6_2,6_3,6_4,7_1	728	9	*inicio*,3_1,4_1,12_1,12_2,12_3,12_4	663
5	*inicio*,3_1,6_1,6_2,6_3,6_4,7_2	723	10	*inicio*,3_1,4_1,5_1,11_1,11_2,11_3	620

5 Discussão dos Resultados

As sequências obtidas por meio dos algoritmos clássicos (em especial, o SPAM) revelam não apenas a dificuldade recorrente dos alunos com a questão 6, mas também destacam alguns padrões de navegação atípicos entre os tópicos. A análise das sequências mostra que muitos alunos realizam a transição direta da questão 4, pertencente ao tópico de "variáveis e atribuição", para as questões 14, 15 e 19, do tópico "comandos condicionais". Esse padrão de navegação, identificado com frequência, sugere que os alunos estão buscando resolver problemas de temas mais avançados antes de consolidar o entendimento das bases iniciais.

No mesmo sentido, os resultados obtidos por meio das metaheurísticas evolutivas evidenciam que os alunos enfrentam grandes dificuldades para superar as questões 6, 7, 11 e 12. Essas questões se destacam em relação às demais, justificando a dificuldade adicional enfrentada pelos alunos. Nas questões 6 e 7, por exemplo, o professor exige a formatação da saída com quantidade específica de casas decimais, porém, a plataforma não fornece exemplos de como fazer isso. As questões 11 e

12 envolvem comandos condicionais e conceitos matemáticos de sistemas lineares, acrescentando um nível extra de complexidade.

Ainda com base nos padrões obtidos pelas metaheurísticas, a Figura 22 mostra que alunos normalmente resolvem a questão 3 (“Hello World”) e, em seguida, avançam diretamente para a questão 6 (“Área de um Retângulo Real”), que exige a formatação da saída com duas casas decimais, o que ajuda a justificar a quantidade excessiva de erros na questão 6. Outra característica observada é que os alunos tentam responder às questões do tópico “comandos condicionais” (questões 5 e 11) sem terem concluído as questões de “variáveis e atribuição”, o que não condiz com o comportamento desejado pelo professor.

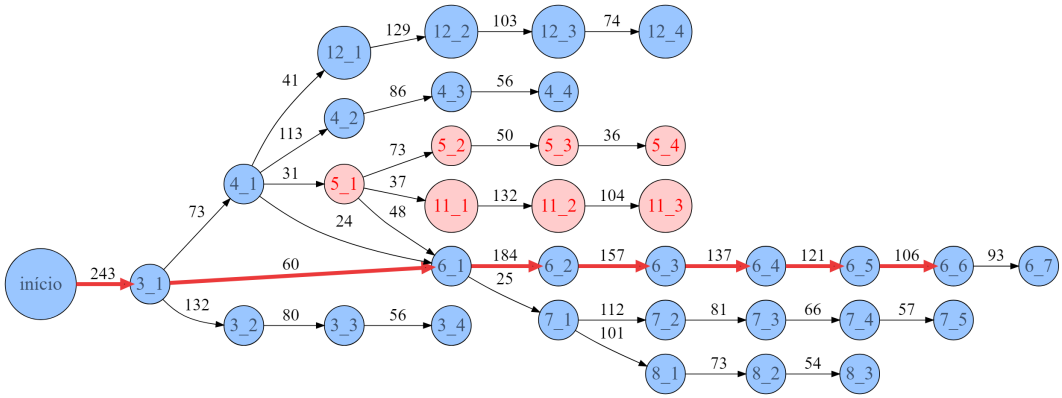

Figura 22: Grafo obtido a partir das maiores sequências.

Observa-se que ambas as abordagens podem contribuir bastante para a melhoria do ambiente de aprendizagem. Elas podem tanto ajudar a identificar quais questões precisam de revisão quanto auxiliar na definição da sequência mais adequada de atividades na interface gráfica. Como resultado, os alunos percebem aumento gradual de dificuldade, o que os mantém motivados a continuar usando o ambiente. Além disso, essas técnicas oferecem ao professor uma visão mais profunda do processo de aprendizagem, mostrando como os estudantes interagem com a plataforma e possibilitando intervenções rápidas para prevenir a desmotivação.

Por fim, comparando as duas abordagens, percebe-se que os algoritmos clássicos são mais indicados quando a exaustividade e a precisão são fundamentais, desde que o volume de dados não seja muito grande. Por outro lado, em cenários onde a escalabilidade é um problema e se deseja explorar um espaço de soluções mais amplo, as metaheurísticas representam uma alternativa eficiente e flexível. Além disso, elas podem ser combinadas aos algoritmos clássicos em hibridizações, aproveitando o que cada estratégia tem de melhor.

5.1 Ameaças à Validade

Embora os resultados obtidos sejam promissores, é importante considerar fatores que podem comprometer a validade da pesquisa. A seguir, analisam-se as quatro dimensões clássicas:

- **Validade Interna:** as metaheurísticas utilizadas (AGP, OEP, OCF e ABH) dependem de parâmetros como número de gerações, tamanho da população e critérios de parada.

Embora essas configurações sigam recomendações da literatura, variações nesses parâmetros poderiam alterar significativamente os resultados. Além disso, o uso de um único ambiente de execução ajuda a minimizar, mas não elimina totalmente, variações de tempo causadas por fatores externos.

- **Validade Externa:** os algoritmos foram avaliados em um único conjunto de dados, o que limita a generalização dos resultados. Mesmo que o conjunto represente bem o contexto de uso da plataforma, bases com características diferentes (como sequências mais longas, maior variedade de questões ou comportamentos mais ruidosos) podem produzir padrões distintos. A escolha dos algoritmos, embora representativa, também não abrange todas as abordagens possíveis.
- **Validade de Conclusão:** cada algoritmo foi executado 30 vezes com o objetivo de reduzir o impacto de variações aleatórias, o que favorece a aproximação da média amostral à distribuição normal. Com isso, é possível obter medidas descritivas mais representativas. No entanto, a ausência de testes estatísticos formais limita a força das comparações entre algoritmos.
- **Validade de Construção:** as métricas utilizadas — número de padrões, uso de memória, tempo de execução e valor de fitness — foram consideradas adequadas para representar o desempenho dos algoritmos. Contudo, outras dimensões, como a utilidade prática dos padrões descobertos ou sua relevância pedagógica, não foram avaliadas. Isso pode restringir a interpretação dos resultados em contextos educacionais mais amplos.

Portanto, as decisões metodológicas adotadas neste estudo buscaram equilibrar rigor experimental e viabilidade prática. Com isso, mesmo diante das ameaças identificadas, os resultados obtidos se mantêm relevantes para a compreensão do comportamento dos alunos, além de oferecerem subsídios valiosos para o aprimoramento de plataformas educacionais com base na mineração de padrões sequenciais.

6 Conclusão

O presente trabalho realiza um comparativo qualitativo entre algoritmos determinísticos clássicos e metaheurísticas evolutivas na tarefa de descoberta de padrões sequenciais em um ambiente real de ensino de algoritmos. O conjunto de dados analisado no presente trabalho diz respeito ao total de 13.916 submissões de código, escritas na linguagem de programação Python, realizadas por 313 alunos matriculados em 7 turmas diferentes, para o total de 68 problemas.

A modelagem adotada neste trabalho está baseada na construção de um grafo para representar a interação dos alunos com questões de uma plataforma de ensino. Os nós representam as questões, enriquecidos com informações sobre a quantidade de tentativas, e as arestas indicam as transições entre questões, criando trajetórias de aprendizado. Os experimentos foram organizados em duas abordagens distintas para facilitar a análise comparativa: algoritmos clássicos de mineração de padrões sequenciais e metaheurísticas evolutivas.

Na primeira abordagem, aplicam-se algoritmos clássicos (GSP, SPADE, FreeSpan, SPAM e LAPIN) para identificar padrões sequenciais de aprendizagem com base nas trajetórias mais recorrentes. Os algoritmos foram executados contra o *dataset* para valores de suporte mínimo

variando no intervalo de $0,1$ a 1. As sequências obtidas revelam a dificuldade recorrente dos alunos com a questão 6 e destacam alguns padrões de navegação atípicos entre os tópicos, como a transição direta da questão 4, que pertence ao tópico de variáveis e atribuição, para as questões 14, 15 e 19, que abordam comandos condicionais.

Na segunda, explora-se o uso de metaheurísticas evolutivas (AG, OEP, OCF, ABH), especialmente vantajosas na detecção de padrões complexos que os algoritmos clássicos podem não captar em virtude de sua natureza determinística e menos flexível. As metaheurísticas foram usadas para gerar soluções candidatas, buscando maximizar o somatório das arestas do grafo para sequências de tamanho igual a sete. As soluções foram avaliadas ao longo de 10.000 gerações, detalhando o valor de *fitness* por geração, tempo de execução e memória consumida. Experimentos foram realizados 30 vezes, resultando em um banco de sequências ordenado pelos pesos das arestas.

Os resultados indicam que o algoritmo OCF teve o melhor desempenho em identificar trajetórias de aprendizado, embora com maior tempo de execução e consumo de memória. O OEP e AG também mostraram bom desempenho, com menor uso de recursos, enquanto o ABH necessitou de mais gerações para convergir. Os padrões mais frequentes revelaram dificuldades dos alunos nas questões 6, 7, 11 e 12, muitas vezes por causa da complexidade adicional não exemplificada pela plataforma.

As duas abordagens podem melhorar o ambiente de aprendizagem de diferentes maneiras, seja ajudando a identificar quais questões precisam de revisão ou facilitando para que os alunos as superem com menos tentativas. Além disso, permitem definir uma sequência eficaz para a apresentação das questões na interface gráfica, garantindo que os alunos percebam aumento gradual na dificuldade e se mantenham motivados a continuar usando a plataforma. Finalmente, oferecem ao professor compreensão mais profunda do processo de aprendizagem, mostrando como os alunos interagem com a plataforma e permitindo intervenções rápidas para evitar a desmotivação.

Como trabalhos futuros, vislumbra-se a investigação do emprego de abordagens híbridas entre algoritmos clássicos e metaheurísticas evolutivas para extração de padrões sequenciais, bem como a integração dessa abordagem na plataforma de ensino com o objetivo de automatizar a recomendação de problemas com base na navegação de usuários com comportamentos de aprendizagem similares. Além disso, essa integração pode fornecer *feedback* imediato ao professor, permitindo ajustes conforme os alunos utilizam a plataforma.

Referências

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. *Proceedings of the eleventh international conference on data engineering*, 3–14. https://doi.org/10.1109/ICDE.1995.380415 [GS Search].

Almeida, D. J., Fernandes, M. A., & da Costa, N. T. (2022). Sequencing and Recommending Pedagogical Activities from Bloom's Taxonomy using RASI and Multi-objective PSO. *Proceedings of the 14th International Conference on Computer Supported Education - Volume 2: CSEDU*, 105–116. https://doi.org/10.5220/0011090000003182 [GS Search].

Ayres, J., Flannick, J., Gehrke, J., & Yiu, T. (2002). Sequential pattern mining using a bitmap representation. *Proceedings of the eighth ACM SIGKDD international conference on*

Knowledge discovery and data mining, 429–435. https://doi.org/10.1145/775047.775109 [GS Search].

Bhaskar, M., Das, M. M., Chithralekha, T., & Sivasatya, S. (2010). Genetic algorithm based adaptive learning scheme generation for context aware e-learning. *International Journal on Computer Science and Engineering*, *2*(4), 1271–1279. [GS Search].

Chiu, D.-Y., Wu, Y.-H., & Chen, A. L. (2004). An efficient algorithm for mining frequent sequences by a new strategy without support counting. *Proceedings. 20th International Conference on Data Engineering*, 375–386. https://doi.org/10.1109/ICDE.2004.1320012 [GS Search].

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms* [PhD Thesis]. Politecnico di Milano. [GS Search].

Eberhart, R., & Kennedy, J. (1995). A New Optimizer Using Particle Swarm Theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 39–43. https://doi.org/10.1109/MHS.1995.494215 [GS Search].

Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.-W., Tseng, V. S., et al. (2014). Spmf: a java open-source pattern mining library. *J. Mach. Learn. Res.*, *15*(1), 3389–3393. https://doi.org/10.5555/2627435.2750353 [GS Search].

Geem, Z., Kim, J., & et al. (2001). A new heuristic optimization algorithm: harmony search. *Simulation*, *76*(2), 60–68. https://doi.org/10.1177/003754970107600201 [GS Search].

Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., & Hsu, M.-C. (2000). FreeSpan: frequent pattern-projected sequential pattern mining. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 355–359. https://doi.org/10.1145/347090.347167 [GS Search].

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. *ACM sigmod record*, *29*(2), 1–12. https://doi.org/10.1145/335191.335372 [GS Search].

Hnida, M., Idrissi, M. K., & Bennani, S. (2016). Adaptive teaching learning sequence based on instructional design and evolutionary computation. *2016 15th International Conference on Information Technology Based Higher Education and Training (ITHET)*, 1–6. https://doi.org/10.1109/ITHET.2016.7760739 [GS Search].

Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press. https://doi.org/10.7551/mitpress/1090.001.0001 [GS Search].

Kang, J., Liu, M., & Qu, W. (2017). Using gameplay data to examine learning behavior patterns in a serious game. *Computers in Human Behavior*, *72*, 757–770. https://doi.org/10.1016/j.chb.2016.09.062 [GS Search].

Li, J.-W., Chang, Y.-C., Chu, C.-P., & Tsai, C.-C. (2012). A self-adjusting e-course generation process for personalized learning. *Expert Systems with Applications*, *39*(3), 3223–3232. https://doi.org/https://doi.org/10.1016/j.eswa.2011.09.009 [GS Search].

Machado, M. d. O. C., Barrére, E., & Souza, J. (2019). Solving the Adaptive Curriculum Sequencing Problem with Prey-Predator Algorithm. *International Journal of Distance Education Technologies (IJDET)*, *17*(4), 71–93. https://doi.org/10.4018/IJDET.2019100105 [GS Search].

Maranhão, D., Borges, P., & Neto, C. (2023). Descoberta de padrões sequenciais de aprendizagem em um ambiente voltado ao ensino de algoritmos. *Anais do XXXIV Simpósio Brasileiro de Informática na Educação*, 1385–1396. https://doi.org/10.5753/sbie.2023.235102 [GS Search].

Maranhão, D., & Neto, C. S. (2024). Aplicação de Metaheurísticas Evolutivas na Mineração de Padrões Sequenciais de Aprendizagem em Ambientes de Ensino de Algoritmos. *Anais do XXXV Simpósio Brasileiro de Informática na Educação*, 1837–1850. https://doi.org/10.5753/sbie.2024.242581 [GS Search].

Martins, A. F., Machado, M., Bernardino, H. S., & de Souza, J. F. (2021). A comparative analysis of metaheuristics applied to adaptive curriculum sequencing. *Soft Computing*, *25*(16), 11019–11034. https://doi.org/10.1007/s00500-021-05836-9 [GS Search].

Mirzaei, M., & Sahebi, S. (2019). Modeling students' behavior using sequential patterns to predict their performance. *Artificial Intelligence in Education: 20th International Conference, AIED 2019, Chicago, IL, USA, June 25-29, 2019, Proceedings, Part II 20*, 350–353. https://doi.org/10.1007/978-3-030-23207-8_64 [GS Search].

Parpinelli, R. S., & Lopes, H. S. (2011). New inspirations in swarm intelligence: a survey. *International Journal of Bio-Inspired Computation (IJBIC)*, *3*, 1–16. https://doi.org/10.1504/IJBIC.2011.038700 [GS Search].

Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., & Hsu, M.-C. (2001). PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. *Proceedings of the 17th International Conference on Data Engineering (ICDE'01)*, 215–224. https://doi.org/10.1109/ICDE.2001.914830 [GS Search].

Sharma, R., Banati, H., & Bedi, P. (2012). Adaptive content sequencing for e-learning courses using ant colony optimization. *Proceedings of the International Conference on Soft Computing for Problem Solving (SocProS 2011) December 20-22, 2011*, 579–590. https://doi.org/10.1007/978-81-322-0491-6_53 [GS Search].

Slim, A., Heileman, G. L., Al-Doroubi, W., & Abdallah, C. T. (2016). The impact of course enrollment sequences on student success. *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, 59–65. https://doi.org/10.1109/AINA.2016.140 [GS Search].

Srikant, R., & Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. *Advances in Database Technology—EDBT'96: 5th International Conference on Extending Database Technology Avignon, France, March 25–29, 1996 Proceedings 5*, 1–17. https://doi.org/10.1007/BFb0014140 [GS Search].

Yang, Z., Wang, Y., & Kitsuregawa, M. (2007). LAPIN: effective sequential pattern mining algorithms by last position induction for dense databases. *Advances in Databases: Concepts, Systems and Applications: 12th International Conference on Database Systems for Advanced Applications, DASFAA 2007, Bangkok, Thailand, April 9-12, 2007. Proceedings 12*, 1020–1023. https://doi.org/10.1007/978-3-540-71703-4_95 [GS Search].

Zaki, M. J. (2000). Sequence mining in categorical domains: incorporating constraints. *Proceedings of the Ninth International Conference on Information and Knowledge Management*, 422–429. https://doi.org/10.1145/354756.354849 [GS Search].

Zaki, M. J. (2001). SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning*, *42*(1-2), 31–60. https://doi.org/10.1023/A:1007652502315 [GS Search].

Zhang, Y., & Paquette, L. (2023). Sequential pattern mining in educational data: The application context, potential, strengths, and limitations. Em *Educational Data Science: Essentials, Approaches, and Tendencies: Proactive Education based on Empirical Big Data Evidence* (pp. 219–254). Springer Nature Singapore. https://doi.org/10.1007/978-981-99-0026-8_6 [GS Search].

Zhou, M., Xu, Y., Nesbit, J. C., & Winne, P. H. (2010). Sequential pattern analysis of learning logs: Methodology and applications. *Handbook of educational data mining*, *107*, 107–121. https://doi.org/10.1201/b10274-10 [GS Search].