Universidade Estadual de Feira de Santana

Programa de Pós-Graduação em Ciência da Computação

Encontrando Regras de Associação sem Especificar Suporte e Confiança

Oto Antonio Lopes Cunha Filho

Feira de Santana

2023

Universidade Estadual de Feira de Santana

Programa de Pós-Graduação em Ciência da Computação

Oto Antonio Lopes Cunha Filho

Encontrando Regras de Associação sem Especificar Suporte e Confiança

Dissertação apresentada à Universidade Estadual de Feira de Santana como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Orientador: João B. Rocha-Junior

Feira de Santana

2023

Ficha Catalográfica – Biblioteca Central Julieta Carteado

C978e Cunha Filho, Oto Antonio Lopes
Encontrando regras de associação sem especificar suporte e confiança / Oto Antonio Lopes Cunha Filho .- Feira de Santana, 2023.

83f.: il
Orientador: João B. Rocha - Júnior
Dissertação (Mestrado) – Universidade Estadual de Feira de Santana, Programa de Pós-graduação em Ciência da Computação , 2023.

1. Regras de associação. 2. Consultas preferenciais. 3. Desempenho 4. Amostragem. I. Rocha Júnior, João B., orient. II. Universidade Estadual de Feira de Santana. III. Título.

CDU: 681.3

Tatiane Souza Santos - Bibliotecária CRB5/1634

Oto Antonio Lopes Cunha Filho

Encontrando Regras de Associação sem Especificar Suporte e Confiaça

Dissertação apresentada à Universidade Estadual de Feira de Santana como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Feira de Santana, 15 de dezembro de 2022

BANCA EXAMINADORA

João Batista da Rocha Junior (Orientador(a))
Universidade Estadual de Feira de Santana

Daniela Barreiro Claro
Universidade Federal da Bahia

Angelo Conrado Loula
Universidade Estadual de Feira de Santana

Abstract

The extraction of information and knowledge in databases has been assuming a relevant role in aiding decision making. One of the main areas of research is association rule mining. This area makes it possible to capture relationships between attributes present in a database.

Most algorithms used to extract association rules use support and confidence as parameters. Support represents the proportion of a given rule in the database and confidence represents the validity of this rule. Thus, professionals responsible for data analysis need to identify and define support and confidence thresholds (minimum support and minimum confidence, respectively) to obtain association rules.

However, in certain contexts, it is difficult to identify good values for support and confidence in order to obtain the desired rules. In these situations, it may be necessary to run several queries with different values of support and confidence in order to obtain the desired rules.

The purpose of this research is to examine association rules mining techniques and algorithms capable of obtaining association rules without the need to specify support and confidence, propose new algorithms and analyze these algorithms in terms of performance and quality of the obtained rules.

Keywords: Association Rules, Preference Queries, Performance, Sampling.

Resumo

A extração de informações e de conhecimento em base de dados vem assumindo um papel relevante no auxílio à tomada de decisão. Uma das principais áreas de pesquisa é a mineração de regras de associação. A partir dela é possível capturar relações entre atributos presentes em um banco de dados.

A maioria dos algoritmos utilizados para extrair regras de associação utilizam como parâmetro suporte e confiança. O suporte representa a proporção de uma determinada regra no banco de dados e a confiança representa a validade desta regra. Desta forma os profissionais responsáveis pelas análises dos dados precisam identificar e definir limiares de suporte e confiança (suporte mínimo e confiança mínima, respectivamente) para obter as regras de associação.

No entanto, em certos contextos, é difícil identificar bons valores para suporte e confiança a fim de obter as regras desejadas. Nestas situações pode ser necessário a execução de diversas consultas com valores diferentes de suporte e confiança até obter as regras pretendidas.

A finalidade desta pesquisa é examinar as técnicas e algoritmos de mineração de regras de associação capazes de obter regras de associação sem a necessidade de especificar suporte e confiança, propor novos algoritmos e analisar estes algoritmos em termos de performance e qualidade das regras obtidas.

Palavras-chave: Regras de Associação, Consultas Preferenciais, Desempenho, Amostragem.

Prefácio

Esta dissertação de mestrado foi submetida à Universidade Estadual de Feira de Santana (UEFS) como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

A dissertação foi desenvolvida no Programa de Pós-Graduação em Ciência da Computação (PGCC), tendo como orientador o Prof. Dr. **João B. Rocha-Junior**.

Agradecimentos

Preciso agradecer primeiramente aos meus parentes, familiares e amigos que sempre me deram apoio e incentivo nas horas difíceis, de desânimo e cansaço, assim como entenderam minhas ausências por conta dos estudos e nunca deixaram de estar próximos a mim.

Gostaria de agradecer ao meu orientador, Prof. Dr. João Batista Rocha Junior, pela oportunidade dada a mim de concluir mais esta etapa pessoal e profissional. Também preciso agradecer pela sua orientação firme e paciente, fornecendo respostas a muitas questões minhas, assim como, conselhos e caminhos a serem seguidos.

Agradeço a todos os professores, avaliadores e colegas que encontrei durante todo meu caminho acadêmico por me proporcionarem não apenas o conhecimento racional, mas a vontade de continuar aprendendo e crescendo na minha área profissional.

Por fim, e não menos importante, gostaria de agradecer à empresa Sanar Saúde e ao grupo São Roque pelo fornecimento de dados fundamentais para a pesquisa. Também agradeço ao Programa Interno de Auxílio Financeiro aos Programas de Pós-Graduação Stricto Sensu (AUXPPG) da Universidade Estadual de Feira de Santana (UEFS), ao Programa de Apoio à Pós-Graduação (PROAP) da CAPES, à equipe do SBBD 2022 - WTDBD e ao Programa de Pós-Graduação em Ciência da Computação (PGCC) da UEFS pelo apoio e incentivo à este trabalho.

Sumário

Alinhamento com a Linha de Pesquisa

Linha de Pesquisa: Software e Sistemas Computacionais

A presente dissertação discute técnicas e algoritmos de mineração de regras de associação capazes de obter regras de associação sem a necessidade do usuário especificar os limiares de suporte e confiança, assim como, apresenta novos algoritmos e análises destes algoritmos em termos de performance e qualidade das regras obtidas. Por conta disto, esta dissertação se enquadra nesta linha de pesquisa.

Produções Bibliográficas, Produções Técnicas e Premiações

Cunha Filho, Oto Antonio Lopes, and João Batista Rocha-Junior. "Encontrando Regras de Associação sem Especificar Suporte Mínimo e Confiança Mínima." Anais Estendidos do XXXVII Simpósio Brasileiro de Bancos de Dados. SBC, 2022.

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

"Imaginar é o princípio da criação. Nós imaginamos o que desejamos, queremos o que imaginamos e, finalmente, criamos aquilo que queremos."

– George Bernard Shaw

Atualmente, muitos dados são produzidos diariamente pela sociedade, que por sua vez, não pode se limitar a produção destes, mas utilizá-los em prol da economia e do povo (Santos, 2017). Diversas empresas veem estes dados como privilégios legais, estratégicos e imprescindíveis para maior autonomia em suas ações (Galvão e Marin, 2009). Todavia, devido ao fluxo e a forma como as informações são criadas e armazenadas, o ser humano é naturalmente incapaz de explorar, extrair e interpretar este contingente de informação para gerar um conhecimento válido (Fournier-Viger et al., 2017)

Nesse sentido, faz-se necessário o uso de técnicas e ferramentas de análise de dados. A mineração de dados é um processo de descoberta de estruturas interessantes, inesperadas ou valiosas em conjuntos de dados (Hand, 2007), assim, mecanismos de estatística e/ou Inteligência Computacional podem ser utilizadas para auxiliar na previsão de eventos, descoberta de padrões e relacionamentos válidos nos dados. Desta forma, muitas pesquisas e estudos são realizados para proporcionar mais clareza, facilidade e/ou agilidade na busca e extração de conhecimento utilizando a mineração de dados.

Introduzida por Agrawal et al. (1994) a mineração de conjuntos frequentes de itens é um subcampo da mineração de dados que consiste em extrair eventos, padrões ou itens frequentes nos dados. A análise desses padrões e eventos oferece benefícios significativos nos processos de tomada de decisão (Fournier-Viger et al., 2017).

Este subcampo de estudo foi inicialmente projetado para análise de cesta de compras, todavia se mantém presente em diversas outras áreas de pesquisa. Sendo visto de maneira mais geral, ele pode ser encontrado em bioinformática, mineração de texto, recomendação de produtos, educação e identificação de segmentos do mercado a partir de compras no cartão de crédito (Gupta e Chandra, 2020). Por conta de suas inúmeras aplicações em domínios diferentes, a mineração de conjuntos frequentes de itens se tornou uma área de pesquisa bastante popular (Fournier-Viger et al., 2017; Gupta e Chandra, 2020).

Os critérios ou medidas de interesse mais comuns utilizados na mineração de regras de associação são suporte e confiança. Suporte representa a porcentagem de transações da base de dados que contém os conjuntos frequentes de itens especificados por uma regra. Confiança representa a força e validade de uma regra considerando os conjuntos frequentes de itens que a compõem (Telikani et al., 2020).

A maioria dos algoritmos desenvolvidos para extrair regras de associação recebe como parâmetro valores mínimos de suporte e confiança e retornam todas as regras que possuem suporte e confiança maiores ou iguais aos valores mínimos passados como parâmetro. Levando em consideração que a maioria dos dados são dinâmicos (seu conteúdo e relações se alteram com o tempo), a identificação desses parâmetros exige um estudo prévio do conjunto de dados para cada momento em que a análise for realizada. Desta forma, é difícil selecionar bons valores de suporte e confiança para se obter as regras desejadas. Por exemplo, os valores de suporte e confiança que retornam poucas regras em uma base, podem retornar milhares de regras em outra. Portanto, extrair informações utilizando esses parâmetros é custoso e demorado, visto que várias consultas precisam ser realizadas até obter as regras pretendidas (Moslehi e Haeri, 2020).

Este cenário se torna ainda mais problemático quando falamos em grandes conjuntos de dados e/ou conjuntos que crescem rapidamente. Keogh et al. (2007) afirmam que um algoritmo sem parâmetros de entrada impede a imposição dos preconceitos e suposições sobre o problema em questão e permite que os próprios dados "falem" com os usuários. Quando se trabalha com muitos elementos, o estudo prévio de toda essa informação se torna algo inviável, tanto o custo quanto a complexidade destes procedimentos se intensificam demasiadamente.

A descoberta de muitos padrões torna difícil para um ser humano analisar os resultados encontrados sendo que para cada informação válida obtida, há muito mais informações triviais e/ou redundantes sendo geradas (Fournier-Viger et al., 2017). Desta forma, o processo deve garantir a qualidade da busca, reduzindo seu espaço de inspeção, e entregar ao usuário um conjunto de informações legíveis em um tempo aceitável.

Na tentativa de reduzir esses obstáculos diversas pesquisas com foco em mineração de dados foram surgindo. Alguns destes trabalhos apresentam meios para melhorar a eficiência do processo de mineração sem perder qualidade das regras. A exemplo de Vo e Le (2011), Han et al. (2004) e Li et al. (2005) que introduzem novas

técnicas e estruturas para diminuir o tempo de obtenção dos padrões frequentes. Zaki (2000) propôs um algoritmo eficiente para mineração em grandes conjuntos de dados. Deshmukh et al. (2019) apresentam um modelo paralelo para melhorar a eficiência da mineração de regras.

No entanto, estas abordagens mencionadas anteriormente ainda necessitam de parâmetros como suporte mínimo e confiança mínima.

Outra forma de abordar estes problemas são através de estudos na área de Inteligência Computacional. Martín et al. (2018) apresentaram um algoritmo evolutivo voltado para descobertas de regras de associação quantitativas em grandes bases de dados. Qodmanan et al. (2011) apresentaram técnicas baseadas em algoritmos genéticos para encontrar regras de associação sem que o usuário defina limites de suporte e confiança. Huang e Kao (2004) propuseram um modelo *fuzzy* para derivar os limites de suporte e confiança para minerar regras de associação.

Entretanto, não existem muitos trabalhos aprofundados sobre a qualidade das regras geradas, consumo de memória e tempo de processamento de algoritmos que tentam evitar a definição dos limiares de suporte e/ou confiança. Além disso, muitos destes algoritmos ainda utilizam parâmetros adicionais para seu funcionamento. Por exemplo, parâmetros de probabilidade, tamanho de estruturas de dados que serão utilizadas ao longo do processamento, taxa de erro aceitável, quantidade de iterações, informações de codificação e decodificação, além de outras medidas de interesse como *lift* e *leverage*.

Com o objetivo de remover o uso dos limiares de suporte e confiança, assim como reduzir o uso de parâmetros adicionais, este estudo se apoia em consultas preferenciais, mais especificamente nos métodos Top-k e Skyline. Estes métodos tem como objetivo recuperar os k melhores elementos e os elementos que não são dominados por outros numa determinada base de dados, respectivamente. Entretanto, para tornar possível a seleção deste conjunto de informações é preciso definir parâmetros preferenciais. "Como saber o que é preciso para ser o melhor?" ou "O que é preciso para não ser dominado por outros dados?". Parâmetros preferenciais são mais facilmente especificados, pois, o usuário informa o que deseja apenas uma única vez e a consulta retorna as melhores regras que atendem a estes critérios (preferências).

Han et al. (2002) apresentam um algoritmo eficiente, chamado TFP, desenvolvido para minerar regras sem suporte mínimo. Este algoritmo utiliza uma técnica de elevação do suporte, ou seja, o suporte mínimo começa zerado e vai sendo incrementado ao longo do processamento. O resultado deste algoritmo são as k regras resultantes das etapas de remoção de itens indesejados utilizando este suporte crescente. Qodmanan et al. (2011) propõem um método baseado em algoritmo genético sem levar em conta o suporte mínimo e a confiança mínima. Este método extrai as regras que têm a melhor correlação entre suporte e confiança utilizando uma função de aptidão.

Neste contexto, a abordagem de Qodmanan et al. (2011), que correlaciona suporte e

confiança apresenta grandes semelhanças com o estudo proposto. Pois, além de não necessitar dos limites de suporte e confiança, o mesmo considera estas medidas de interesse durante o processamento. Ele também utiliza uma função de aptidão para quantificar as regras e selecionar as melhores possíveis num dado espaço de busca.

1.1 Justificativa e Motivação

Como as organizações então cada vez mais competitivas, são necessários algoritmos que auxiliem na tomada de decisão, entregando bons resultados no menor tempo possível. Algoritmos de mineração de regras de associação fazem parte deste processo. Estes algoritmos são capazes de extrair relações presentes numa base de dados. Por exemplo, é possível encontrar relações entre doenças, pacientes e remédios. Também é possível utilizá-los para identificar conjuntos de itens de supermercado que são comprados em conjunto (Devi, 2012) ou até mesmo utilizá-los em sistemas de recomendação de produtos ou serviços online.

No entanto, conforme a quantidade de informações e complexidade das relações crescem, mais demorado fica o processo de configuração e execução destes algoritmos. Em termos de análise combinatória, é possível afirmar que para um banco de dados com apenas 10 itens distintos podem ser geradas mais de 50 mil regras. Desta forma, os profissionais responsáveis pelas análises dos dados precisam gastar horas e mais horas tentando encontrar os parâmetros adequados daquele contexto, para então começarem a trabalhar sobre as informações adquiridas.

Nesse contexto, faz-se necessário estudos a respeito de melhorias e/ou novos modelos de algoritmos que permitam encontrar relações relevantes de forma mais eficiente.

1.2 Objetivos

Este trabalho tem como objetivo examinar algumas técnicas e algoritmos de mineração de regras de associação capazes de obter regras de associação sem a necessidade de especificar suporte e confiança. Além disso, esta pesquisa apresenta novos algoritmos (que apelidamos de algoritmos *Iterativos*) e análises destes em termos de performance e qualidade das regras obtidas. Para isto os seguintes objetivos devem ser alcançados.

- Identificar algoritmos presentes na literatura capazes de obter regras de associação sem a necessidade de especificar suporte mínimo e/ou confiança mínima;
- Propor novos algoritmos capazes de obter regras de associação sem a necessidade de especificar suporte mínimo e confiança mínima;
- Comparar esses algoritmos em termos de desempenho (processamento e consumo de memória) e qualidade das regras produzidas

1.3 Contribuições

O resultado desta pesquisa pode contribuir para criar mecanismos que permitam simplificar o processo de mineração de regras de associação. Assim, seu potencial é voltado tanto para facilitar a seleção das regras de interesse, independentemente do conjunto de dados em questão, quanto para a qualidade das regras adquiridas num tempo hábil.

1.4 Organização do Trabalho

Este trabalho está estruturado da seguinte forma:

Capítulo 1: neste capítulo os problemas que este trabalho se propõem a estudar são apresentados, bem como os objetivos deste estudo em questão;

Capítulo 2: possui uma definição para termos comuns e/ou importantes para o entendimento da pesquisa, bem como trabalhos relacionados à mesma;

Capítulo 3: contém uma visão geral de como o estudo é conduzido, contendo elementos importantes para a replicação do mesmo;

Capítulo 4: os algoritmos utilizados durante esta pesquisa são apresentados e descritos a partir de seus pseudocódigos;

Capítulo 5: inclui algumas características e definições de elementos que compõem os experimentos;

Capítulo 6: contém discussões sobre os resultados obtidos e as devidas análises e comparações para cada algoritmo presente no estudo;

Capítulo 7: contém informações sobre uma aplicação web implementada durante o período da pesquisa, bem como dados sobre impressões de usuários sobre esta aplicação;

Capítulo 8: são apresentadas as considerações finais sobre os resultados e os possíveis passos para pesquisas futuras relacionadas;

Capítulo 2

Revisão Bibliográfica

"O importante é não parar de questionar. A curiosidade tem sua própria razão de existir."
– Albert Einstein

Santos (2017) apresenta a descoberta de conhecimento em base de dados (do inglês *Knowledge Discovery in Databases*, KDD) como um processo complexo de extração de informações implícitas e interessantes a partir de um conjunto de dados. O KDD é dividido nas seguintes etapas, as quais são ilustradas na Figura 2.1.

1. Entender o domínio da aplicação e definir seu objetivo;
2. Definir casos de uso e variáveis de exemplo que são aplicadas no KDD;
3. Realizar a limpeza e pré-processamento do subconjunto de dados. Assim, removendo dados inconsistentes e/ou indesejados, bem como homogeneizando os dados para processamento;
4. Realizar projeção e redução de dados para mitigar restrições de espaço, memória ou tempo de processamento. Por exemplo, eliminar atributos desnecessários para o processamento;
5. Realizar de fato a descoberta de conhecimento de padrões. Com a mineração de dados, grandes quantidades de informações são analisadas para extração de informações úteis;
6. Ao final do processo é avaliada a real usabilidade do conhecimento capturado.

A mineração de dados tem como objetivo prever um evento futuro ou compreender informações do passado (Fournier-Viger et al., 2017). Diversas ferramentas e técnicas são utilizadas através de algoritmos baseados em aprendizado de máquina

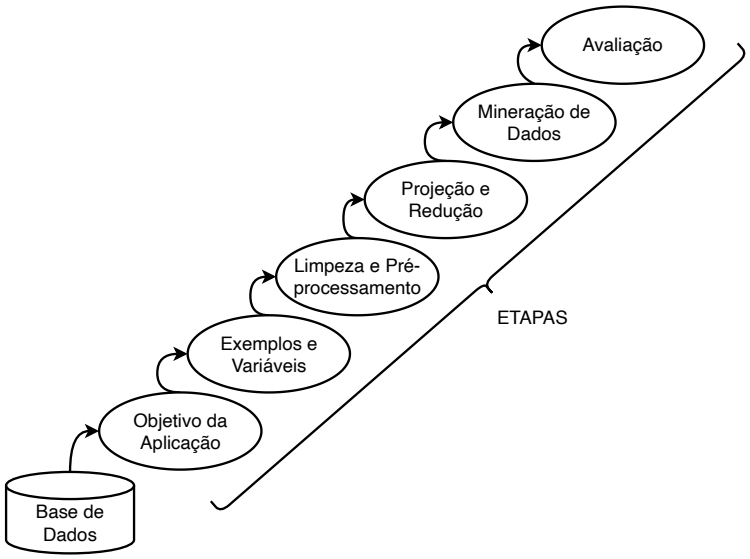


Figura 2.1: Etapas do processo de descoberta de conhecimento em base de dados. Fonte: Recriado a partir de Santos (2017).

e/ou estatística para criação de modelos mais precisos, que por sua vez, sejam capazes de explorar um conjunto de dados e extrair padrões, tendências ou informações proveitosas aos interessados. Santos (2017) afirma que as técnicas de mineração de dados mais comuns são a modelagem preditiva, que visa à construção de um modelo preditivo. A análise de agrupamento, que se baseia em medidas de similaridade para encontrar *clusters* (agrupamentos de dados). A detecção de anomalias (ou desvios), que busca identificar registros que não atendem ao padrão considerado normal. E a descoberta de associação (ou análise de associação), que foca em encontrar relacionamentos ou padrões frequentes na base de dados.

Para uma melhor compreensão desta pesquisa, faz-se necessária a exploração de alguns outros conceitos. A Seção 2.1 contém uma breve introdução sobre regras de associação, além de apresentar três algoritmos populares para mineração de regras de associação. A Seção 2.2 contém uma sucinta definição sobre algoritmos genéticos. A Seção 2.3 contém uma descrição sobre duas técnicas para consultas preferenciais. A Seção 2.4 contém informações sobre técnicas de amostragem. E a Seção 2.5 contém um resumo dos principais trabalhos relacionados a essa pesquisa.

2.1 Mineração de Regras de Associação

Moslehi e Haeri (2020) declaram que a mineração de dados consiste num conjunto de técnicas que são utilizadas para descobrir conhecimento, padrões ocultos e/ou regras a partir de informações de diversas outras ciências. Dentre estas técnicas, uma que se destaca é a mineração de regras de associação (Moslehi e Haeri, 2020). Voltada para o auxílio à decisão, a mineração de regras de associação é responsável por encontrar relações entre atributos de um grande conjunto de dados (Agrawal

et al., 1994). Vale ressaltar que uma das principais etapas para a geração de regras de associação envolve a busca de conjuntos frequentes de itens (Luna et al., 2019).

A busca por itens frequentes ainda pode ser mais específica quando envolve os conceitos de itens máximos ou itens fechados. Um determinado conjunto de itens é fechado, se e somente se, nenhum de seus superconjuntos imediatos possuir o mesmo suporte do conjunto em questão (Anselmo, 2017). Já um conjunto de itens se caracteriza como máximo, se e somente se, nenhum de seus superconjuntos imediatos for frequente (Anselmo, 2017).

Se tratando de itens fechados frequentes ou itens máximos frequentes, além de serem fechados ou máximos, respectivamente, eles também precisam ter suporte superior ao suporte limite (suporte mínimo). A Figura 2.2 ilustra a relação entre estas categorias.

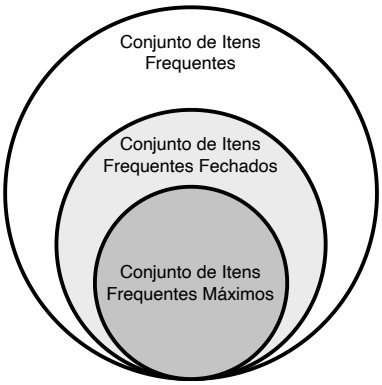


Figura 2.2: Relacionamento entre itens frequentes, fechados e máximos. Fonte: Recriado a partir de Anselmo (2017)

Entre as abordagens mais conhecidas que apoiam ou atuam sobre a mineração de regras de associação estão o Apriori (Seção 2.1.1), Eclat (Seção 2.1.2) e FP-Growth (Seção 2.1.3).

Regras de Associação

Segundo Agrawal et al. (1996), regras de associação são expressões no formato $X \Rightarrow Y$, onde X e Y são conjuntos de itens. Tal expressão tem o intuito de identificar relações entre registros num banco de dados, onde transações que contém X também conterão Y. Por exemplo, dada uma base de dados onde são registrados itens adquiridos por clientes, é possível gerar a seguinte regra: $\{cinto, bolsa\} \Rightarrow \{sapato\}$, a qual indica que na compra de cinto e bolsa, considerando um determinado grau de certeza, sapato também é comprado (Vasconcelos e Carvalho, 2018).

Este grau de certeza de uma regra é definido, na maioria das vezes, pelas medidas de suporte e confiança. Lai e Cerpa (2001) afirmam que a mineração de dados tende a produzir uma quantidade elevada de regras. Nesse sentido, tanto o suporte quanto a confiança são usados para auxiliar na filtragem das regras úteis para os usuários,

assim como, permitir a redução do custo computacional. As regras são úteis quando os valores de suporte e confiança são iguais ou ultrapassam um limite definido pelo próprio usuário (limites estes chamados de suporte mínimo e confiança mínima).

Conjunto Frequente de Itens

O conceito de conjunto frequente de itens foi introduzido inicialmente para mineração de transações em bases de dados (Agrawal et al., 1993). Considerando uma base de dados D contendo n itens distintos, onde $I = \{i_1, i_2..., i_n\}$ é o conjunto de todos estes itens. Um subconjunto de itens contendo k itens de I será frequente, se e somente se, estiver presente em no mínimo $\theta \cdot |D|$ transações na base de dados D, onde θ representa o limiar de suporte definido pelo usuário (suporte mínimo) e $|D|$ o total de transações em D (Agrawal et al., 1993).

Suporte

O suporte representa a frequência que um conjunto de itens (X) aparece nas transações do banco de dados (T), conforme Equação 2.1. Desse modo, o suporte de uma regra $X \Rightarrow Y$ torna-a frequente caso $Sup(X \cup Y)$ apresente um valor superior ao limite definido.

$$Sup(X) = \frac{Frequência\ de\ X}{Total\ T} \tag{2.1}$$

Com base na Tabela 2.1, o suporte para a regra $A \Rightarrow B$ é 40% pois, do total de cinco transações, duas delas (transações 1 e 2) contém o item A e o B. A tabela também contém outras regras como $B \Rightarrow C$ e $\{A, B\} \Rightarrow C$, com suporte de 60% e 20%, respectivamente.

Tabela 2.1: Exemplo da medida de suporte.

Id da Transação	**Itens**	**Ex.: Valores de Suporte**
1	{A, B, C}	Total Transações = 5 Sup$(A \cup B) = \frac{2}{5} = 40\%$ Sup$(B \cup C) = \frac{3}{5} = 60\%$ Sup$(A \cup B \cup C) = \frac{1}{5} = 20\%$
2	{A, B, D}	
3	{B, C}	
4	{A, C}	
5	{B, C, D}	

Confiança

A medida de confiança ou precisão preditiva de uma regra $X \Rightarrow Y$ é definida pela Equação 2.2 e indica a quantidade de vezes que ao existir o conjunto de itens X na base de dados também existirá o conjunto Y na mesma transação. Da mesma

forma que o suporte, a confiança possibilita que uma regra não seja podada durante o processamento caso esta medida atinja o limite definido pelo usuário.

$$Conf(X \Rightarrow Y) = \frac{Sup(X \cup Y)}{Sup(X)} \tag{2.2}$$

A Tabela 2.2 exemplifica essa medida. A confiança para regra $A \Rightarrow B$ é de 66% pois, temos duas transações contendo os itens A e B (transações 1 e 2) e três transações contendo A (transações 1, 2 e 4). Com isso, 66% das vezes que A está presente, B também está. A tabela também apresenta as regras $B \Rightarrow C$ e $\{A, B\} \Rightarrow C$, com confiança de 75% e 50%, respectivamente.

Tabela 2.2: Exemplo da medida de confiança.

Id da Transação	**Itens**	**Ex.: Valores de Confiança**
1	{A, B, C}	Total Transações = 5 Conf($A \Rightarrow B$) = $\frac{40\%}{60\%}$ = 66% Conf($B \Rightarrow C$) = $\frac{60\%}{80\%}$ = 75% Conf($\{A, B\} \Rightarrow C$) = $\frac{20\%}{40\%}$ = 50%
2	{A, B, D}	
3	{B, C}	
4	{A, C}	
5	{B, C, D}	

2.1.1 Apriori

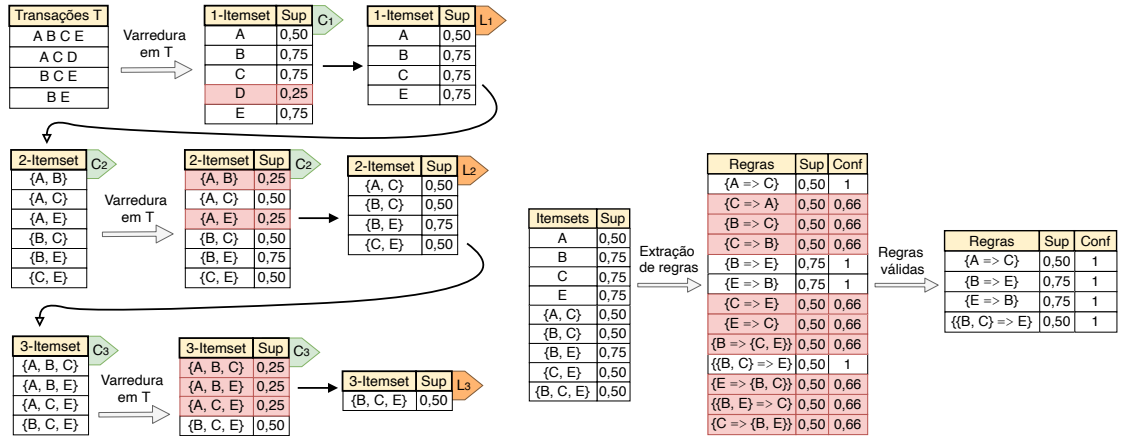
Vasconcelos e Carvalho (2018) afirmam que o algoritmo Apriori é um dos algoritmos mais conhecidos para geração de regras de associação. Este algoritmo emprega busca em largura e utiliza uma abordagem iterativa com a criação de itens candidatos (conjunto de itens que requerem teste para ver se atendem a um determinado requisito). Nesse sentido, o Apriori combina duas etapas durante sua execução (Baldomir, 2017):

> **Junção**: o conjunto de itens candidatos C_k (ou *k-itemset*, onde k é o comprimento do item) é gerado pela combinação de $(k-1)$-*itemset* (ou L_{k-1}) com ele mesmo, ou seja, L_{k-1} com L_{k-1} gera C_k. Na primeira geração deste conjunto todos os itens frequentes são localizados no banco e servem como ponto de partida para este processo. A ideia desta etapa é sustentada pelo princípio de que, se um conjunto de itens frequentes C tem suporte mínimo, todos os seus subconjuntos também terão (Agrawal et al., 1996).
>
> **Poda**: os padrões não frequentes são eliminados do conjunto dos itens frequentes.

Estas etapas ocorrem até não ser mais possível a criação de um *k-itemset*. Para isto, é necessário que o usuário defina valores de suporte mínimo e confiança mínima. Suporte mínimo para encontrar o subconjunto dos itens frequentes e confiança mínima para localizar as regras de associação a partir dos itens frequentes.

A Figura 2.3 ilustra esta ideia. Na Figura 7.2(a), por exemplo, os conjuntos candidatos C_1, C_2 e C_3 são criados tomando como suporte mínimo o valor de 50%. Embora o tamanho de itens combinados aumente em um C_k, para $k+1$, essas combinações ainda fazem parte do conteúdo presente nas transações do banco em questão, ou seja, cada combinação deve pertencer às transações do banco. Além disso, para cada criação de um C_k, uma varredura no banco de dados é realizada para o cálculo de suporte, visto em L_1, L_2 e L_3.

Já na Figura 7.2(b), as regras de associação originadas do mesmo conjunto de itens frequentes possuem o mesmo suporte, mas a confiança não necessariamente será igual. Nesse sentido, as regras consideradas "fortes" ou válidas são selecionadas através dos conjuntos de itens frequentes tendo como fator de corte a confiança, neste exemplo a confiança mínima é 100%.



(a) Geração de *itemsets* para suporte mínimo de 50%.

(b) Extração de regras para confiança mínima de 100%.

Figura 2.3: Exemplo das etapas do algoritmo Apriori.

Mesmo sendo um dos algoritmos mais importantes, o Apriori possui gargalos significativos no desempenho de uma busca. Kotsiantis e Kanellopoulos (2006) expõem que a geração das estruturas de itens candidatos pode se tornar um processo muito complexo e custoso, consumindo muito processamento e memória. Assim, para um grande conjunto de transações pode não haver recursos computacionais suficientes.

Outro gargalo apresentado é a varredura múltipla do banco de dados. Assim, caso o banco de dados possua muitas transações, o tempo para percorrer esses dados aumenta consideravelmente. Vale ressaltar que um valor baixo de suporte mínimo pode implicar em mais combinações e itens nas estruturas de candidatos, ocupando mais espaço em armazenamento e requerendo mais varreduras no conjunto dos dados.

Para solucionar ou minimizar tais gargalos, novos modelos e melhorias de algoritmos foram projetados. Algumas dessas propostas envolvem computação paralela, redução das transações, novas formas de gerar as listas de candidatos ou novas es-

truturas para armazenar estes itens candidatos, Inteligência Computacional, entre outros (Camilo e Silva, 2009).

No entanto, Borgelt (2010) ressalta que embora o desempenho do Apriori clássico seja inferior a algumas das novas abordagens, o mesmo ainda é considerado um algoritmo de mineração de regras de associação importante pois, sua ideia básica de encontrar todos os conjuntos de itens frequentes em um determinado banco de dados é simples e a torna universal, fácil de implementar para quaisquer problemas de mineração de regras de associação.

2.1.2 Eclat

Zaki (2000) propôs o algoritmo Eclat (*Equivalence CLAss Transformation*), que possui formato de dados vertical e emprega busca em profundidade. O foco do Eclat é na etapa mais lenta do Apriori, a descoberta dos *itemsets* frequentes. Fournier-Viger et al. (2017) mostram que a abordagem de busca em profundidade do algoritmo evita manter muitos conjuntos de itens na memória.

Na abordagem horizontal tradicional, cada transação tem um id referente ao conjunto de itens que o compõe. Por outro lado, o formato vertical vincula para cada item uma lista de ids de transações onde este item aparece (Zaki e Gouda, 2003). Assim, ao utilizar uma representação vertical do banco de dados, o Eclat permite obter o suporte dos itens candidatos sem precisar varrer toda a base de dados inúmeras vezes. Nesse contexto, os passos para geração dos *itemsets* frequentes são as seguintes:

- Em primeiro lugar um conjunto vertical é criado baseado nas transações e itens por transação. Para cada item X, uma lista de transações que contém este item é gerada (TID) e denotada como $tid(X)$.
- Uma vez transformados os dados formatados horizontalmente para o formato vertical, a contagem de suporte de um conjunto de itens é simplesmente o comprimento do conjunto TID (ou, em porcentagem, o comprimento da lista TID vezes 100 sobre a quantidade de transações).
- Baseado no mesmo princípio do Apriori, os k-*itemset* frequentes podem ser usados para gerar os $(k+1) - itemset$ candidatos. Dados os itens X e Y, esta geração é realizada a partir da interseção das listas TID correspondentes, ou seja, $tid(\{X, Y\}) = tid(X) \cap tid(Y)$.

O Eclat explora melhor o espaço de busca em comparação com o Apriori. Ele varre o banco de dados uma única vez para criar as listas TID iniciais. A geração de candidatos e contagem de suporte ocorrem diretamente sem percorrer o banco de dados novamente. Assim como o Apriori, o Eclat também realiza seus procedimentos de forma iterativa. Sua execução ocorre até não ser possível gerar conjuntos de itens frequentes ou itens candidatos.

A Figura 2.4 ilustra as ideias descritas anteriormente utilizando suporte mínimo de 50%. Neste exemplo, uma varredura no banco de dados gera a lista de itens candidatos C_1 e logo após permite a criação da L_1 (lista com itens frequentes de comprimento 1 para suporte mínimo de 50%). Em L_1, o item A está contido nas transações de id 1 e 2, logo $tid(A) = [1, 2]$. As interseções das listas ocorrem até não existir mais possibilidades de geração de listas de itens frequentes. Assim, ao final temos L_3 com um único registro correspondendo ao $tid(\{B, C, E\}) = [1, 3]$. O valor de suporte está ligado ao comprimento da lista TID. Por exemplo, $tid(B) = [1, 3, 4]$ possui suporte de 75% pois, o comprimento da TID divido pelo total de transações é 3/4, assim como $tid(\{B, C, E\}) = [1, 3]$ possui suporte de 50% (2/4).

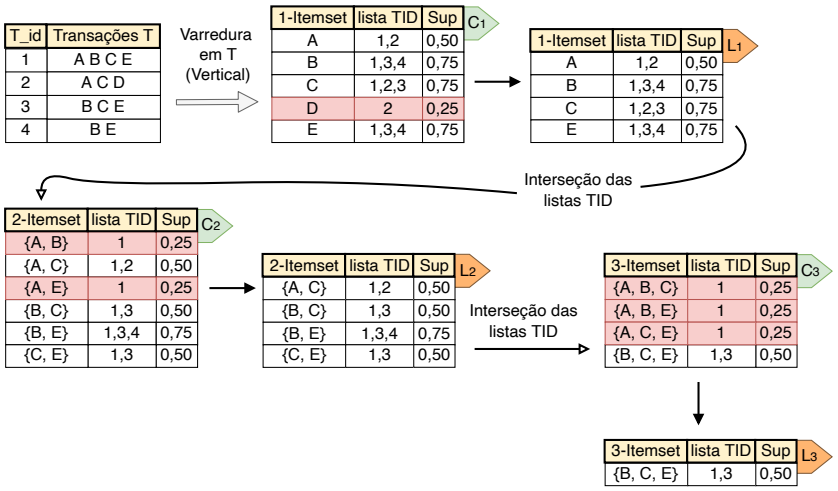


Figura 2.4: Exemplo das etapas do algoritmo Eclat para suporte mínimo de 50%.

Fournier-Viger et al. (2017) afirmam que o Eclat é geralmente muito mais rápido que o Apriori. Este ganho ocorre porque no Eclat não há a necessidade de várias varreduras de banco de dados. Contudo, este algoritmo apresenta algumas desvantagens. Por conta do Eclat também gerar as listas de itens candidatos sem varrer o banco de dados, pode ocorrer do algoritmo gastar tempo considerando combinações de itens que não existem na base de dados (ou seja, conjuntos com suporte 0). Além disso, a formação das listas TID podem consumir muita memória, principalmente quando se trata de conjuntos de dados densos (conjuntos de dados onde todos os itens estão em praticamente todas as transações).

No entanto, existem pesquisas direcionadas para o aperfeiçoamento deste algoritmo. Por exemplo, trabalhos voltados para reduzir o tamanho das listas TID usando estruturas melhoradas chamadas *diffsets* (Zaki e Gouda, 2003), e também, propostas para reduzir o uso de memória e velocidade nas interseções das listas TID a partir da codificação dessas listas em vetores de bits (Zaki e Gouda, 2003; Lucchese et al., 2005).

2.1.3 FP-Growth

Proposto por Han et al. (2000), o FP-Growth (*Frequent Pattern Growth*) é um aprimoramento do algoritmo Apriori. O FP-Growth usa a abordagem de busca em profundidade e com duas varreduras no banco de dados permite produzir os conjuntos de itens frequentes sem geração de conjuntos de itens candidatos. Este algoritmo utiliza uma estratégia de compactação dos conjuntos dos itens frequentes. Essa estratégia se apoia na construção de uma estrutura em árvore, chamada FP-Tree, que mapeia todos os itens na memória, respeitando o limite de suporte mínimo, e mantém as informações de associação entre eles. Assim, tornando o processo de mineração mais eficiente em relação ao Apriori (Zeng et al., 2015).

De modo geral, uma FP-Tree é uma representação compacta do conjunto dos itens. Han et al. (2000) a descreve como uma estrutura em árvore com as seguintes definições:

- É formada por um nó raiz rotulado como "null", um conjunto de subestruturas (subárvores) de prefixo ligadas ao nó raiz e uma tabela auxiliar de itens frequentes;
- Cada nó de uma subárvore contém o nome do item, a frequência do item e uma ligação para outro nó. O nome do item representa o item a que o nó se refere. A frequência do item indica a quantidade de transações que contém o mesmo. E a ligação para um nó serve para se conectar a um outro nó que contém o mesmo nome de item;
- Cada entrada da tabela auxiliar de itens consiste em um campo que identifica um item e um campo para referenciar o primeiro nó da árvore que compreenda o nome do item em questão.

Nesse contexto, Zeng et al. (2015) e Yin et al. (2018) apresentam os seguintes passos executados por este algoritmo: inicialmente, realiza-se uma primeira varredura no banco de dados para capturar a lista dos conjuntos de itens frequentes de comprimento 1 e a sua frequência de aparição nas transações. Em seguida, ordena esta lista de itens frequentes L de forma decrescente baseada no valor da frequência.

Para uma segunda varredura, cada transação pode ser rearranjada com base na lista L, de forma que seus itens estejam em ordem decrescente. Zeng et al. (2015) afirmam que, embora o algoritmo não dependa dessa ordem específica, experimentos mostraram que o mesmo executa mais rápido que usando uma ordem aleatória. Em seguida, para cada transação os itens são inseridos na FP-Tree a partir do nó raiz ("null") com um valor de ocorrência igual a 1. O valor de ocorrência é incrementado toda vez que outro nó com o mesmo item e prefixo precisar ser inserido.

Caso um nó localizado na extremidade da árvore seja alcançado antes do fim da inserção ou se na sequência da subárvore referente à transação não houver mais um nó com o mesmo item que está sendo inserido, um novo nó é criado na FP-Tree. Se este nó não for o primeiro contendo o determinado item, o mesmo é vinculado

a partir do último nó que compreende o item em questão. A tabela auxiliar de itens frequentes também é "alimentada" durante este processo, fornecendo um rápido acesso aos nós.

A Figura 2.5 ilustra os procedimentos descritos anteriormente para construção da FP-Tree. Neste exemplo, um suporte de 50% equivale a uma frequência de 2, pois o total de transações é 4 (tal como a frequência 1 equivale a 25%). Assim, a primeira varredura gera a lista L com os itens ordenados de maneira decrescente pelo seu valor de frequência. Respeitando um suporte de 50%, a segunda varredura cria a tabela auxiliar de itens frequentes e a FP-Tree.

A transação $ABCE$ ou $BCEA$ (considerando a ordem dos itens em L) gera a primeira subárvore com valor de ocorrência 1 em cada item. Posteriormente a transação CA é analisada, e por não existir uma subárvore com o mesmo caminho desde a origem, novos nós são criados e atribuídos ocorrência de valor 1. No entanto, quando a transação BCE é lida, um caminho em comum já existe na FP-Tree, então são incrementados os valores de ocorrência dos nós deste caminho sem a criação de novos nós. Por fim, a transação BE faz com que o primeiro nó contendo B tenha sua ocorrência incrementada e um novo nó para o item E seja criado, visto que não existe uma ligação direta entre estes nós. Além disso, a tabela auxiliar faz referência para os primeiros nós criados de cada item, e em seguida esses nós referenciam um nó semelhante. A ordem de criação dessas subárvores é vista na Figura 2.6.

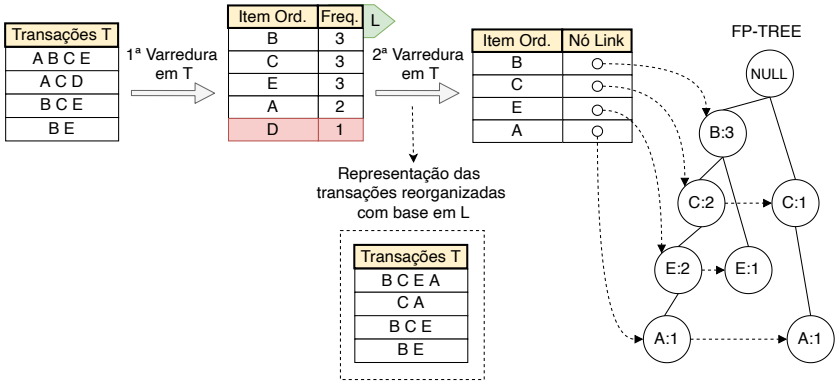


Figura 2.5: Exemplo da geração da FP-Tree do algoritmo FP-Growth para suporte mínimo de 50% (para este exemplo, é o mesmo que frequência igual a 2).

A FP-Tree é construída de forma que os itens com maior frequência/ocorrência fiquem localizados no topo da árvore. Então, com a FP-Tree formada, um processo recursivo é realizado para geração dos padrões frequentes. Zeng et al. (2015) descreve que este processo gera subestruturas que consistem nos caminhos dos prefixos que aparecem com um determinado sufixo, ou seja, a partir de um item são montados os caminhos dos nós que o compreendem até o nó raiz. Essas subestruturas, chamadas de subpadrão condicional, contém os prefixos de um determinado item, juntamente com os valores de ocorrência de cada prefixo.

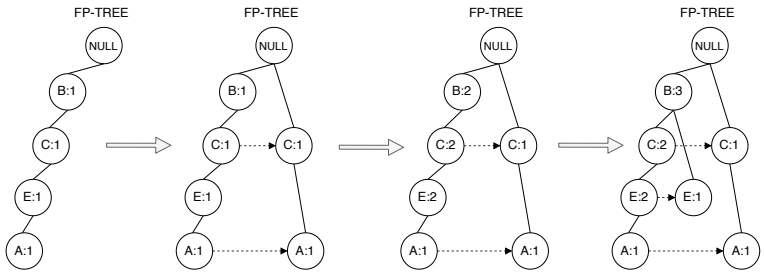


Figura 2.6: Subárvores geradas para formação da FP-Tree.

A partir da criação dos subpadrões condicionais, o algoritmo também constrói uma FP-Tree (condicional) de forma recursiva com base em cada um destes subpadrões. Assim, cada FP-Tree condicional permite gerar padrões frequentes (itens frequentes) ao se relacionar com seu determinado sufixo (item). Apoiado no exemplo da Figura 2.5, a Tabela 2.3 sumariza a etapa final do algoritmo. O item A, por exemplo, fornece dois padrões condicionais $(B : 1, C : 1, E : 1)$ e $(C : 1)$ que quando combinados geram a FP-Tree condicional $C : 2$ para suporte mínimo de 50% (frequência igual a 2). Assim, ao relacionar o sufixo A com sua FP-Tree condicional, encontra-se o padrão frequente AC com suporte de 50%.

Tabela 2.3: Componentes da etapa de geração de padrões frequentes do exemplo da Figura 2.5.

Item	**Subpadrão Condicional**	**FP-Tree Condicional**	**Padrão Frequente**
A	{(B:1, C:1, E:1), (C:1)}	{(C:2)}	AC:2
E	{(B:2, C:2), (B:1)}	{(B:3), (B:2, C:2)}	BE:3, BCE:2, CE:2
C	{(B:2)}	{(B:2)}	BC:2
B	∅	∅	∅

Embora o algoritmo requeira duas varreduras no banco de dados, e não gere conjuntos de itens candidatos, o mesmo precisa criar uma FP-Tree que contém todos os conjuntos de itens. Desta forma, para conjuntos de dados em grande escala, o algoritmo possui deficiências quanto ao uso de recursos computacionais (Yin et al., 2018).

2.1.4 Apriori x Eclat x FP-Growth

Com base nas seções anteriores e apoiado nos estudos de Prithiviraj e Porkodi (2015), a Tabela 2.4 apresenta um comparativo entre os algoritmos Apriori, Eclat e FP-Growth.

De modo geral, esta tabela indica que o algoritmo Apriori é o algoritmo mais simples de entender e implementar em relação aos outros dois, no entanto é o que pode

consumir mais memória e levar mais tempo executando. O algoritmo Eclat possui vantagens em relação aos outros dois algoritmos por conta de sua representação vertical, que normalmente melhora o consumo de memória, entretanto as listas que tornam possíveis esta representação vertical podem ser muito longas e custosas para serem manipuladas. Já o algoritmo FP-Growth é o que realiza menos varreduras no banco de dados e não gera itens candidatos, permitindo que em alguns casos a extração das regras seja mais rápida, contudo é o mais complexo de implementar em relação aos algoritmos Apriori e Eclat.

Tabela 2.4: Tabela comparativa Apriori x Eclat x FP-Growth.

Algoritmo	**Pontos Positivos**	**Pontos Negativos**
Apriori	- Fácil entendimento e implementação; - Gera o itens candidatos a partir dos itens frequentes;	- Múltiplas varreduras na base de dados; - O Tempo para geração dos itens candidatos pode ser muito lento; - Alto custo de memória.
Eclat	- Representação vertical melhora o consumo de memória; - Eficiente para conjuntos pequenos ou médios.	- O uso de listas intermediárias pode consumir muita memória; - Usado apenas para encontrar os itens frequentes.
FP-Growth	- Duas varreduras na base de dados; - Sem geração de itens candidatos; - Compactação de dados com a FP-Tree; - Eficiente para padrões longos.	- Não tão simples a implementação; - Para grandes bases de dados a memória pode não suportar a mineração recursiva na FP-Tree.

2.2 Algoritmo Genético

Pesquisadores e desenvolvedores estão tentando tornar máquinas e softwares mais eficientes e inteligentes. Neste contexto, a Inteligência Computacional (IC) é utilizada no desenvolvimento de soluções eficientes e otimizadas de algoritmos de busca, classificação, regressão, agrupamentos, previsão, entre outros (Tabassum et al., 2014).

O algoritmo genético (AG) é uma das técnicas de busca heurística mais difundidas e avançadas desenvolvidas em IC. Mitchell (1998) afirma que um AG é uma técnica meta-heurística de otimização de busca derivada do processo de evolução natural dos seres vivos.

Neste sentido, um AG é um algoritmo evolucionário baseado em processos que ocorrem em organismos vivos na natureza, como por exemplo, herança, mutação, seleção

e cruzamento. Além disso, o AG requer uma função de aptidão (função fitness) para "guiar" o resultado final, uma vez que emula o conceito de evolução desenvolvendo estocasticamente gerações de populações de soluções (Mitchell, 1998).

Eiben et al. (2015) descrevem os AGs como representações de um sistema evolutivo no qual uma população de representações abstratas (chamadas de cromossomos ou genótipo do genoma) de soluções candidatas (chamadas de indivíduos, criaturas ou fenótipos) "evolui", a cada geração, para uma outra população. No exemplo da Figura 2.7 são ilustrados alguns destes elementos, onde a população possui três cromossomos e cada cromossomo é formado por oito genes.

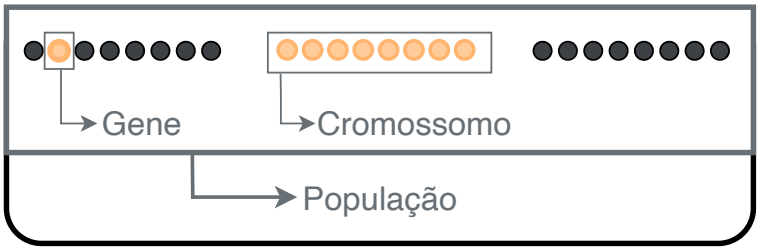


Figura 2.7: População, cromossomos e genes. Fonte: Recriado a partir de Tabassum et al. (2014).

A evolução geralmente começa a partir de uma população de indivíduos gerados aleatoriamente e ocorre em gerações. Em cada geração, a aptidão de cada indivíduo na população é avaliada, vários indivíduos são selecionados da população atual (com base em sua aptidão) e modificados (recombinados e/ou possivelmente mutados aleatoriamente) para formar uma nova população (Qodmanan et al., 2011).

Este algoritmo é aplicável para buscar a solução de alto grau de complexidade que muitas vezes envolve atributos que são grandes, não lineares e discretos por natureza (Qodmanan et al., 2011). Um AG é finalizado quando um número máximo de gerações é atingido ou um nível de aptidão satisfatório é alcançado para a população. No entanto, o AG não garante que o ótimo global possa ser encontrado, mas os resultados geralmente estão próximos a este.

Tabassum et al. (2014) afirmam que AGs são usados em vários campos da biologia, biotecnologia, ciência da computação, engenharia, economia, química, manufatura, matemática, medicina e farmacologia entre outros e possuem inúmeras vantagens e desvantagens, algumas delas listadas na Tabela 2.5.

É necessário analisar o domínio da aplicação para saber se vale a pena empregar um AG. Neste aspecto os AGs possuem pontos positivos e negativos. Como pontos positivos estes algoritmos permitem otimizar uma vasta variedade de problemas, como por exemplo o problema do caixeiro-viajante que pode eventualmente vincular-se a problemas de projeto, programação e entrega de circuitos. Um AG é um mecanismo de resolução de problemas que é capaz de gerar múltiplos resultados apropriados. O AG ainda permite que parâmetros incorretos inicialmente não afetem o resultado

final. Além disso, o AG permite a resolução de problemas com um número mais complexo e finito de parâmetros, permitindo se relacionar facilmente com situações do mundo real, em vez de apenas codificar genes.

Quanto aos pontos negativos, pode-se alegar que um AG não garante que o resultado final será o ótimo global. Além disso, a velocidade de processamento desses algoritmos, geralmente, é baixa. Para um não profissional, o resultado codificado de retorno pode não ser inteligível, pois o AG pode não simular as instruções, mas sim mostrar os cromossomos (codificações) como valores de solução. No geral, um AG fornece resultados baseados em tentativa e falha.

A Tabela 2.5 apresenta simplificadamente os pontos positivos e negativos discutidos anteriormente.

Tabela 2.5: Pontos positivos e negativos de um algoritmo genético (AG).

Pontos Positivos	**Pontos Negativos**
- O AG permite otimizar uma enorme variedade de problemas;	- O AG nem sempre retorna o ótimo global;
- AG é um mecanismo de resolução de problemas e capaz de gerar múltiplos resultados apropriados;	- Normalmente o AG possui tempo de processamento alto;
- O parâmetro incorreto inicialmente não afeta o resultado final;	- Os resultados de um AG podem trazer os dados codificados como cromossomos, assim dificultando o entendimento da informação;
- AG pode resolver problemas com um número mais complexo e finito de parâmetros;	- No geral o AG fornece resultados baseados em tentativa e falha.
- AG pode se relacionar facilmente com a situação do mundo real existente.	

2.3 Consulta Preferencial

Nas consultas tradicionais a busca é realizada consultando o conteúdo armazenado no banco de dados e retornando o mesmo resultado independentemente do usuário que está realizando a operação (Medeiros et al., 2015). Além disso, Rocha-Junior (2013) afirma que essas consultas têm restrições rígidas que fazem com que estes resultados sejam um conjunto de dados completo ou nada. Assim, a quantidade de informação disponível e a taxa de mudança podem ocultar a solução desejada. Isto mostra a necessidade de um mecanismo que identifique as melhores opções dentre todos os cenários possíveis.

Considerando a Tabela 2.6, um usuário interessado nos hotéis com diárias baratas pode tentar localizar um hotel com preço abaixo de quarenta reais. No entanto, o resultado será vazio, visto que ele não conhece o conteúdo do banco de dados. Na tentativa de encontrar algo, este usuário pode alterar o preço desta busca para valores abaixo de cem reais. Entretanto, a resposta será quase todos os dados do conjunto em questão. Ao utilizar consultas preferenciais, este usuário pode buscar hotéis pelos menores preços limitando a quantidade de informações desejadas. Por exemplo, solicitando apenas os 3 hotéis mais baratos.

Tabela 2.6: Conjunto de dados que trazem preço da diária e distância até a praia de hotéis. Tabela baseada de Rocha-Junior (2013).

	Hotel	**Preço (R$)**	**Distância (m)**
t_1	A	300	50
t_2	B	70	100
t_3	C	100	100
t_4	D	80	200
t_5	E	100	300
t_6	F	50	800

Nesse contexto, Chomicki (2003) define que consultas preferenciais são usadas para filtragem e extração de informações. Além de reduzir o volume de dados apresentados ao usuário, elas também permitem rastrear os perfis dos utilizadores e formular políticas para melhorar e automatizar a tomada de decisões. A depender do perfil de busca do usuário, estas consultas podem ser classificadas como qualitativas ou quantitativas (Rocha-Junior, 2013).

Nas consultas qualitativas, as preferências entre pares de objetos (tuplas) no conjunto de dados são especificadas diretamente (Rocha-Junior, 2013). Assim, dadas as tuplas t_1 e t_2 no conjunto de dados P, a tupla t_1 é preferida sobre t_2 se uma fórmula de preferência $f(t_1, t_2)$ for válida. Esta fórmula é uma preferência binária entre quaisquer $t \in P$, e pode ser expressa usando operações lógicas.

Nas consultas quantitativas, as preferências são especificadas indiretamente com o uso de funções de pontuação que associam uma pontuação numérica a cada tupla da resposta da consulta (Chomicki, 2003). Desta forma, a pontuação gerada para cada tupla indica sua importância e é utilizada para classificar estes dados. Assim, uma tupla t_1 é preferida a uma tupla t_2 se a pontuação de t_1 for maior que a de t_2.

Diversos são os estudos a respeito de técnicas e métodos para operações envolvendo as preferências dos usuários. Duas estratégias populares para lidar com esses tipos de consultas são o Top-k (Seção 2.3.1) e Skyline (Seção 2.3.2).

2.3.1 Top-k

Kalyvas e Tzouramanis (2017) afirmam que consultas Top-k retornam as melhores k tuplas num conjunto de dados baseados em um função de preferência. Esta função de preferência, ou função de pontuação, atua na avaliação dos objetos de acordo com suas características (como preço da diária de um hotel e distância até a praia, por exemplo). Geralmente estes objetos são avaliados por vários critérios de pontuação que contribuem para a pontuação final do objeto (Ilyas et al., 2008). Portanto, uma função de pontuação é, em geral, definida como uma agregação sobre pontuações parciais. De uma maneira mais formal, Vlachou et al. (2010) apresenta a Definição 2.3.1 para o Top-k.

Definição 2.3.1. ***Consulta Top-k:*** *dado um espaço de dados D definido por um conjunto de dimensões $\{d_1, ..., d_d\}$ e um conjunto de dados S em D com cardinalidade $|S|$, um ponto $p \in S$ pode ser representado como $p = \{p[1], ..., p[d]\}$ onde $p[i]$ é um valor na dimensão d_i. Desta forma, dado um número inteiro positivo k e um vetor de ponderação w, definido pelo usuário, o conjunto de resultados $TOP_k(w)$ da consulta Top-k é um conjunto de pontos tais que $TOP_k(w) \subseteq S$, $|TOP_k(w)| = k$ e $\forall p_1, p_2 : p_1 \in TOP_k(w)$, $p_2 \in S - TOP_k(w)$ mantém as funções de pontuação $f_w(p_1) \leq f_w(p_2)$.*

Para a Tabela 2.6, é possível definir, por exemplo, a função de pontuação $f(t) = \textit{preço} + \textit{distância}$, $k = 3$ e tendo como objetivo minimizar o valor retornado pela função de pontuação. Desta forma, as pontuações para os hotéis A, B, C, D, E e F são respectivamente, $f(t_1) = 350$, $f(t_2) = 170$, $f(t_3) = 200$, $f(t_4) = 280$, $f(t_5) = 400$ e $f(t_6) = 850$. Portanto, as 3 melhores tuplas (que tiveram o menor score obtido pela função de pontuação) são t_2, t_3 e t_4.

2.3.2 Skyline

Borzsony et al. (2001) propuseram o Skyline como um operador para consultas preferenciais em banco de dados. Eles definem o Skyline como os pontos que não são dominados por nenhum outro ponto em um número arbitrário de dimensões. Um ponto domina outro ponto se for tão bom ou melhor em todas as dimensões e melhor em pelo menos uma dimensão. Esta é exatamente a noção de fronteira de Pareto. Em outras palavras, as consultas pelo Skyline retornam todas as tuplas em uma relação que possuem o conjunto de seus atributos não dominados pelos atributos das outras tuplas. De uma maneira mais formal, Rocha-Junior (2013) apresenta a Definição 2.3.2 para o Skyline.

Definição 2.3.2. ***Consulta Skyline.*** *A consulta Skyline retorna o conjunto de pontos $SKY \subseteq P$ que não são dominados por nenhum outro ponto em P. Um ponto $p \in P$ domina outro ponto $q \in P$, denotado como $p \prec q$, se em toda dimensão i, $p[i] \leq q[i]$; e em pelo menos uma dimensão j, $p[j] < q[j]$.*

Considerando a Tabela 2.6, cada tupla t é representada como um ponto no espaço

compreendendo as características do hotel (Rocha-Junior, 2013). A dimensão $\boldsymbol{x}$ reflete o preço da diária do hotel, enquanto a dimensão $\boldsymbol{y}$ corresponde à distância do hotel à praia. Neste caso, pretende-se minimizar todas as dimensões ($\boldsymbol{x}$ e $\boldsymbol{y}$), logo, um hotel com menor preço e menor distância para a praia é preferível em relação a um hotel com maior preço e mais distante da praia.

A Figura 2.8 ilustra as relações de dominância entre estes hotéis. Nenhum hotel domina o hotel F no atributo preço, assim como nenhum outro domina o hotel A no atributo distância à praia. O hotel B por ser melhor que F em relação à distância e melhor que A em relação ao preço, não é dominado pelos mesmos. Por outro lado, o hotel B domina C, D e E. Assim, as tuplas t_1, t_2 e t_6 (hotéis A, B e F, respectivamente) estão situados na fronteira de Pareto (neste contexto também chamado de Skyline) e representam a resposta do algoritmo.

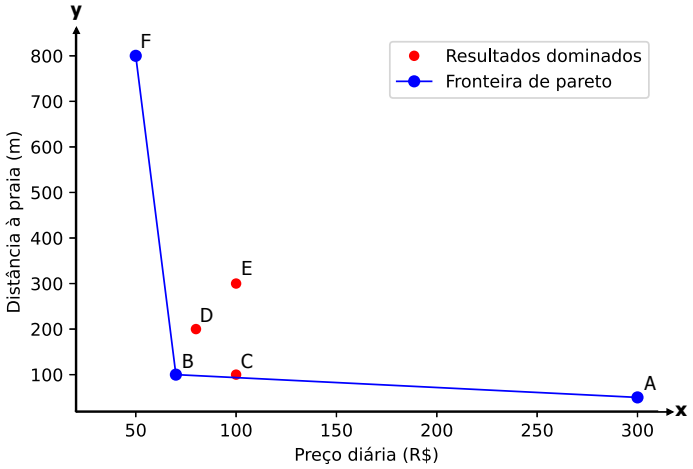


Figura 2.8: Skyline dos hotéis da Tabela 2.6.

Em contraste com as consultas Top-k, as consultas Skyline são direcionadas para uma abordagem mais compreensível aos humanos (Kalyvas e Tzouramanis, 2017; Vlachou et al., 2022). Enquanto que o Top-k utiliza funções e critérios de classificação específicos, o Skyline assume que cada usuário tem uma série de preferências sobre os atributos dos dados (Kalyvas e Tzouramanis, 2017; Vlachou et al., 2022). Essas preferências indicam o que o usuário gosta e não gosta ("Gosto mais de praia que cidade" ou "Prefiro viajar para uma ilha que viajar para uma fazenda"). Todas as preferências são consideradas equivalentes e auxiliam a descartar os objetos que não são preferidos por ninguém. E com isso, fornecem um subconjunto de dados conciso e interessante com base nos gostos de todos os usuários.

2.4 Amostragem

De acordo com Correa (2003), a estatística é uma parte da matemática que fornece métodos para coleta, organização, descrição, análise e interpretação de dados com o

objetivo de tornarem as tomadas de decisões mais eficientes. Neste contexto, uma técnica de amostragem é a parte da teoria estatística que define as etapas para o planejamento e técnicas de estimação (Correa, 2003). Segundo Scheaffer et al. (2011) e Correa (2003), os seguintes termos são extensamente utilizados nesta área de estudo.

Indivíduo (ou Elemento): é um objeto no qual a medição é realizada;

População: é a coleção de elementos ao qual se deseja fazer uma inferência;

Amostra: é parcela selecionada da totalidade de observações abrangidas pela população, através da qual se faz inferência sobre as características da população;

Unidade Amostral: é o indivíduo da população amostral sobre o qual a medida de interesse é observada. As Unidades amostrais são coleções não sobrepostas de elementos da população que cobrem toda a população;

Parâmetro: é uma medida numérica que descreve uma característica de uma população. Esta característica, a princípio, é fixa e desconhecida;

Estimativa: é uma medida numérica que descreve uma característica de uma amostra. Este valor é utilizado para se estimar o valor desconhecido do parâmetro.

Segundo Correa (2003), na realização de um estudo nem sempre é possível examinar todos os elementos de uma população de interesse. Neste contexto, o objetivo dos planejamentos amostrais é fazer inferências sobre uma população a partir de informações contidas em uma amostra selecionada dessa população, assim cada observação, ou elemento, retirado da população contém uma certa quantidade de informações sobre o parâmetro ou parâmetros populacionais de interesse (Scheaffer et al., 2011). Correa (2003) afirma que a inferência estatística fornece elementos para generalizar de maneira "segura", as conclusões obtidas da amostra para a população, a Figura 2.9 apresenta uma visão geral deste processo de inferência.

A quantidade de informação obtida na amostra depende do número de unidades amostrais e da quantidade de variação nos dados. Assim, uma amostra tem que ser representativa, ou seja, a amostra deve possuir as mesmas características básicas da população, logo, cuidados devem ser tomados para que os resultados não sejam distorcidos. Tais cuidados giram em torno da definição cuidadosa da população de interesse, as características que serão pesquisadas/estudadas e o método de amostragem selecionado (Correa, 2003). Em relação aos métodos para a composição de amostras, Correa (2003) afirma que em geral, são usados os métodos probabilísticos e não probabilísticos (intencional).

Os métodos de amostragem probabilística permitem selecionar aleatoriamente um pequeno grupo de elementos que representarão uma população. Normalmente, estes métodos exigem que cada elemento possua a mesma probabilidade de ser selecionada, com isso, se a população possui tamanho N, a probabilidade de seleção para cada

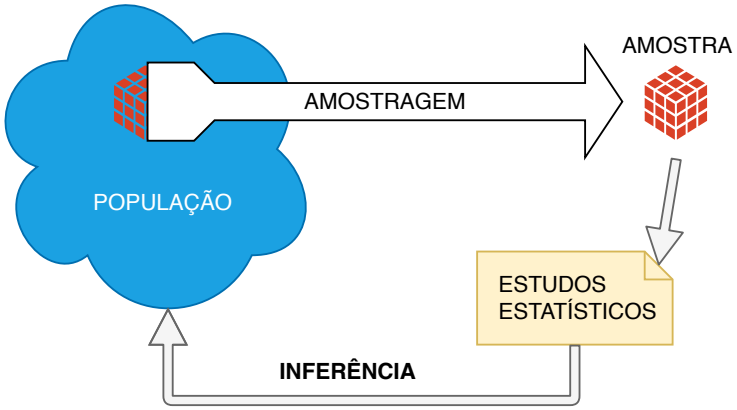


Figura 2.9: Visão geral do processo de inferência estatística.

elemento será $\frac{1}{N}$. Correa (2003) declara que generalizações dos resultados só podem ser realizadas com base em amostragens probabilísticas. Conforme Scheaffer et al. (2011), com a aleatoriedade adequada na amostragem, é possível fazer afirmações como "Nossa estimativa é imparcial e estamos 95% confiantes de que nossa estimativa estará dentro de 2 pontos percentuais da proporção real". Correa (2003); Scheaffer et al. (2011) apresentam os seguintes métodos probabilísticos como os principais utilizados:

> **Amostragem Aleatória Simples**: neste método a amostra de tamanho n é selecionada ao acaso dentre os N elementos da população (Correa, 2003);
>
> **Amostragem Aleatória Estratificada**: para este método a população é subdividida em grupos menores, que geralmente não se sobrepõem e em seguida aplica-se a amostragem aleatória simples (Correa, 2003);
>
> **Amostragem Sistemática**: neste método os elementos da população são escolhidos de acordo com suas classificações em uma determinada função (Correa, 2003);
>
> **Amostragem por Conglomerado (ou por *Clusters*)**: este método realiza agrupamentos dos elementos e aplica a amostragem aleatória simples, onde as unidades amostrais são representadas por estes conglomerados de elementos (Correa, 2003).

Os métodos de amostragem não probabilística permitem a escolha deliberada de elementos que irão compor as amostras. Enquanto que os métodos probabilísticos possuem uma certa quantidade de aleatoriedade embutida para que o viés ou imparcialidade do estimador possa ser estabelecido, os métodos não probabilísticos, por sua vez, não seguem um modelo aleatório, não possibilitando a generalização dos resultados das pesquisas sobre as amostras para a população, ou seja, a representatividade da população por estas amostras não é garantida (Correa, 2003; Scheaffer et al., 2011). Os seguintes métodos são apresentados por Correa (2003) como sendo

os principais métodos não probabilísticos:

Amostragem Acidental (ou de Conveniência): neste método as amostras são formadas por elementos que vão surgindo. Elementos estes que são possíveis de se obter até completar o número de elementos da amostra (Correa, 2003). Ex.: Pesquisa de opinião, onde os entrevistados são acidentalmente escolhidos;

Amostragem Intencional: neste método as amostras são formadas por elementos escolhidos intencionalmente pelo estimador através de um determinado critério (Correa, 2003). Ex.: Pesquisa de opinião, onde os entrevistados são escolhidos por conta de suas vestimentas;

Amostragem por Quotas (ou Cotas): este método tende a ter um rigor mais elevado dentro das demais técnicas não probabilísticas. O estimador deve definir classes populacionais em termos de propriedades relevantes, em seguida são determinadas as proporções da população para cada uma dessas classes (Correa, 2003). Ex.: Pesquisas eleitorais ou análise de mercado.

2.4.1 Determinando o Tamanho da Amostra

O tamanho da amostra deve ser cuidadosamente definido para que seja adequado realizar conclusões válidas e generalizadas. A definição do tamanho amostral adequado requer informações específicas sobre os problemas investigados na população em estudo (Singh e Masuku, 2014). Em geral, o tamanho da amostra utilizado em um estudo é determinado com base no custo da coleta de dados e com base no poder estatístico suficiente. Em estudos avançados, pode haver vários tamanhos de amostra diferentes envolvidos no estudo (Singh e Masuku, 2014). Tamanhos de amostra maiores geralmente levam a uma maior precisão ao estimar parâmetros desconhecidos.

Singh e Masuku (2014) afirmam que os tamanhos da amostra podem ser escolhidos de três maneiras diferentes:

Base de Custo: são incluídos os itens prontamente disponíveis ou convenientes. A escolha de tamanhos pequenos de amostra, embora às vezes necessária, pode resultar em amplos intervalos de confiança ou riscos de erros no teste de hipóteses estatísticas;

Base de Variância: usa-se uma variância alvo para uma estimativa a ser derivada da amostra eventualmente obtida;

Base de Poder Estatístico: usa-se um alvo para o poder de um teste estatístico a ser aplicado assim que a amostra for coletada. Os tamanhos das amostras são julgados com base na qualidade das estimativas resultantes, o tamanho da amostra pode ser avaliado com base no poder de um teste de hipótese.

Além disto, Singh e Masuku (2014) afirmam que o tamanho da amostra, no geral, depende de cinco parâmetros:

Tabela 2.7: Guia de tamanho da amostra (n) para níveis de precisão (e) de ±5% e ±10%, confiança da amostra de 95% e proporção (p) de 50%.

Tamanho População	**Tamanho Amostra (n)**	
	e=±5%	**e=**±10%
500	222	83
1.000	286	91
2.000	333	95
3.000	353	97
4.000	364	98
5.000	370	98
7.000	378	99
9.000	383	99
10.000	385	99
15.000	390	99
20.000	392	100
25.000	394	100
50.000	397	100
100.000	398	100
$>$ 100.000	400	100

- Diferença mínima esperada (ou tamanho do efeito): a definição deste parâmetro é subjetiva e é baseada no julgamento e experiência com o problema que está sendo investigado;
- Variabilidade estimada: este parâmetro é representado pelo desvio padrão esperado dentro de cada grupo de comparação. Se os dados preliminares não estiverem disponíveis, este parâmetro pode ser estimado com base na experiência subjetiva ou uma faixa de valores pode ser assumida;
- Poder estatístico desejado: à medida que este parâmetro aumenta, o tamanho da amostra aumenta. Embora o alto poder seja sempre desejável, há uma compensação com o número de indivíduos que podem ser investigados de maneira viável, dada a quantidade de tempo e recursos disponíveis para realizar a pesquisa ou estudo;
- Critério de significância (valor P): este parâmetro é o valor máximo de P para o qual uma diferença deve ser considerada estatisticamente significativa. À medida que o critério de significância diminui, o tamanho da amostra necessário para detectar a diferença mínima aumenta. Normalmente este parâmetro é definido como 5%;
- Análise estatística uni ou bicaudal: a escolha do tipo de análise depende da direção da diferença entre os grupos de comparação. Em geral, a análise unicaudal exige um tamanho amostral menor para detecção da diferença mínima que a análise bicaudal.

Singh e Masuku (2014) apresenta a Tabela 2.7 como um guia com tamanhos pré-

Tabela 2.8: Resumo dos trabalhos relacionados.

Referência	**Método**	**Limitações**
Qodmanan et al. (2011)	Algoritmo genético.	Pode gerar regras redundantes e inválidas.
Moslehi e Haeri (2020)	Algoritmo genético para regras quantitativas.	Pode gerar regras redundantes e inválidas.
Huang e Kao (2004)	Lógica Fuzzy com derivação de suporte e confiança.	Pode gerar regras inválidas ao usuário e necessidade de análise semântica.
Djenouri e Comuzzi (2017)	Algoritmos genéticos e enxames de partículas.	Necessita do parâmetro de suporte mínimo.
Koh et al. (2014)	Algoritmos gulosos e buscas Top-k reversas.	Faz muitas comparações (elevado consumo computacional).
Mohammed et al. (2015)	Semântica dos atributos, noção de dominância e preferências de usuário.	Considera análise semântica.
Dahbi et al. (2016)	Relação de dominância e classifica regras em múltiplos critérios.	Excessivo custo computacional.
Nguyen et al. (2018)	Top-k com duas etapas de filtragem dos itens candidatos.	Pode gerar regras redundantes e necessita do limiar de confiança.
Hirate et al. (2004)	Combina Top-k e FP-Growth.	Regras extraídas são entregues ao usuário em blocos de itens.
Riondato e Upfal (2015)	Top-k com amostragem progressiva.	Pode gerar regras falsas e alta complexidade na condição de parada para bases de dados grandes.
Kameya e Sato (2012)	Combina Top-k e FP-Growth com busca *branch-and-bound*.	Exclusivo para padrões discriminativos.

definidos de amostras para um determinada população e conjuntos de parâmetros. Sendo os parâmetros n o tamanho da amostra, p a proporção da população que tem uma determinada característica (quando não se tem este dado, o mesmo é definido para 50%) e, por fim, e como o nível de precisão da amostra.

2.5 Trabalhos Relacionados

Nesta seção são apresentados alguns trabalhos que contém elementos semelhantes à proposta apresentada nesta pesquisa. Também são apontadas algumas lacunas dos mesmos. A Tabela 2.8 apresenta brevemente sobre estes trabalhos.

Qodmanan et al. (2011) propõem o algoritmo ARMMGA baseado no estudo de algoritmos genéticos. O ARMMGA foca na mineração de regras de associação sem

utilizar os parâmetros de suporte mínimo e confiança mínima. O mesmo utiliza uma função de aptidão (que pode ser alterada de acordo com o utilizador) que calcula a correlação entre suporte e confiança para cada conjunto de itens. Com isso, o algoritmo seleciona as regras a partir da média destas correlações ao longo de cada população gerada no algoritmo. Entretanto, nada garante que o resultado final contém todas as melhores regras ou as regras válidas para o utilizador. Assim como, existe a possibilidade do resultado final trazer itens redundantes.

Moslehi e Haeri (2020) também propõem um algoritmo genético. Apoiado por uma função de aptidão específica, ele permite recuperar regras de associação sem obrigatoriamente passar os limites de suporte e confiança como parâmetro. Este algoritmo apresenta resultados mais vantajosos que seus antecedentes e similares no que diz respeito à quantidade de regras geradas com maior medida de suporte e confiança dos itens. No entanto, também é possível que o resultado final não traga todas as melhores regras para o utilizador. Além disso, o foco deste algoritmo é em encontrar regras de associação exclusivamente quantitativas (pelo menos um termo da regra de associação deve envolver um atributo numérico).

Huang e Kao (2004) descrevem um modelo baseado em Lógica Fuzzy para minerar regras de associação a partir da derivação de suporte mínimo e confiança mínima em multiníveis de hierarquia. Nesta configuração os itens são representados de forma hierárquica e o utilizador realiza a busca passando como parâmetro um termo linguístico que representa o suporte mínimo para níveis mais altos da cadeia hierárquica. Desta forma, o usuário consegue encontrar um valor de suporte mínimo mais rapidamente. Contudo, presume-se que a base de regras linguísticas possa ser bem definida para o banco de dados e traduzidas para os termos corretos durante as derivações. Além disso, a derivação dos valores de suporte e confiança podem gerar regras esdrúxulas em certas ocasiões, visto que o modelo tenta recuperar, também, regras potencialmente interessantes.

Djenouri e Comuzzi (2017) descrevem os algoritmos GA-Apriori e PSO-Apriori para mineração de itens frequentes. GA-Apriori e PSO-Apriori usam algoritmos genéticos e otimização de enxame de partículas, respectivamente. Eles exploram o espaço dos conjuntos de itens combinando a propriedade recursiva de conjuntos de itens frequentes e o processo de pesquisa estocástica. Embora estes algoritmos apresentem tempos de execução melhores que algoritmos como Eclat e Apriori, seus resultados são aproximados. Além disso, necessitam receber como parâmetro um valor de suporte mínimo.

Koh et al. (2014) introduzem dois algoritmos, SimpleGrendy e FastGrendy, baseados na abordagem de algoritmos gulosos e consultas Top-k reversas. Estes algoritmos consideram um conjunto de clientes com preferências diferentes para características de itens, e a partir das funções de pontuação de cada cliente, recuperam os k melhores itens de uma lista de itens candidatos de forma que o número total de potenciais clientes seja maximizado. Enquanto que o SimpleGrendy atua de forma incremental para encontrar as soluções, o FastGrendy emprega Skyline para reduzir o espaço de

busca. No entanto, estes algoritmos foram propostos para estimar o impacto de um potencial produto no mercado encontrando uma solução aproximada. Além disso, muitas comparações podem ser geradas durante o processamento dos dados e com isso prejudicar o tempo de resposta.

Mohammed et al. (2015) introduziram o $S_{EM}MDP_{REF}$. Este algoritmo se baseia no valor semântico dos atributos (ou seja, é o significado atribuído aos itens dentro de um determinado contexto), na noção de dominância entre as regras de associação e na preferência do usuário. Esta abordagem não favorece nem exclui quaisquer medidas, a exemplo de suporte e confiança, mas utiliza um coeficiente semântico para avaliar se uma regra deve ser ou não desconsiderada. Contudo, apenas considerar o significado semântico ou a semelhança entre as regras pode gerar resultados equivocados para certos conjuntos de dados.

Dahbi et al. (2016) descrevem uma abordagem baseada na relação de dominância para seleção de regras de associação. O método proposto avalia e classifica todas as regras a partir de múltiplos critérios, além de permitir ao usuário escolher quantas regras devem ser retornadas. No entanto, atribuir uma pontuação e então ranquear um alto conjunto de regras gera um excessivo custo computacional. Além disso, este algoritmo também considera o valor semântico dos atributos na hora da classificação.

Nguyen et al. (2018) propõem um algoritmo eficiente para explorar as Top-k regras de associação. O algoritmo ETARM obtém as k melhores regras de associação reduzindo o espaço de busca ao aplicar duas etapas de filtragem de regras candidatas. Este algoritmo não necessita do parâmetro de suporte mínimo. Entretanto, ainda requer a confiança mínima. O algoritmo também permite a geração de regras redundantes ao final da busca.

Hirate et al. (2004) descrevem o TF^{2}P-growth. Este algoritmo combina as estratégias do FP-Growth e Top-k para encontrar os padrões frequentes sem utilizar nenhum limiar de entrada. No entanto, o TF^{2}P-growth é uma abordagem exaustiva e aumenta o valor do suporte mínimo lentamente. Além disso, os padrões extraídos são fornecidos aos usuários após cada nc-padrões, onde nc é o tamanho do bloco, assim o usuário precisa analisar blocos separados de padrões a cada momento.

Kameya e Sato (2012) propõem um algoritmo para mineração Top-k de padrões discriminativos altamente relevantes para uma classe de interesse. O RP-growth conduz uma pesquisa *branch-and-bound* usando limites superiores anti-monotônicos de pontuações de relevância como F-score e χ^2. Os autores também utilizam a noção de fraqueza entre padrões e uma estratégia de poda agressiva adicional baseada nesta fraqueza. No entanto, realizar buscas utilizando classes de interesse não garante que os resultados sejam relevantes ao usuário, além disso o RP-growth se destina à mineração de padrões discriminativos, ou seja, recuperar padrões fortemente ligados a uma classe e que sejam fracas nas demais classes.

Riondato e Upfal (2015) introduzem um algoritmo para extrair uma aproximação de alta qualidade dos Top-k conjuntos frequentes de itens a partir de amostras

aleatórias de um conjunto de dados. Este algoritmo emprega uma amostragem progressiva, com uma condição de parada baseada em limites na média empírica de *Rademacher*. Os autores afirmam que este algoritmo é bem-sucedido em parar a busca rapidamente nas iterações iniciais e permite extrair uma aproximação da coleção de itens frequentes. No entanto, os autores também mencionam a necessidade de estudar mais sobre a relação entre a complexidade da condição de parada e o tamanho das amostras geradas.

Capítulo 3

Metodologia

"Concentre todos seus pensamentos na tarefa que está realizando. Os raios de sol não queimam até que sejam colocados em foco."
– Alexander Graham Bell

O estudo em questão realiza uma pesquisa exploratória quantitativa. Logo, parte-se da observação e estudo de um problema de mineração de dados para a construção de uma solução que permita diminuir os esforços de configuração dos parâmetros de uma determinada consulta (De Oliveira, 2011).

Este estudo possui natureza aplicada, pois fornece um mecanismo útil para ser usado em aplicações práticas. Assim, são realizados experimentos controlados utilizando dois tipos de bases de dados (reais e sintéticas). As bases de dados reais servem para avaliar o comportamento da solução proposta em um cenário real, enquanto que as bases de dados sintéticas servem para avaliar o comportamento da solução proposta em cenários hipotéticos, garantindo maior confiabilidade aos resultados obtidos.

As seguintes subseções apresentam de forma mais detalhada os passos e procedimentos utilizados para a construção desta pesquisa. Na Seção 3.1 é apresentada uma visão geral da construção da solução. A Seção 3.2 apresenta as características dos bancos de dados selecionados para o estudo experimental. A Seção 3.3 contém uma descrição de como os dados são processados e, por fim, a Seção 3.4 possui uma descrição das análises realizadas pós-processamento.

3.1 Design do Experimento

Os procedimentos seguidos para a construção do algoritmo proposto são ilustrados na Figura 3.1. A etapa de coleta e construção de base de dados representa a aquisição

das bases de dados reais e construção das bases de dados sintéticas, levando em consideração os requisitos mínimos para os experimentos (vide Seção 3.2). As bases de dados reais requerem uma etapa de pré-processamento, que serve para retirar registros indesejados, bem como homogeneizar o formato dos dados para que possam ser processados, Seção 3.3.

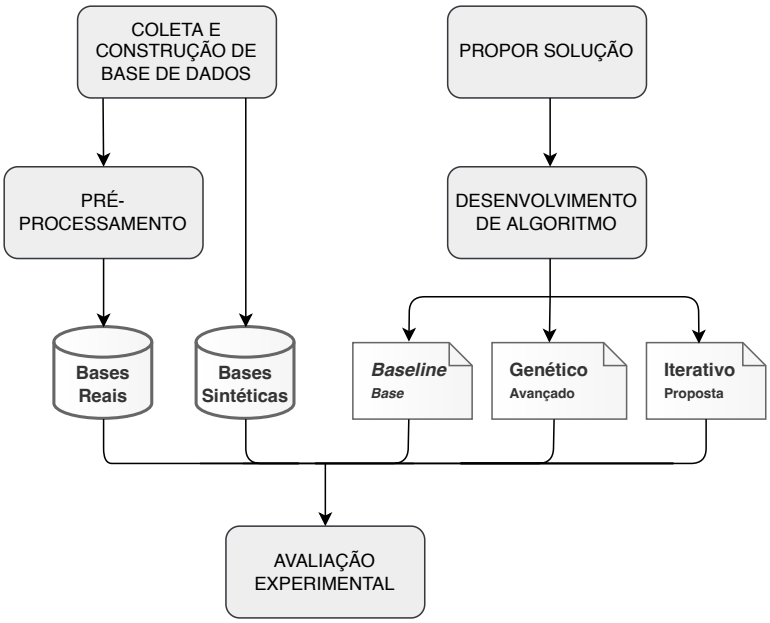


Figura 3.1: Visão geral do experimento.

Paralelamente ao processo de coleta do banco de dados, tem-se a etapa de propor solução. Nessa fase, como o nome já diz, um ou mais algoritmos são planejados e propostos a fim de sanar ou mitigar alguns dos desafios abordados nesta pesquisa.

Com uma proposta de solução melhor definida, tem-se início a fase de desenvolvimento. Nesta fase são implementados algoritmos base e avançados. Os algoritmos base (*Baseline*) são desenvolvidos a partir do estado da arte e são utilizados para servir de comparação com os algoritmos mais avançados (identificados como algoritmo Genético na Figura 3.1) e a solução proposta (nomeado por algoritmo Iterativo na Figura 3.1).

Por fim, são realizados experimentos controlados utilizando os artefatos adquiridos até então. Na fase de avaliação experimental, tanto o *Baseline* quanto os algoritmos Genético e Iterativo são submetidos a casos de testes, criados durante esta etapa e relacionados a cada tipo de base de dados (real e sintética). Em seguida os resultados destes testes são analisados e avaliados com o objetivo de quantificar os ganhos obtidos usando a solução proposta.

3.2 Base de Dados

Para este experimento o banco de dados deve possuir transações envolvendo dois ou mais itens. Este requisito é importante pois permite descobrir relacionamentos ou padrões frequentes entre os elementos.

Esta propriedade pode ser vista na Tabela 3.1. Esta tabela contém cinco transações que representam compras de produtos como leite, pão, manteiga e café. A primeira transação (id_T igual a 1), por exemplo, caracteriza uma compra contendo os produtos leite e pão, sem conter manteiga ou café. Salientando que para o problema que pretendemos solucionar a base de dados não necessita estar exclusivamente relacionada a sistemas de supermercado.

Tabela 3.1: Exemplo de base de dados com 4 itens e 5 transações.

id_T	**leite**	**pão**	**manteiga**	**café**
1	1	1	0	0
2	0	1	1	0
3	0	0	0	1
4	1	1	1	0
5	0	1	0	0

No mínimo, duas bases de dados distintas devem ser utilizadas no estudo experimental. Uma contendo valores reais, ou seja, adquirida de alguma empresa atuante no mercado. E outra criada sinteticamente para avaliar cenários específicos e que possam afetar a solução proposta. As bases reais são utilizadas para obter resultados mais realísticos e mais próximos aos que ocorrem no mercado, enquanto que as bases sintéticas são utilizadas para avaliar as soluções propostas em diversos cenários.

3.3 Processamento dos Dados

Neste trabalho, dois algoritmos são utilizados para orientar a pesquisa e as soluções propostas. Um deles é a adaptação do algoritmo Apriori, visto na Tabela 2.4, e funciona como *Baseline*. A escolha do mesmo leva em consideração a facilidade em adaptá-lo para o problema proposto. Também é levado em conta o tempo base de execução e uso de memória desses algoritmos. E o outro algoritmo é uma adaptação do algoritmo genético proposto por Qodmanan et al. (2011). Este outro algoritmo foi selecionado porque ele também não necessita dos parâmetros iniciais de suporte e confiança para obter regras de associação, sendo a referência mais relacionada com o trabalho proposto nesta pesquisa.

Para todos os algoritmos, conjuntos de casos de teste são implementados e executados. Cada conjunto de teste consiste em um par único de informações, a *base de*

dados e *critérios de busca*. O termo *critérios de busca* representa os parâmetros necessários para a execução de cada algoritmo e o mesmo teste é usado em todos os algoritmos.

3.4 Avaliação e Análise dos Resultados

Para analisar os algoritmos é necessário modificar algumas variáveis e observar os efeitos causados. Neste contexto existem dois tipos de variáveis: aquelas que são modificadas e as que são analisadas.

As variáveis modificadas pelos controladores do experimento são o tipo da base de dados e o bloco de execução do algoritmo. Cada alteração influencia aspectos da execução do algoritmo e permite que certas características dos algoritmos e comparações entre os mesmos possam ser realizadas. Os itens frequentes variam conforme o conjunto de dados utilizado e o tamanho deste conjunto. Assim, ao variar os itens da lista a seguir é possível observar a formação dos conjuntos frequentes de itens e regras de associação com seus valores de suporte e confiança, bem como o tempo necessário para computar tais informações.

Tipo base de dados: esta variável é alterada ao mudar o tipo de base de dados utilizada no algoritmo, ou seja, alterando o conjunto de itens utilizados (contexto). Vale ressaltar que ao alternar entre as bases de dados, o tamanho das mesmas também pode sofrer alteração.

Blocos de Execução: são parâmetros ou condições da base de dados para cada execução do algoritmo. Para o *Baseline*, cada bloco de execução utiliza um valor de suporte mínimo diferente (95%, 80%, 70%, 60%, 50%, 40%, 30%, 20%, 10%, 5%). Para o Iterativo, cada bloco de execução utiliza uma amostra diferente. Os demais algoritmos não possuem essas configurações.

Nesse contexto, analisar o impacto dessas variáveis é importante para o entendimento desta pesquisa. Assim, ao modificar as variáveis acima, pretende-se observar o impacto nos seguintes itens:

Memória: quantidade de memória utilizada por cada algoritmo (em Megabyte);

Tempo de resposta: tempo total de execução do algoritmo, entre a solicitação da consulta e a resposta do mesmo (em segundos);

Regras válidas: regras produzidas pelo algoritmo e que são válidas, ou seja, que realmente existem no base de dados;

Qualidade das regras: pontuação produzida pela função de aptidão (fitnesss) para as regras produzidas pelo algoritmo.

Capítulo 4

Estratégias e Algoritmos Desenvolvidos

Neste capítulo são apresentados e descritos os algoritmos e as estratégias desenvolvidas durante toda a pesquisa. Dentre estes algoritmos e estratégias temos o *Baseline* que é fundamentado no algoritmo Apriori, utilizando a biblioteca *mlxtend*[1]. Entretanto, para gerar as regras de associação usando o *Baseline* é necessário inserir os parâmetros de suporte e confiança mínima.

Além do *Baseline* (Seção 4.1), tem-se um algoritmo Genético que funciona como uma base para observação do comportamento dos dados sobre aspectos da inteligência computacional e também representa a solução mais similar ao algoritmo proposto neste estudo (Seção 4.2). E por fim a estratégia proposta nesta pesquisa (Seção 4.3).

4.1 *Baseline*

Para a escolha do algoritmo *Baseline* foram levados em consideração os fatos de que o mesmo deve estar amplamente presente em trabalhos relacionados e também possuir significativos materiais de apoio. Além disso, deve-se considerar os recursos de *hardware* disponíveis para esta pesquisa.

Neste contexto, o algoritmo *Baseline* se fundamenta no algoritmo Apriori, vide Seção 2.1.1. Portanto, ele usa uma abordagem onde os subconjuntos frequentes de itens são desdobrados em outros conjuntos, chamados de itens candidatos, e então estes conjuntos de candidatos são validados utilizando o suporte mínimo. Os conjuntos frequentes de itens resultantes podem ser usados para determinar as regras de associação que destacam certas tendências no banco de dados, ressaltando que as regras encontradas também precisam ser validadas por um limiar, a confiança mínima. Neste sentido, o que diferente o *Baseline* do Apriori é que após a geração das regras de associação, as mesmas são submetidas a um método Top-k (utilizando

[1] http://rasbt.github.io/mlxtend/

como função fitness a Equação 4.1), que por sua vez retorna as regras resultantes do *Baseline*. O Algoritmo 1 traz uma visão geral do *Baseline* utilizado.

Algorithm 1: *Baseline*

Input: D: conjunto dos dados
Input: ϵ: limiar de suporte
Input: $conf\_min$: limiar de confiança
Input: k: quantidade de regras para o Top-k
Result: Top-k das regras

```
1  C_1 ← countItems(D);
2  L_1 ← prune(C_1, ε);
3  i ← 2;
4  L ← ∅;
5  topk ← ∅;
6  while ! isEmpty(L_{i-1}) do
7      C_i ← aprioriGen(L_{i-1});
8      foreach d ∈ D do
9          foreach c ∈ C_i do
10             if c in d then
11                 c.count ← c.count + 1;
12             end
13         end
14     end
15     L_i ← prune(C_i, ε);
16     L ← L + L_i;
17     i ← i + 1;
18 end
19 rules ← genRules(L, conf_min);
20 topk ← calcTopK(topk, k, rules);
21 return topk;
```

Nas linhas $1-2$ do Algoritmo 1 o primeiro conjunto de itens candidatos de tamanho um C_1 é localizado a partir de uma varredura no banco de dados D e em seguida o conjunto de itens frequentes, também de tamanho um, L_1 é encontrado a partir da remoção dos itens indesejados utilizando o limiar de suporte ϵ.

Na linha 7 do Algoritmo 1, o método de geração dos itens candidatos é chamado. Este método utiliza os itens frequentes gerados anteriormente (L_{i-1}) para, entre si, combinar e formar um novo conjunto de itens candidatos de tamanho i, C_i.

Com um novo conjunto de itens candidatos formado C_i, é então necessário validar se cada uma das combinações que existem neste conjunto estão presentes na base de dados D. Além disso, para cada combinação existente na base, sua frequência (quantidade de vezes em que aparece) é incrementada a fim de ser usada na fase de remoção de itens indesejados. Estes procedimentos estão descritos nas linhas $8-14$.

Na linha 15 do Algoritmo 1 um novo conjunto de itens frequentes L_i é formado a partir da remoção dos itens indesejados (itens com suporte inferior ao suporte mínimo) em C_i. Estas etapas se repetem até que o conjunto L_i seja vazio (linha 6 do Algoritmo 1) e assim, o conjunto resultante com todos os itens frequentes de todos os tamanhos é gerado pelo algoritmo, linha 16 do Algoritmo 1.

Por fim, regras de associação são geradas utilizando o limiar de confiança, linha 19 do Algoritmo 1. Em seguida, estas regras são passadas como parâmetro para um método Top-k (que utiliza a Equação 4.1) e então as k regras com o score mais alto são devolvidas pela função, linhas $20 - 21$ do Algoritmo 1.

4.2 Genético

O algoritmo Genético utilizado neste estudo corresponde ao algoritmo Genético proposto por Qodmanan et al. (2011). O Algoritmo 2 apresenta uma visão geral da implementação do mesmo.

Algorithm 2: Genético

```
   Input: D: conjunto dos dados
   Input: SP: taxa de seleção
   Input: CP: taxa de cruzamento
   Input: MP: taxa de mutação
   Result: Melhor população
 1 pop ← initialize(D);
 2 best_pop ← pop;
 3 while ! terminate(pop) do
 4 |   parents_pop ← select(pop, SP);
 5 |   children_pop ← crossover(parents_pop, CP);
 6 |   children_pop ← mutate(children_pop, MP);
 7 |   pop ← selectNextPop(parents_pop, children_pop);
 8 |   if isBest(pop, best_pop) then
 9 |   |   best_pop ← pop;
10 |   end
11 end
12 return best_pop;
```

Seguindo a estratégia de Qodmanan et al. (2011), o Algoritmo 2 possui as etapas de codificação (linha 1), inicialização (linhas $1 - 2$), seleção (linha 4), cruzamento (linha 5), mutação (linha 6) e decisão (linhas 3 e $7 - 10$). Além disso este algoritmo necessita de alguns parâmetros probabilísticos com valores de 0 à 1 (de 0 à 100%), são eles: probabilidade da seleção - *sp* (*selection probability*); probabilidade de cruzamento - *cp* (*crossover probability*); e probabilidade de mutação - *mp* (*mutation probability*). Estes parâmetros são utilizados nas etapas de seleção, cruzamento e mutação respectivamente.

Este algoritmo também utiliza uma função que ajuda na busca das melhores regras. Esta função chamada função fitness, Equação 4.1, é responsável por atribuir uma pontuação a cada cromossomo levando em consideração as medidas de suporte e confiança da regra.

$$f(X \Rightarrow Y) = \frac{(1 + Sup(X \cup Y))^2}{1 + Sup(X)} \tag{4.1}$$

Qodmanan et al. (2011) apresenta a Figura 4.1 como sendo o gráfico dos valores da Equação 4.1, indicando que a função vai de $0,5$ a 2 e sugerindo que esta função cresce conforme o crescimento do suporte e/ou confiança, ou seja, uma função que relaciona positivamente ambas as medidas de interesse.

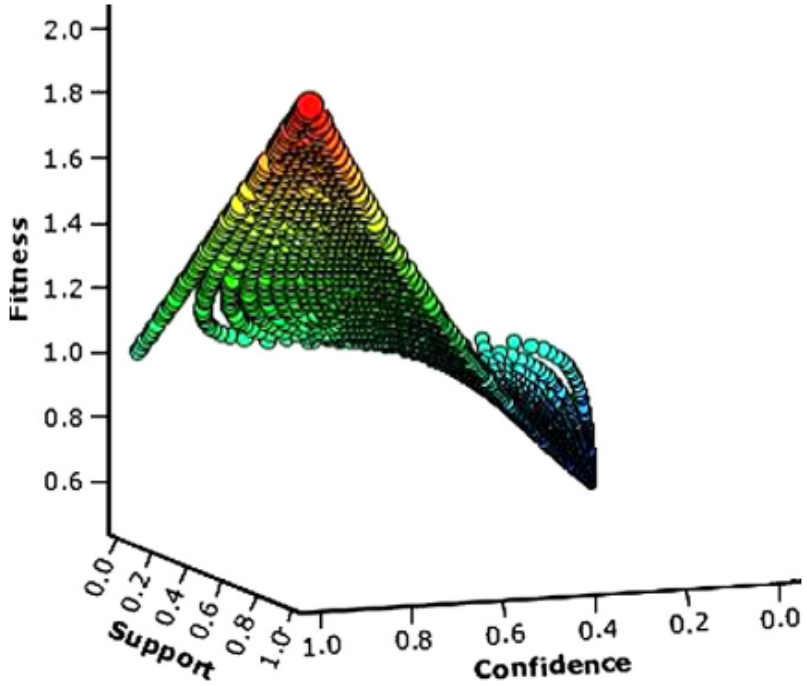


Figura 4.1: Valores da função fitness com base em suporte e confiança. Fonte: Qodmanan et al. (2011).

4.2.1 Codificação e Inicialização

As etapas de codificação e inicialização compõem a fase inicial do algoritmo. Nesta etapa, a população inicial de tamanho n (e por enquanto a melhor população) é formada aleatoriamente por conjuntos de itens existentes na base de dados de forma que seja possível construir regras de associação a partir deles. Vale ressaltar que a geração aleatória dessas regras não garante que as regras formadas serão válidas em

termos de suporte e confiança, ou seja, uma ou mais regras podem ter suporte de valor zero.

Os conjuntos de itens que formam a regra $X \Rightarrow Y$ são codificados seguindo o modelo da Figura 4.2, onde o antecedente da regra é representado pelo conjunto $X = \{A_1, ..., A_j\}$, o consequente da regra é representado pelo conjunto $X = \{A_{j+1}, ..., A_k\}$, a variável j indica o divisor entre antecedente e consequente de uma regra e a variável k indica o tamanho da regra. Essa codificação forma o cromossomo e a população é preenchida por uma certa quantidade desses cromossomos. Estes processos estão ilustrados nas linhas $1 - 2$ do Algoritmo 2.

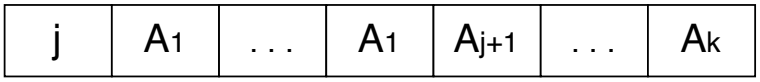


Figura 4.2: Codificação de uma k-regra. Fonte: Recriado a partir de Qodmanan et al. (2011).

4.2.2 Seleção

Na fase de seleção, a técnica utilizada foi por ordenação ou *ranked-based* baseada numa distribuição linear. O algoritmo ordena a população utilizando uma função fitness (Equação 4.1) e então gera um valor para cada cromossomo baseado em sua posição. Após isso utiliza a variável *sp* para selecionar, ou não, conjuntos de cromossomos que serão chamados de população pai. O processo de seleção leva em consideração esta ordenação e permite que os cromossomos mais "valiosos" (topo da lista) tenham mais chances de serem escolhidos, assim como evita a pressão evolutiva exercida por um super indivíduo (Bäck et al., 2018). Esta etapa está inclusa na linha 4 do Algoritmo 2.

4.2.3 Cruzamento

Após obter a população pai, o algoritmo passa para a etapa de cruzamento. Nesta etapa os cromossomos também são ordenados pela função fitness e então "cruzados" com uma probabilidade *cp* através da técnica *Order-1*. Este cruzamento permite que "novos" cromossomos sejam criados a partir da combinação de dois cromossomos da população pai, assim, gerando uma nova população, chamada de filha. Vale ressaltar o cuidado de durante o cruzamento não permitir itens replicados num mesmo cromossomo, o que resultaria em uma regra de associação inválida. Esta etapa está inclusa na linha 5 do Algoritmo 2.

4.2.4 Mutação

Na fase de mutação, a população filha é ordenada pela função fitness e então cada cromossomo passa por um processo de mutação considerando a probabilidade *mp* e a posição do cromossomo. Neste processo, tanto o suporte quanto a confiança

da regra representa pelo cromossomo podem ser alterados, pois o indicador j pode ser substituído aleatoriamente por outro número (alterando o antecedente e consequente da regra e consequentemente podendo alterar seu suporte), tal como um item presente em $A_1, ...A_k$ pode ser alterado para outro item existente da base de dados (assim gerando uma nova regra e possivelmente um outro valor de suporte e confiança). Neste processo o cuidado sobre itens replicados também existe. Esta etapa está inclusa na linha 6 do Algoritmo 2.

4.2.5 Decisão

Após a etapa de mutação, o algoritmo realiza um procedimento de seleção entre a população pai e a população filha. De modo geral, este procedimento consiste em aleatoriamente recuperar cinco elementos da população pai e cinco da população filha e em seguida agrupá-los para então selecionar o elemento com maior score para compor a nova população; isto se repete até a população resultante atingir o tamanho definido previamente. Este método permite que ótimos cromossomos sejam escolhidos, assim como abre possibilidades para que cromossomos não tão bons também sejam capturados, com isso evitando ótimos locais. Esta etapa está inclusa na linha 7 do Algoritmo 2.

Em seguida tem-se uma população de tamanho n que pode ser comparada com a população anterior (melhor população). Desta forma, a média e a variância da função fitness dos cromossomos de cada população são consideradas a fim de verificar qual é a melhor população, aquela que possui maior média e menor variância. Assim, caso uma nova população seja considerada melhor, ela é salva e passa a ser a base da próxima geração (linhas $8 - 10$ do Algoritmo 2). Este ciclo se repete até que um número máximo de gerações seja atingido e/ou a diferença entre o melhor e pior cromossomo da melhor população esteja dentro de um limite chamado ϵ, este controle está incluso na linha 3 do Algoritmo 2.

Por fim, na linha 12 do Algoritmo 2, a melhor população é retornada. Vale salientar que esta população não necessariamente consiste no conjunto dos melhores cromossomos (codificação das regras) presentes na base de dados, mas sim o melhor conjunto que o algoritmo conseguiu encontrar durante os procedimentos descritos anteriormente. Por exemplo, na população final podem existir regras que são inexistentes na base de dados, no entanto esta população ainda foi a melhor formada no algoritmo.

4.3 Iterativo - Amostragem

Zaki et al. (1997) afirmam que a amostragem aplicada ao processo de mineração de dados é capaz de reduzir os custos computacionais, reduzindo drasticamente o número de transações a serem consideradas. Ao utilizar técnicas de amostragem probabilística na lógica de algoritmos de mineração de conjuntos frequentes de itens

e suas regras de associação é possível reduzir o espaço de busca (trabalhando sobre as amostras) e em seguida realizar inferências sobre o conjunto total de itens (população). Neste contexto, o algoritmo explicado nesta seção tem como objetivo encontrar as Top-k regras de associação ao reduzir seu campo de busca através da amostragem e realizar uma inferência a respeito do suporte mínimo que deve ser utilizado na busca operando sobre toda base de dados. Assim, permitindo que o processo de mineração de dados seja livre de parâmetros externos (fornecidos pelo usuário buscador) e se torne mais vantajoso que realizar diversas consultas, alterando os parâmetros de suporte mínimo e confiança mínima, no que diz respeito ao tempo gasto para se encontrar as regras desejadas. Este algoritmo consiste em duas etapas: a primeira etapa chamada de amostragem e a segunda de busca global.

Na primeira etapa ocorre o processo de amostragem da base de dados, onde um valor n que satisfaça a Equação 4.2 (Israel, 1992) é encontrado e então, de forma aleatória, n registros são selecionados da base de dados. Com esse conjunto reduzido de registros um sub algoritmo é executado para se obter um outro conjunto contendo as regras de associação, que por conseguinte permitirá que uma destas regras seja utilizada para calcular o suporte mínimo e/ou confiança mínima utilizada na segunda etapa (busca global). Na primeira etapa (amostragem), este sub algoritmo possui as seguintes variações:

> **Variação com Redução da Transação (ou Variação com Log)**: na primeira etapa o conjunto reduzido de registros é formado pela amostragem da base de dados e posteriormente existe uma fase de amostragem dos itens nas transações com o objetivo de diminuir o tamanho das mesmas, logo todas as regras possíveis para esta amostra são formadas, Seção 4.3.1;
>
> **Variação com Top-k**: o conjunto reduzido de registros é formado pela amostragem da base de dados e então as regras de associação são formadas utilizando o método Top-k, Seção 4.3.2;
>
> **Variação com Skyline**: o conjunto reduzido de registros é formado pela amostragem da base de dados e então as regras de associação são formadas utilizando o método Skyline, Seção 4.3.3;

A solução proposta é representada pelo Algoritmo 3. A primeira etapa, descrita nas linhas $1-3$ desta representação, utiliza a Equação 4.2 para fornecer um valor n que representará o tamanho da amostra. As variáveis N, Z, p, q e E representam, respectivamente, o tamanho da base de dados, o valor que especifica o nível de confiança da amostra, a proporção da população que tem uma determinada característica, a proporção da população que não tem uma determinada característica e a precisão das proporções amostrais.

Dado isto, o processo de seleção das amostras estão presentes nas linhas $1-2$, onde o método *sizeSample* utiliza a Equação 4.2 para encontrar o tamanho da amostra e o método *calculateSample* realiza a ação de amostragem propriamente dita. O método de amostragem utilizado é o aleatório simples (vide Seção 2.4). A linha 3

do Algoritmo 3 é onde o sub algoritmo (ou uma de suas variações, mencionadas anteriormente) é executado.

$$n = \frac{N \cdot Z^2 \cdot p \cdot q}{E^2 \cdot (N-1) + Z^2 \cdot p \cdot q} \tag{4.2}$$

Algorithm 3: Iterativo

Input: D: conjunto de dados
Input: k: quantidade de regras para o Top-k
Result: Top-k das regras
`/* Primeira Parte - Amostragem */`
1 $n \leftarrow sizeSample(D)$
2 $sample \leftarrow calculateSample(D, n)$
3 $rule \leftarrow subIterativo(sample)$; `// comum ou alguma das variações`
`/* Segunda Parte - Busca Global */`
4 $sup\_min, conf\_min \leftarrow findRealMetrics(D, rule)$;
5 $frequent\_items \leftarrow frequentItems(D, sup\_min)$;
6 $candidate\_items \leftarrow candidates(D, frequent\_items)$;
7 $candidates\_aux \leftarrow candidate\_items$;
8 $topk \leftarrow \emptyset$;
9 **while** *!* $isEmpty(candidates\_aux)$ **do**
10 $rules \leftarrow genRules(candidates\_aux, conf\_min)$;
11 $topk \leftarrow calcTopK(topk, k, rules)$;
12 $candidates\_aux \leftarrow candidates(D, candidates\_aux, sup\_min)$;
13 **end**
14 **return** $topk$;

Na etapa de busca global (linhas $4-14$ do Algoritmo 3), ou segunda etapa, a regra selecionada pela primeira etapa (o método de escolha de regra depende da variação utilizada) é mapeada na base de dados completa a fim de se obter seu valor "real" de suporte e/ou confiança, linha 4. Logo, este valor de suporte é utilizado como suporte mínimo para obtenção dos conjuntos frequentes, linhas 5 e 12. Em seguida, é utilizado o método Top-k para mineração das regras de associação, linha 11.

Nesta etapa utiliza-se uma função de aptidão ou função fitness (Equação 4.1) para pontuar as regras geradas durante as etapas de construção dos itens candidatos. Desta forma a cada ciclo as melhores regras são armazenadas.

Após encontrar o valor real de suporte e confiança da regra utilizando o método $findRealMetrics$ (linha 4), o valor de suporte é usado como suporte mínimo no decorrer do algoritmo. Em sequência são identificados, na base de dados, todos os itens unitários juntamente com suas frequências (suporte) e então monta-se o conjunto de itens candidatos. Esta etapa está presente nas linhas $5-7$ do Algoritmo 3.

Após obter o conjunto de itens candidatos, o algoritmo gera todas as regras a partir deste conjunto e calcula o Top-k das mesmas apoiada na Equação 4.1, linhas $10-11$. Em seguida um novo conjunto de itens candidatos é gerado, linha 12.

Estes processos descritos anteriormente ocorrem até que não existam itens candidatos. Esta validação está presente na linha 9 do algoritmo, função $isEmpty$. Por fim, o Top-k resultante das regras é devolvido pela função, este retorno está presente na linha 14.

A estratégia base (sem variação) para o sub algoritmo utilizado na primeira etapa (amostragem) consiste em localizar todos os conjuntos frequentes de itens sem a utilização de nenhum limiar e em seguida gerar todas as regras possíveis para então ordená-las pelo valor de suporte e então selecionar a regra de tendência central deste conjunto. Uma visão geral deste procedimento é descrito no Algoritmo 4.

Algorithm 4: SubIterativo_Comum

Input: *sample*: conjunto dos dados da amostra
Result: uma regra de associação da amostra

```
1  frequent_items ← frequentItems(sample);
2  candidate_items ← candidates(sample, frequent_items);
3  candidates_aux ← candidate_items;
4  frequent_items ← ∅;
5  while ! isEmpty(candidates_aux) do
6  |   candidates_aux ← candidates(sample, candidates_aux);
7  |   frequent_items ← frequent_items ∪ candidates_aux;
8  end
9  rules ← genRules(frequent_items);
10 rule ← orderedMedianRule(rules);
11 return rule;
```

Inicialmente é formada uma lista de itens e sua respectiva frequência na amostra. Em seguida é gerado o conjunto de itens candidatos de tamanho dois também com suas respectivas frequências, linhas $1-3$. A partir disto ocorre uma repetição responsável por gerar novos conjuntos de itens candidatos e adicioná-lo no conjunto de itens frequentes resultante, linhas $6-7$ do Algoritmo 4. Esta repetição ocorre até não existirem itens na lista de itens candidatos (o método $isEmpty$ verifica o tamanho desta lista), linha 5 do Algoritmo 4.

4.3.1 Variação com Redução da Transação

Esta variação, além de realizar um processo de amostragem na base de dados, realiza uma amostragem nos itens das transações utilizando uma função com distribuição normal para gerar números aleatórios que serão responsáveis pela identificação das posições dos itens selecionados dentro de uma transação e uma função para definir o tamanho da "nova" transação.

Desta forma, o tamanho destas transações também é reduzido, o que acelera o processo de seleção da regra de associação da primeira etapa. A Equação 4.3 permite que o tamanho das transações mesmo após as amostragens não seja elevado. Esta equação foi montada com o objetivo de que transações com um número alto de itens (longas) se transformem em transações com um número reduzido de itens (curtas), enquanto que transações curtas se mantenham curtas.

$$nt = \log_2 |t| + 2 \tag{4.3}$$

Algorithm 5: SubIterativo_L

Input: *sample*: conjunto dos dados da amostra
Result: uma regra de associação da amostra

```
1 foreach t ∈ sample do
2 |   t ← samplingLog(t);
3 end
4 sample ← frequentItems(sample);
5 frequent_items ← frequentItems(sample);
6 ...
7 return rule;
```

Na linha 2 do algoritmo, a função *samplingLog* aplica a Equação 4.3 modificando as transações presentes na amostra e com isso reduzindo o total de itens inclusos em cada transação. Em seguida a lógica segue como no Algoritmo 4.

4.3.2 Variação com Top-k

Nesta variação, após ser realizado o processo de amostragem na base de dados, todas as regras possíveis de gerar a partir desta amostra são então montadas e processadas pelo método Top-k (também utilizando a Equação 4.1 como função fitness), linhas $2-3$. Com as k melhores regras da amostra recuperadas, é localizada a última regra deste conjunto para servir como retorno da função, linhas $4-5$. O uso do método Top-k e o retorno de seu k-ésimo item nesta variação tem o intuito de possibilitar que na segunda etapa do algoritmo (busca global) o mesmo seja capaz de localizar pelo menos k regras relevantes ao usuário, visto que o suporte mínimo utilizado nesta segunda etapa refere-se a uma regra que possui k-1 regras com score mais alto.

Algorithm 6: SubIterativo_T

Input: *sample*: conjunto dos dados da amostra
Result: uma regra de associação da amostra

```
1 ...
2 rules ← genRules(frequent_items);
3 topk ← calcTopK(rules);
4 rule ← lastItem(topk);
5 return rule;
```

O início deste algoritmo segue a mesma lógica presente no Algoritmo 4, com inclusão do Top-k e a alteração na forma de capturar a regra para o retorno da função.

4.3.3 Variação com Skyline

Nesta variação é encontrado o Skyline das regras presentes na amostra. Desta forma é possível, de maneira rápida, construir um conjunto de regras que não são dominadas por nenhuma outra, e então selecionar a regra de tendência central deste conjunto. O fato de se escolher a regra de tendência central do conjunto diz respeito às posições que representam pontos representativos (Vlachou et al., 2022). Assim, a inferência sobre o suporte mínimo utilizado na etapa global ocorre de maneira mais confiável quando comparado à utilização de outras posições do Skyline.

Algorithm 7: SubIterativo_S

Input: *sample*: conjunto dos dados da amostra
Result: uma regra de associação da amostra

1 ...
2 $rules \leftarrow genRules(frequent\_items)$;
3 $skyline \leftarrow calcSkyline(rules)$;
4 $rule \leftarrow median(skyline)$;
5 **return** $rule$;

A lógica e passos deste algoritmo segue quase que semelhante ao Algoritmo 4, diferenciando-se na geração do conjunto Skyline, função *calcSkyline* na linha 3, e obtenção do ponto representativo deste conjunto, função *median* na linha 4.

Capítulo 5

Estudo Experimental

Este capítulo levanta as estratégias e componentes utilizados para a realização deste estudo. A Seção 5.1 contém as bases de dados utilizadas durante o experimento. A Seção 5.2 possui informações sobre o processo de avaliação dos algoritmos. Finalmente, a Seção 5.3 contém as características e especificações do *hardware* utilizado para execução dos algoritmos.

5.1 Bases de Dados

Para o desenvolvimento da estratégia em questão foram utilizadas as seguintes bases de dados reais:

- `Purchases`: uma base de dado da empresa Sanar Saúde[1], contendo transações que se referem aos itens comprados por cliente;
- `Visited`: também uma base de dados fornecida pela empresa Sanar Saúde, no entanto, contendo produtos visitados por clientes;
- `FVMushroom`: uma modificação do conjunto de dados referentes a cogumelos extraídos do guia de campo da Audubon Society para Cogumelos Norte-Americanos. O conjunto original (*Mushroom*) foi recuperado do repositório *fimi*[2];
- `São Roque`: uma base de dados contendo itens comprados num determinado período no supermercado São Roque[3] na cidade de Feira de Santana, estado da Bahia, Brasil.

Três das quatro bases de dados utilizadas possuem informações reais. Além disso, os conjuntos de dados foram escolhidos por causa da quantidade de itens que eles

[1]https://www.sanarsaude.com/
[2]http://fimi.uantwerpen.be/data/
[3]http://www.gruposaoroque.com/

dispõem, indo de aproximadamente 1.631 (conjunto dos dados do Sanar para itens comprados) a cerca de 50 mil (conjunto dos dados do São Roque) de transações. Além disso, todas as bases possuem os requisitos mínimos para utilização de mineração de regras de associação.

Em relação ao conjunto de dados `FVMushroom`, ela foi utilizada como uma base sintética, ou seja, foram feitas alterações no conteúdo da base original *Mushroom* com o objetivo de observar o comportamento dos algoritmos.

Por questões de segurança e facilidade na execução dos algoritmos, todos os itens presentes nas transações estão codificados em valores numéricos, por exemplo, o item *café* é representado pelo número 4 e o sal é representado por 115.

Purchases

Esta base de dados reais, também fornecida pela empresa Sanar, foi extraída entre o período de setembro de 2020 a fevereiro de 2021. Para este conjunto de dados, cada transação representa os produtos adquiridos por clientes neste período e contém um total de 1.631 transações com 315 itens distintos.

Visited

Este conjunto de dados reais foi extraído entre o período de setembro de 2020 a fevereiro de 2021 e fornecido pela empresa Sanar Saúde. Os dados representam listas de produtos visitados (quando usuários clicam no link do produto) por clientes neste período e contém um total de 27.098 transações com 942 itens distintos.

FVMushroom

Este conjunto de dados sintético inclui descrições de amostras hipotéticas correspondentes a 23 espécies de cogumelos *gilled* na Família *Agaricus* e *Lepiota* extraídas do guia de campo da sociedade *audobon*. Estes cogumelos são descritos em termos de características físicas e classificação de venenosa ou comestível. No entanto, diferentemente da base original, a base `FVMushroom` contém apenas as 5 primeiras características de um cogumelo (num total de 22) por registro. Além disso, esta base de dados possui um total de 8.124 registros (mesmo tamanho da original), com 24 itens distintos.

São Roque

Este conjunto de dados reais foi extraído e fornecido pelo grupo São Roque. Os dados representam compras realizadas neste supermercado, contendo um total de 50.000 transações com 24.622 itens distintos.

Tabela 5.1: Parâmetros utilizados para executar o algoritmo Genético.

Parâmetro	**Valor**
sp	0,95
cp	0,85
mb	0,01
ruleLen	4
alfa	0,01
minGen	25
maxGen	1.000
sizePop (k)	10

5.2 Avaliação

O comportamento dos algoritmos é observado a partir de alterações de algumas variáveis. Além do uso de parâmetros (a exemplo dos que são utilizados nos algoritmos genéticos, Tabelas 5.1), os algoritmos também utilizam uma função fitness (Equação 4.1). A solução proposta utiliza a Equação 4.2 para definição do tamanho da amostra. Ambas as equações mencionadas podem ser substituídas por outras funções, no entanto, para esta avaliação apenas estas duas foram utilizadas.

Os algoritmos genéticos precisam de alguns parâmetros para seu funcionamento, a Tabela 5.1 contém os parâmetros utilizados nesta pesquisa. Deste modo, *sp*, *cp* e *mp* correspondem respectivamente às probabilidades de seleção, cruzamento e mutação de elementos no algoritmo. O parâmetro *ruleLen* representa o tamanho da regra (utilizado na codificação do cromossomo); $alfa$ representa o erro máximo aceitável entre o melhor e pior cromossomo de uma população. Já os atributos *minGen*, *maxGen* e *sizePop* correspondem respectivamente ao mínimo e máximo permitido para o ciclo de gerações e o tamanho da população (que representa a quantidade final de itens desejados, ou seja, k).

O algoritmo proposto pelos autores desta pesquisa requer apenas o parâmetro k, que representa o número de regras desejadas pelo usuário.

Este estudo experimental modifica o valor de duas variáveis: tipo da base de dados e os blocos de execução (Tabela 5.2). Existem quatro tipos diferentes de bases de dados: `Purchases`, `Visited`, `FVMushroom` e `São Roque`. Em relação ao experimento com os blocos de execução, optou-se por executar cada algoritmo dez vezes sobre certas circunstâncias, mantendo o mesmo valor de regras desejadas ($k = 10$). Tais circunstâncias estão resumidas na Tabela 5.3.

Ao variar o tipo (juntamente com o tamanho) das bases de dados, os experimentos permitem observar, em cada execução, o impacto sobre: o custo de memória, tempo

Tabela 5.2: Variáveis modificadas durante o experimento.

Variável	Valores
Tipo Database	`Purchases` (tam: 1.631) `Visited` (tam: 27.098) `FVMushroom` (tam: 8.124) `São Roque` (tam: 50.000)
Quantidade de Blocos de Execução	10

Tabela 5.3: Parâmetros modificados a cada execução do algoritmo.

Algoritmo	Circunstância
Baseline	Valores diferentes de suporte mínimo em cada execução (95%, 80%, 70%, 60%, 50%, 40%, 30%, 20%, 10% e 5%).
Genético	Nenhuma condição.
Iterativo	Cada execução utiliza uma amostra diferente, empregando o mesmo método de seleção de amostragem.

de resposta dos algoritmos, quantidade de regras válidas e a qualidade dessas regras.

5.3 *Hardware* Utilizado

O experimento foi realizado em um *MacBook Air (13-inch, 2017)* com processador *Dual-Core Intel Core i5 1,8 GHz* e memória de *8 GB 1600 MHz DDR3*. Este computador possui placa gráfica *Intel HD Graphics 6000 1536 MB* e o sistema operacional é o *macOs Monterey versão 12.2.1*.

Capítulo 6

Resultados e Análises

Neste capítulo são apresentados os resultados e análises do estudo experimental. Estes resultados estão divididos de acordo com a variação do tipo de base de dados. Assim, a Seção 6.1 contém observações sobre os resultados obtidos para cada uma das base de dados. E na Seção 6.2 é apresentada uma análise geral destes resultados.

Para a coleta dos dados, cada experimento foi executado 10 vezes (sobre certas condições, Tabela 5.3) para cada uma das base de dados. Além disso, foi definido arbitrariamente um tempo limite máximo de três horas para cada execução. Neste contexto, após as execuções dos algoritmos, os resultados são armazenados em uma estrutura contendo indicadores como memória total utilizada, tempo total gasto no processamento, assim como as Top-k regras retornadas (para geração dos resultados desta pesquisa foi definido $k = 10$). As condições de execução de cada algoritmo são detalhadas a seguir:

> ***Baseline***: o algoritmo é executado mantendo o valor padrão de confiança mínima (80%) presente na biblioteca *mlxtend*[1] e para cada execução são definidos os seguintes valores, respectivamente, para suporte mínimo: 95%, 80%, 70%, 60%, 50%, 40%, 30%, 20%, 10%, 5%. Desta forma, busca-se alcançar mais resultados para este algoritmo;
>
> **Genético**: por conta da natureza estocástica deste algoritmo não foi necessária a realização de nenhuma configuração prévia para as execuções.
>
> **Iterativo**: para o algoritmo Iterativo, foram avaliados diferentes mecanismos para criação e utilização das amostras (Seção 6.1). Desta forma, busca-se validar a solução proposta executando-a em diversas amostras de uma mesma base de dados.

Os nomes dos algoritmos utilizados neste estudo experimental foram abreviados para facilitar e simplificar a identificação dos mesmos em tabelas e gráficos da seção de resultados a seguir. O algoritmo *Baseline* (Seção 4.1) passa a ser reconhecido como

[1] http://rasbt.github.io/mlxtend/

Bas. O algoritmo Genético (Seção 4.2) é reconhecido como *Gen*. Já o algoritmo Iterativo Comum (sem variação, Seção 4.3) e suas variações com Redução de Transação (ou chamada Variação com Log, Seção 4.3.1), com Top-k (4.3.2) e com Skyline (Seção 4.3.3) são nomeados como *Ic*, *Ilog*, *It* e *Is*, respectivamente. A Tabela 6.1 apresenta as abreviações destes algoritmos.

Tabela 6.1: Legenda de abreviações usadas nas tabelas de resultados

Algoritmo	**Abreviação**
Baseline	*Bas*
Genético	*Gen*
Iterativo Comum	*Ic*
Iterativo c/ Log	*Ilog*
Iterativo c/ Top-k	*It*
Iterativo c/ Skyline	*Is*

6.1 Resultados para Diferentes Bases de dados

Em todas as bases de dados, para o algoritmo Iterativo, foram geradas previamente 10 pastas contendo, em cada pasta, uma amostra da base de dados em questão. Estas amostras são geradas aleatoriamente e sem replicação de registros, logo, a probabilidade de geração de amostras iguais é baixa, no entanto, foi realizada uma verificação manual sobre estes dados. As amostras também são fixas por execução e comum entre todos os algoritmos Iterativos (todas as variações do mesmo). Seguindo a Equação 4.2, a Tabela 6.2 contém os parâmetros utilizados para encontrar o tamanho da amostra e o valor da mesma para as bases de dados.

Após o processamento de todos os algoritmos em todas as bases de dados, são obtidas informações sobre: quantidade de regras válidas (Qtd. Regras), suporte (Sup), confiança (Conf) e score dessas regras, bem como memória utilizada (mem.) e tempo total de processamento. As seções 6.1.1, 6.1.2, 6.1.3 e 6.1.4 apresentam os resultados encontrados.

6.1.1 Purchases

A Tabela 6.3 mostra a média de valores para 10 execuções dos algoritmos na base `Purchases`. Vale ressaltar que para os algoritmos Iterativos, além de operarem sobre toda a base de dados, cada execução está relacionada a uma amostra especifica gerada previamente e aleatoriamente.

Os resultados apresentados nesta tabela indicam que os algoritmos *Ic*, *Ilog*, *It* e *Is* foram capazes de extrair entre 88% a 100% das k regras pretendidas. Enquanto o

Tabela 6.2: Parâmetros para construção das amostras e tamanho das mesmas para cada base de dados utilizada nos algoritmos Iterativos.

Base de Dados	**Parâmetros comuns: Z=1,96, p=50%, q=50%, E=5%** **Parâmetro N**	**Tamanho Amostra (n)**
`Purchases`	1.631	311
`Visited`	27.098	378
`FVMushroom`	8.124	366
`São Roque`	50.000	381

Ic permitiu localizar todas as k regras, o algoritmo *Ilog* consumiu aproximadamente $1,5$MB de memória no processamento dos dados e o algoritmo *It* foi executado em aproximadamente 28 segundos.

Para questões de custo computacional (memória e tempo de execução), o algoritmo *Bas* foi o que apresentou melhor desempenho, no entanto ele não permitiu encontrar nenhuma regra. Neste contexto, pode-se afirmar que seu ganho computacional se deu porque o intervalo de suporte mínimo utilizado [95%, 5%] foi alto demais para a base de dados em questão e com isso a fase de poda removeu todos os candidatos fazendo com que poucos dados fossem armazenados na memória e com isso finalizando o processamento rapidamente. Em relação ao custo computacional apresentado pelo algoritmo *Gen*, este foi o mais custoso dentre todos os outros algoritmos.

Tabela 6.3: Média dos resultados de 10 execuções dos algoritmos na base *Purchases*.

Alg.	**Qtd. Regras**	**Sup**	**Conf**	**Score**	**Mem.**	**Tempo**
Ic	100%	0,0025	0,88	1,00	2,6MB	41,8s
Ilog	88%	0,0033	0,53	0,87	1,5MB	28,4s
It	98%	0,0032	0,70	0,98	2,0MB	28,1s
Is	98%	0,0034	0,61	0,98	1,7MB	116,0s
Bas	0%	0,0000	0,00	0,00	1,2MB	0,3s
Gen	60%	0,000061	0,10	0,10	3,7MB	196,9s

A Figura 6.1 contém a quantidade de regras retornadas para cada execução dos algoritmos. Estes valores indicam que para o algoritmo *Gen*, ou foi possível localizar 100% das Top-10 regras ou não foi possível encontrar nenhuma regra para esta base de dados. Em contrapartida, os algoritmos Iterativos apresentaram uma porcentagem significativamente alta (quando comparado aos outros algoritmos) na quantidade de regras retornadas para as 10 amostras (com exceção da 9^a execução

do algoritmo *Ilog*). Esta porcentagem baixa na 9^a execução do *Ilog*, ocorre porque após o processo de encurtar as transações da amostra de dados em questão, este algoritmo define um valor relativamente alto de suporte mínimo, que por sua vez permite encontrar apenas 20% das 10 regras.

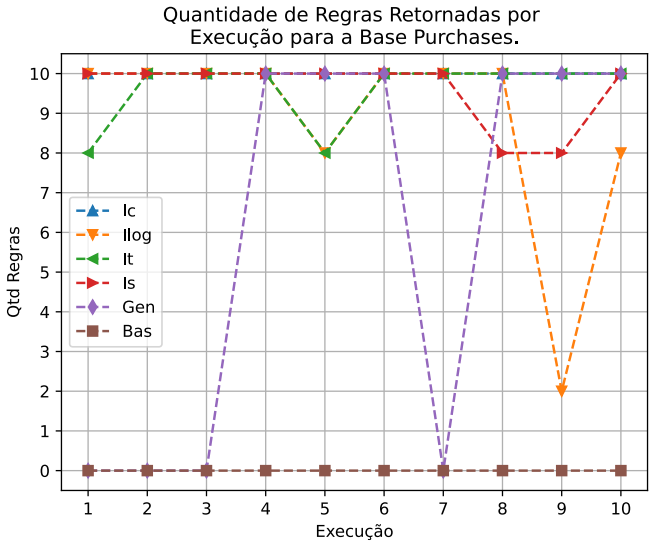


Figura 6.1: Quantidade de regras por execução para a base *Purchases*.

6.1.2 Visited

A Tabela 6.4 mostra a média de valores para 10 execuções dos algoritmos na base `Visited`. Os resultados apresentados nesta tabela indicam que tanto o algoritmo *Bas* quanto o *Gen* não encontraram regras para este conjunto de dados. Enquanto que o *Bas* apresentou a mesma dificuldade mencionada para a base de dados `Purchases` (valores de suporte mínimo altos para os dados em questão) o algoritmo *Gen* excedeu o tempo limite de processamento estipulado para três horas em cada execução.

Em geral os algoritmos Iterativos apresentaram valores acima de 50% para a média da quantidade de regras válidas retornadas. Onde o algoritmo *It* retornou aproximadamente 64% das k regras pretendidas com score em torno de $0,64$. O algoritmo *Ilog* consumiu aproximadamente 53MB de memória. E o algoritmo *Is*, também, com valores de score das regras de aproximadamente $0,64$, processou os dados em 38 minutos. Vale ressaltar que mesmo as bases de dados `Purchases` e `Visited` possuindo o mesmo contexto para uma mesma empresa, o tempo de processamento para base `Visited` foi muito maior porque, esta base contém aproximadamente 16 vezes mais registros que a base `Purchases`, além de possuir, no geral, mais itens por registro.

Novamente, o algoritmo *Bas* não foi capaz de retornar regras. Isso reafirma a dificuldade em definir certos parâmetros de acordo com a base de dados. Os valores de suportes mínimos encontrados pelos algoritmos Iterativos foram de aproximadamente $0,002$ $(0,2\%)$, enquanto que o intervalo estipulado previamente para os

suportes mínimos utilizado no *Bas* foi de [95%, 5%]. Neste contexto, quando o mesmo suporte mínimo encontrado por um algoritmo Iterativo (numa execução que teve resultado) é usado no algoritmo *Bas*, são obtidas as mesmas regras, no entanto, ao diminuirmos esse suporte mínimo aplicado ao *Bas*, o algoritmo excede o tempo limite de processamento, logo, o manuseio deste parâmetro é algo "sensível".

Tabela 6.4: Média dos resultados de 10 execuções dos algoritmos na base *Visited*.

Alg.	**Qtd. Regras**	**Sup**	**Conf**	**Score**	**Mem.**	**Tempo**
Ic	40%	0,002	0,097	0,39	75,0MB	41,0min
Ilog	58%	0,002	0,196	0,58	53,3MB	54,9min
It	64%	0,002	0,267	0,64	58,9MB	57,3min
Is	60%	0,001	0,362	0,64	56,7MB	38,6min
Bas	0%	0,000	0,000	0,00	54,0MB	0,4s
Gen	-	-	-	-	-	timeout

Ao considerar a Figura 6.2, o algoritmo *Ic* apresentou consistência na quantidade de regras retornadas, no entanto ele encontrou menos regras (na média) que os outros algoritmos Iterativos. A figura também indica que a média dos valores (apresentados na Tabela 6.4) para os algoritmos *Ilog*, *It* e *Is*, sofreram interferência negativa do processamento das últimas três execuções. Por conta desta interferência ter acontecido em três dos quatro algoritmos Iterativos, isso demonstra que para esta base de dados, as transações presentes nas amostras das últimas três execuções não foram capazes de refletir satisfatoriamente a base de dados completa.

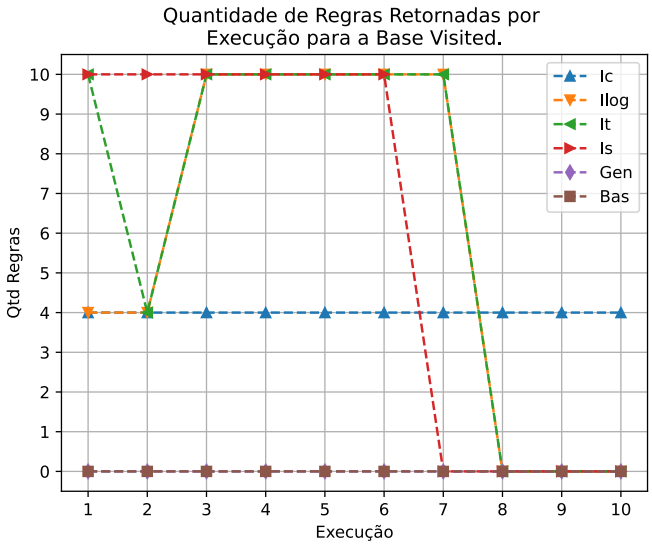


Figura 6.2: Quantidade de regras por execução para a base *Visited*.

6.1.3 FVMushroom

Os resultados obtidos para a base sintética `FVMushroom` estão apresentados na Tabela 6.5. Os algoritmos *Ic* e *It* foram os únicos capazes de retornar 100% das k regras desejadas (com média de score de $1,20$), no entanto, foram os que mais consumiram memória dentre os outros algoritmos. Em contrapartida, os algoritmos *Ilog* e *Is* devolveram 96% e 86% das k regras, consumindo $2,2$MB e $4,0$MB, respectivamente. Os algoritmos *Bas* e *Gen* consumiram aproximadamente $2,0$MB de memória, mas retornaram 25% e 9% das k regras desejadas, respectivamente.

No geral, os algoritmos Iterativos apresentaram média de quantidade de regras retornadas acima de 85%, com tempo de processamento maior que 20 segundos. Por outro lado, o algoritmo *Bas* foi o que apresentou melhor desempenho em relação ao tempo de processamento, contudo, sua média de regras retornadas foi de 25%.

Tabela 6.5: Média dos resultados de 10 execuções dos algoritmos na base *FVMushroom*.

Alg.	Qtd. Regras	Sup	Conf	Score	Mem.	Tempo
Ic	100%	0,257	0,837	1,20	7,1MB	1,8min
Ilog	96%	0,257	0,757	1,14	2,2MB	20,9s
It	100%	0,257	0,873	1,20	7,9MB	1,8min
Is	86%	0,246	0,619	1,01	4,0MB	2,2min
Bas	25%	0,061	0,218	0,30	2,1MB	0,4s
Gen	9%	0,0002	0,015	0,09	2,0MB	66,9s

A Figura 6.3 apresenta a quantidade de regras retornadas por execução para todos os algoritmos. Os algoritmos *Ic* e *It* apresentaram valores semelhantes em suas 10 execuções. Em alguns casos os algoritmos *Ilog* e *Is* identificaram valores de suporte mínimo relativamente altos para a base de dados, fazendo com que retornassem menos que 60% das k regras desejadas, no entanto, na maioria das execuções foi possível encontrar 100% das regras. O algoritmo *Gen* apresentou muita inconsistência na quantidade de regras retornadas, além de devolver menos que 5% das k regras em todas as execuções.

Para o algoritmo *Bas*, a distribuição dos dados na base `FVMushroom` (cinco itens por registro e com alguns itens muito frequentes) permitiu que fosse possível encontrar as regras, no entanto, apenas após a utilização do suporte mínimo de 40%. Sendo que a quantidade de regras retornadas atingiu 100% apenas para os valores de suporte mínimo de 10% e 5%.

6.1.4 São Roque

Os resultados para a base de dados `São Roque` apresentaram, em geral, valores baixos para as médias de quantidade de regras resultantes e consequentemente para a média do score dessas regras. Assim, os resultados presentes na Tabela 6.6 indicam

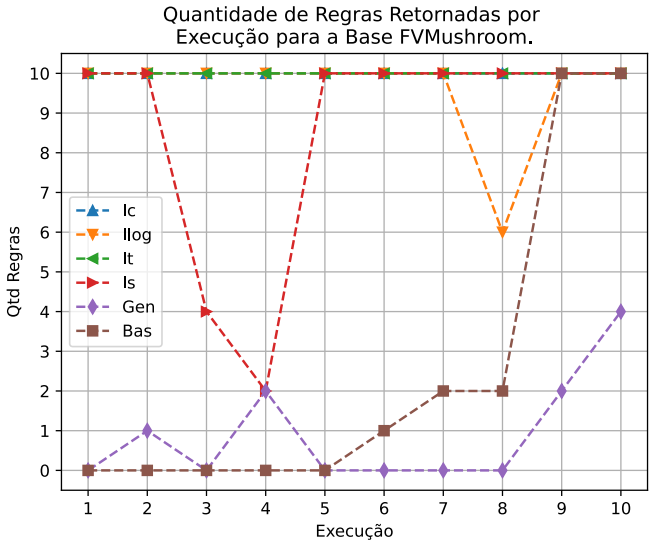


Figura 6.3: Quantidade de regras por execução para a base *FVMushroom*.

que os algoritmos *Ic*, *It* e *Gen* não foram capazes de devolver ao usuário as regras durante o tempo limite de processamento (3 horas). O algoritmo *Bas* também não foi capaz de encontrar regras, mas isso ocorreu por conta dos valores de suportes mínimos utilizados [95%, 5%]. Estes valores fizeram com que não houvessem conjuntos frequentes de itens candidatos para geração de regras.

Em contrapartida, os algoritmos *Ilog* e *Is* foram capazes de encontrar regras resultantes, mesmo que em baixa quantidade. Nesse contexto, o algoritmo *Is* foi o que apresentou os melhores resultados, quando comparado a todos os outros algoritmos Iterativos, tanto em questão de quantidade de regras retornadas, quanto em score, memória utilizada e tempo total de processamento.

Tabela 6.6: Média dos resultados de 10 execuções dos algoritmos na base *São Roque*.

Alg.	**Qtd. Regras**	**Sup**	**Conf**	**Score**	**Mem.**	**Tempo**
Ic	-	-	-	-	-	timeout
Ilog	10%	0,0004	0,04	0,10	247,7MB	2,4h
It	-	-	-	-	-	timeout
Is	16%	0,0009	0,09	0,16	246,6MB	5,5min
Bas	0%	0,0000	0,00	0,00	248,8MB	8,6s
Gen	-	-	-	-	-	timeout

A Figura 6.4 contém a quantidade de regras retornadas para cada execução dos algoritmos. Estes valores indicam que apenas os algoritmos Iterativo com Log (Il) e Skyline (Is) foram capazes de encontrar regras resultantes para a base de dados em questão. Sendo o algoritmo Is o que apresentou maior constância na busca das regras. No entanto, a disposição das quantidades de regras retornadas presentes na

Figura 6.4, aponta para uma amostragem não muito representativa desta base de dados, ainda sim, estes algoritmos Iterativos se saíram melhor que os demais.

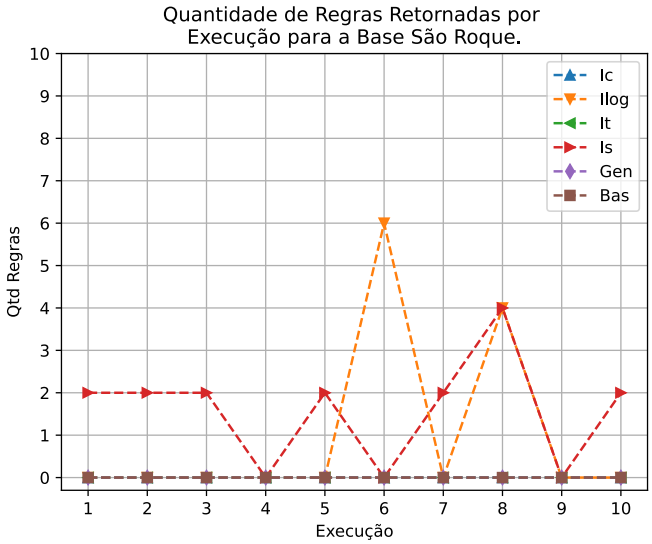


Figura 6.4: Quantidade de regras por execução para a base *São Roque*.

6.2 Análises

Com base nos resultados ilustrados nas seções anteriores, é possível reafirmar a dificuldade de estabelecer certos parâmetros em algoritmos de mineração de conjuntos frequentes de itens e regras de associação.

O algoritmo *Gen* conseguiu obter regras válidas para as bases de dados `Purchases` e `FVMushroom`. No entanto, o mesmo demonstrou a necessidade de executar o algoritmo várias vezes, alterando seus parâmetros até que seja possível encontrar resultados tão próximos ou mais altos que os encontrados pelos outros algoritmos (em termos de média de quantidade de regras, score, memória utilizada e tempo de processamento). Além disso, as taxas de probabilidade utilizadas neste algoritmo afetam de maneira específica cada conjunto de dados e o parâmetro de tamanho das regras também interfere na qualidade e quantidade final de regras.

A definição de parâmetros externos também afetou os resultados gerados pelo algoritmo *Bas*. O intervalo fixo utilizado para os valores de suporte mínimo [95%, 5%] permitiu que para algumas execuções a fase de poda do algoritmo removesse todos os conjuntos de itens candidatos, enquanto em outras execuções o valor de suporte mínimo gerasse muitos conjuntos frequentes de itens e com isso excedesse o tempo limite de processamento. Isso evidencia ainda mais a dificuldade em selecionar estes parâmetros (limiares).

De modo geral, os algoritmos Iterativos apresentaram resultados que possuem maiores valores para as médias da quantidade de regras válidas e do score dessas regras,

especialmente os algoritmos que envolvem o uso de Log (*Ilog*) e Skyline (*Is*). Estes foram capazes de encontrar regras resultantes na maioria das bases de dados utilizadas. Isso demonstra que os algoritmos Iterativos conseguem se adaptar melhor a diferentes bases de dados e não necessitam de parâmetros externos para localizar regras. No entanto, os resultados apresentados também sugerem que conforme o tamanho das bases de dados aumenta os algoritmos Iterativos passam a enfrentar maiores dificuldades para encontrar um suporte mínimo adequado, o que indica que nem todas as amostras estão refletindo fielmente a base de dados completa.

Neste contexto, torna-se importante o desenvolvimento e aperfeiçoamento de algoritmos que não dependam de parâmetros externos tão "sensíveis" e que permitam encontrar regras para qualquer base de dados sem necessitar de muitas execuções de alto custo computacional.

Capítulo 7

Aplicação

Para avaliar melhor a importância de um sistema capaz de obter regras de associação sem que o operador deste sistema especifique os limiares de suporte e confiança, foi criada uma aplicação[1] que permite comparar dois métodos: um método que requer suporte e confiança e outro que não requer. Utilizou-se como base de dados para esse aplicativo as disciplinas que são cursadas concomitantemente pelos estudantes da UEFS. A base de dados utilizada corresponde ao segundo semestre letivo do ano de 2022 da UEFS, onde cada registro desta base representa uma disciplina cursada por aluno e contém informações como o código que representa um estudante, mas não permite identificá-lo, visto que não é o número de matrícula (UsId), o nome do curso (Curso) que o aluno está inserido, bem como o departamento deste curso (Dept.), o código da disciplina (CodDis.) e o nome da disciplina (Disciplina) que o aluno está matriculado. Um exemplo deste conjunto de dados está representado na Tabela 7.1.

A aplicação desenvolvida oferece duas interfaces (mecanismos), Figura 7.1. Com o

[1] https://uefs-search.herokuapp.com/

Tabela 7.1: Exemplo do conjunto de dados utilizado na aplicação, contendo os campos departamento da disciplina (Dept.), nome do curso (Curso), nome da disciplina (Disciplina), código da disciplina (CodDis.) e código do aluno (UsId).

Dept.	Curso	Disciplina	CodDis.	UsId
TEC	Engenharia de Computação	Arquitetura de Computadores	402	5
TEC	Engenharia de Computação	Sistemas Operacionais	408	5
TEC	Engenharia de Computação	MI - Sistemas Digitais	499	5
CIS	Ciências Econômicas	Monografia I	227	1029
CIS	Ciências Econômicas	Economia Baiana	255	1029

intuito de facilitar a validação dos resultados por parte dos usuários, o parâmetro de curso do aluno foi incluído em ambas as interfaces. O Mecanismo 1 (localizado no lado esquerdo da Figura 7.1) utiliza o algoritmo Apriori para realizar a busca, enquanto que o Mecanismo 2 (localizado no lado direito da Figura 7.1) utiliza o algoritmo Iterativo sem variações (comum). Ao passo que o Mecanismo 1 requer parâmetros adicionais como suporte, confiança e número de regras desejadas (k), o Mecanismo 2 requer apenas o parâmetro adicional k.

Deste modo, o usuário após selecionar o curso, preencher o restante dos parâmetros e clicar no botão de busca, Figura 7.2, pode então comparar os resultados, por exemplo, em termos de quantidade e conteúdo das regras resultantes, tempo de processamento, valores de pontuação (score), suporte e confiança de cada regra.

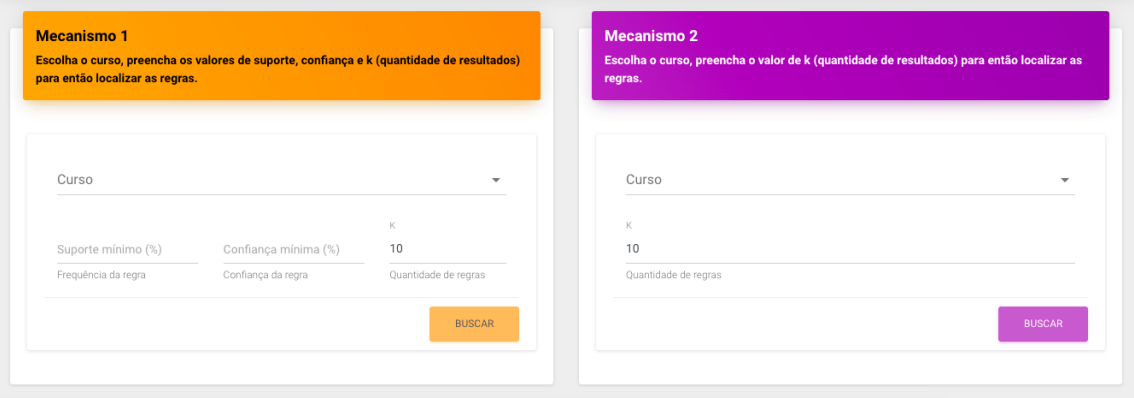


Figura 7.1: Interfaces de busca da aplicação web.

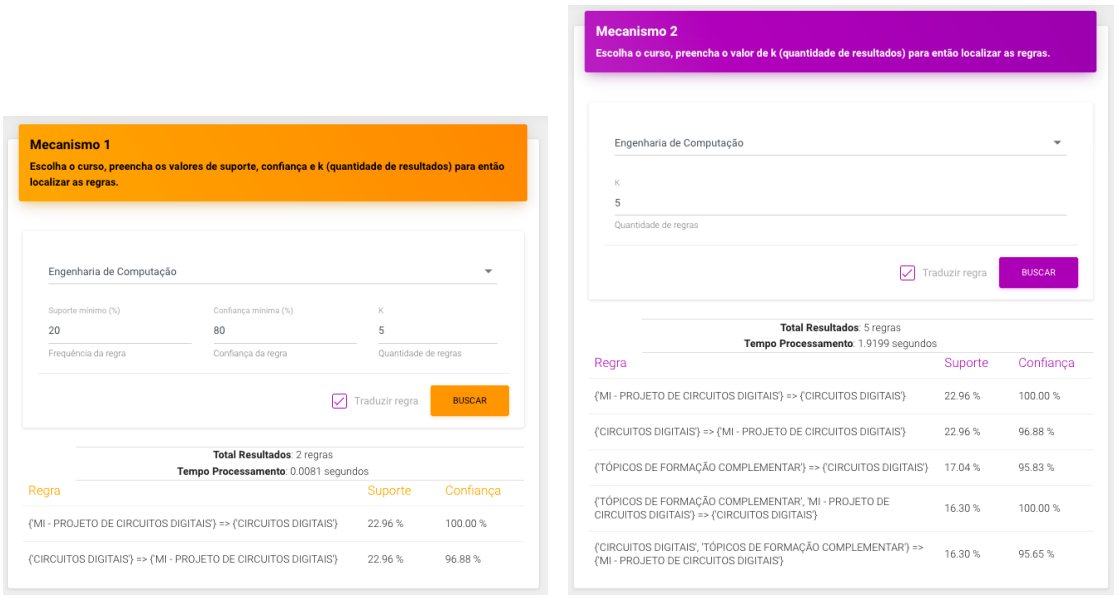
É possível conferir o score de uma determinada regra ao posicionar o cursor do mouse sobre a mesma, conforme exemplo presente na Figura 7.3. Além disto, vale ressaltar que as regras apresentadas nesta aplicação estão no formato $X => Y$, onde X e Y representam conjuntos de disciplinas separadas por vírgula, no qual X caracteriza o antecedente da regra e Y o consequente da regra, vide Figuras 7.2 e 7.3.

Para a aquisição dos dados sobre as impressões dos usuários, um questionário foi incluído na aplicação. Deste questionário foram extraídas as seguintes informações.

1. O perfil dos usuários que realizaram a pesquisa;
2. A percepção sobre a vantagem de usar um determinado mecanismo em detrimento do outro baseado na comparação do processo de busca e dos resultados de ambos os mecanismos;
3. A percepção sobre a utilidade real da busca de disciplinas cursadas concomitantemente para o auxílio à assuntos acadêmicos utilizando a aplicação desenvolvida.

Esta ferramenta foi apresentada no SBBD[2] e contou com a avaliação de 21 participantes. Os resultados desta avaliação estão na Figura 7.4.

[2]https://sbbd.org.br/2022/



(a) Resultados para o algoritmo Apriori.

(b) Resultados para o algoritmo Iterativo.

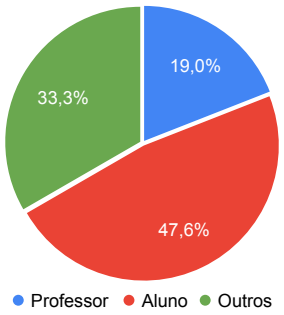
Figura 7.2: Exemplo dos resultados gerados para os dois mecanismos utilizando os parâmetros [curso: Engenharia de Computação, suporte: 20%, confiança: 80% e k: 5] para o Mecanismo 1 e os parâmetros [curso: Engenharia de Computação e k:5] para o Mecanismo 2.

Dentre os que acessaram a ferramenta, foi possível observar a presença de professores, alunos de mestrado e doutorado, entre outros interessados no assunto (Figura 7.4(a)).

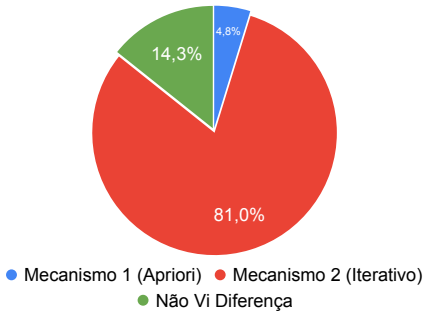
Além disso, os resultados também indicam que, para os que utilizaram a ferramenta, o mecanismo que se apoia no algoritmo Iterativo se mostrou ser o mais vantajoso, Figura 7.4(b). Isto pode ser explicado devido ao fato de que o mecanismo 2 requer menos parâmetros de busca que o mecanismo 1, além disso, inserir valores arbitrários nos campos de suporte mínimo e confiança mínima podem trazer resultados não esperados (valores de corte relativamente altos), bem como demora na busca (valores de corte relativamente baixos).

Em relação a utilização da ferramenta para vida real, no que diz respeito ao auxílio à assuntos acadêmicos, os participantes do questionário acreditam, em sua maioria, que esta ferramenta pode sim ser útil para este fim, Figura 7.4(c).

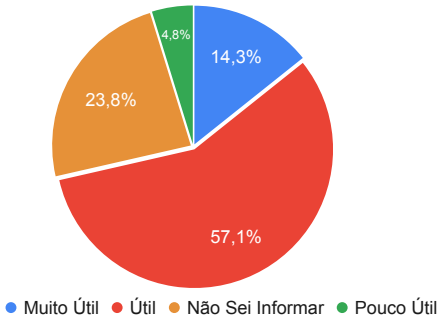
Figura 7.3: Modo de visualização do score de uma regra na aplicação web.



(a) Resultado da 1ª questão.



(b) Resultado da 2ª questão.



(c) Resultado da 3ª questão.

Figura 7.4: Resultados do questionário envolvendo 21 participantes.

Capítulo 8

Conclusão

Esta pesquisa realizou um estudo experimental em seis algoritmos de mineração de regras de associação. Os resultados apontam que tanto o algoritmo Genético quanto o *Baseline* necessitam de uma análise prévia das bases de dados para que a escolha dos parâmetros destes algoritmos torne possível a extração de regras válidas aos usuários. O algoritmo Genético permitiu extrair regras em apenas uma das bases de dados, sendo que muitas dessas regras estavam repetidas. O algoritmo *Baseline* foi o algoritmo mais rápido no processamento dos dados, no entanto foi o que apresentou a pior taxa de retorno de resultados válidos. Isto reafirma a dificuldade de definir os parâmetros de suporte mínimo e confiança mínima para o mesmo.

Em contrapartida, os algoritmos Iterativos, propostos pelos autores deste trabalho permitiram encontrar regras de associação válidas para a maioria das bases de dados. Com destaque para os algoritmos Iterativos utilizando Log e Skyline, que foram capazes de localizar regras para todas as bases de dados. Isto demonstra que a solução proposta é capaz de se adaptar melhor a variação da base de dados e não necessita de parâmetros externos para localizar regras. No entanto, os algoritmos Iterativos apresentaram limitações quanto à mineração de regras em bases de dados grandes, pois, mesmo operando inicialmente sobre amostras pequenas, conforme a base de dados crescia as amostras geradas não eram capazes de refletir a base de dados completa e com isso prejudicando o processo de inferência do algoritmo. Além disso, o tempo de reposta obtido pelos algoritmos ainda é muito insatisfatório para grandes bases de dados. Sendo assim, ainda se faz necessário o desenvolvimento de novas técnicas e abordagens que permitam reduzir esse tempo de resposta, possibilitando que a obtenção de regras sem a especificação prévia dos limiares de suporte e confiança seja aplicada em Sistemas de Informação.

Neste contexto, uma nova forma de amostrar os dados pode ser aplicada aos algoritmos, permitindo que essas amostras consigam refletir de forma mais fiel as transações das bases de dados. Também é válida a ideia de executar o algoritmo proposto em diversas pequenas amostras de uma única base ao invés de utilizar apenas uma única amostra. Além disto, uma abordagem de crescimento do suporte mínimo pode ser

empregada tanto na primeira parte do algoritmo (amostragem) quanto na segunda parte (busca global) a fim de acelerar o processo de identificação dos conjuntos frequentes de itens e das regras de associação. Outras sugestões para melhoria desta pesquisa são adicionar mais algoritmos e bases de dados no experimento e utilizar diferentes funções fitness para o cálculo do score das regras de associação.

Finalmente, vale ressaltar a importância de desenvolver novas tecnologias que permitam encontrar regras de associação sem a necessidade de especificar valores de suporte mínimo e confiança mínima, pois, estes valores são difíceis de serem encontrados na prática. Em algumas bases de dados utilizadas nesta pesquisa, os valores de suporte e confiança necessários para retornar regras válidas eram muito baixos e sensíveis a pequenas variações, desta forma, em certos momentos nenhuma regra era retornada e outras vezes muitos conjuntos de itens eram processados, requerendo muito tempo de processamento.

Referências

Agrawal, R., Imieliński, T., e Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A. I., et al. (1996). Fast discovery of association rules. *Advances in knowledge discovery and data mining*.

Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*. Citeseer.

Anselmo, F. C. G. (2017). Regras de associação-market basket analysis-items frequentes e itens raros.

Bäck, T., Fogel, D. B., e Michalewicz, Z. (2018). *Evolutionary computation 1: Basic algorithms and operators*. CRC press.

Baldomir, R. A. (2017). Aplicação do algoritmo Apriori para detectar relacionamentos entre empresas nos processos licitatórios do Governo Federal.

Borgelt, C. (2010). Simple algorithms for frequent item set mining. In *Advances in machine learning II*. Springer.

Borzsony, S., Kossmann, D., e Stocker, K. (2001). The skyline operator. In *Proc. 17th int. conf. on data engineering*. IEEE.

Camilo, C. O. e Silva, J. C. d. (2009). Mineração de dados: Conceitos, tarefas, métodos e ferramentas. *Universidade Federal de Goiás (UFC)*.

Chomicki, J. (2003). Preference formulas in relational queries. *ACM Transactions on Database Systems (TODS)*.

Correa, S. (2003). Probabilidade e estatística.

Dahbi, A., Jabri, S., Balouki, Y., e Gadi, T. (2016). A new method for ranking association rules with multiple criteria based on dominance relation. In *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*. IEEE.

De Oliveira, M. F. (2011). Metodologia científica: um manual para a realização de pesquisas em administração. *Universidade Federal de Goiás. Catalão–GO*.

Deshmukh, R. A., Bharathi, H., e Tripathy, A. K. (2019). Parallel processing of frequent itemset based on mapreduce programming model. In *2019 5th International Conference On Computing, Communication, Control And Automation (ICCUBEA)*. IEEE.

Devi, M. R. (2012). Applications of association rule mining in different databases. *Journal of Global Research in Computer Science*.

Djenouri, Y. e Comuzzi, M. (2017). Combining Apriori heuristic and bio-inspired algorithms for solving the frequent itemsets mining problem. *Information Sciences*.

Eiben, A. E., Smith, J. E., et al. (2015). *Introduction to evolutionary computing*. Springer.

Fournier-Viger, P., Lin, J. C.-W., Vo, B., Chi, T. T., Zhang, J., e Le, H. B. (2017). A survey of itemset mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*.

Galvão, N. D. e Marin, H. d. F. (2009). Técnica de mineração de dados: uma revisão da literatura. *Acta Paulista de Enfermagem*.

Gupta, M. K. e Chandra, P. (2020). A comprehensive survey of data mining. *International Journal of Information Technology*.

Han, J., Pei, J., e Yin, Y. (2000). Mining frequent patterns without candidate generation. *ACM sigmod record*.

Han, J., Pei, J., Yin, Y., e Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*.

Han, J., Wang, J., Lu, Y., e Tzvetkov, P. (2002). Mining top-k frequent closed patterns without minimum support. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* IEEE.

Hand, D. J. (2007). Principles of data mining. *Drug safety*.

Hirate, Y., Iwahashi, E., e Yamana, H. (2004). Tf2p-growth: An efficient algorithm for mining frequent patterns without any thresholds. In *Proceedings of ICDM*. ICDM.

Huang, Y.-P. e Kao, L.-J. (2004). Using fuzzy support and confidence setting to mine interesting association rules. In *IEEE Annual Meeting of the Fuzzy Information, 2004. Processing NAFIPS'04.* IEEE.

Ilyas, I. F., Beskales, G., e Soliman, M. A. (2008). A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*.

Israel, G. D. (1992). Determining sample size.

Kalyvas, C. e Tzouramanis, T. (2017). A survey of skyline query processing. *arXiv preprint arXiv*.

Kameya, Y. e Sato, T. (2012). Rp-growth: top-k mining of relevant patterns with minimum support raising. In *Proceedings of the 2012 SIAM international conference on data mining*. SIAM.

Keogh, E., Lonardi, S., Ratanamahatana, C. A., Wei, L., Lee, S.-H., e Handley, J. (2007). Compression-based data mining of sequential data. *Data Mining and Knowledge Discovery*.

Koh, J.-L., Lin, C.-Y., e Chen, A. L. P. (2014). Finding k k most favorite products based on reverse top-t t queries. *The VLDB Journal*, 23(4).

Kotsiantis, S. e Kanellopoulos, D. (2006). Association rules mining: A recent overview. *GESTS International Transactions on Computer Science and Engineering*.

Lai, K. e Cerpa, N. (2001). Support vs. confidence in association rule algorithms. In *Proc. of the OPTIMA Conference, Curicó*.

Li, Z.-C., He, P.-L., e Lei, M. (2005). A high efficient AprioriTid algorithm for mining association rule. In *2005 international conference on machine learning and cybernetics*. IEEE.

Lucchese, C., Orlando, S., e Perego, R. (2005). Fast and memory efficient mining of frequent closed itemsets. *IEEE Transactions on Knowledge and Data Engineering*.

Luna, J. M., Fournier-Viger, P., e Ventura, S. (2019). Frequent itemset mining: A 25 years review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*.

Martín, D., Martínez-Ballesteros, M., García-Gil, D., Alcalá-Fdez, J., Herrera, F., e Riquelme-Santos, J. (2018). Mrqar: A generic mapreduce framework to discover quantitative association rules in big data problems. *Knowledge-Based Systems*.

Medeiros, A. F. et al. (2015). Uma proposta de personalização de consultas em documentos xml baseada em preferências condicionais.

Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.

Mohammed, M., Azzeddine, D., Youssef, B., e Taoufiq, G. (2015). S $_{\text{em}}$ MDP $_{\text{ref}}$: algorithm to filter and sort rules using a semantically based ontology technique. In *Proc. 7th int. conf. on management of computational and collective intelligence in Digital EcoSystems*. ACM.

Moslehi, F. e Haeri, A. (2020). A genetic algorithm-based framework for mining quantitative association rules without specifying minimum support and minimum confidence. *Scientia Iranica*.

Nguyen, L. T., Vo, B., Nguyen, L. T., Fournier-Viger, P., e Selamat, A. (2018). ETARM: an efficient top-k association rule mining algorithm. *Applied Intelligence*.

Prithiviraj, P. e Porkodi, R. (2015). A comparative analysis of association rule mining algorithms in data mining: a study. *Open J. Comput. Sci. Eng. Surv.*

Qodmanan, H. R., Nasiri, M., e Minaei-Bidgoli, B. (2011). Multi objective association rule mining with genetic algorithm without specifying minimum support and minimum confidence. *Expert Systems with applications*.

Riondato, M. e Upfal, E. (2015). Mining frequent itemsets through progressive sampling with rademacher averages. In *SIGKDD*.

Rocha-Junior, J. B. (2013). *Efficient Processing of Preference Queries in Distributed and Spatial Databases.* Tese de Doutorado, Citeseer.

Santos, M. A. S. d. (2017). Estudo comparativo de algoritmos exaustivos para mineração de padrões discriminativos em bases de dados biomédicas. Dissertação de Mestrado, Universidade Federal de Pernambuco.

Scheaffer, R. L., Mendenhall III, W., Ott, R. L., e Gerow, K. G. (2011). *Elementary survey sampling*. Cengage Learning.

Singh, A. S. e Masuku, M. B. (2014). Sampling techniques & determination of sample size in applied statistics research: An overview. *International Journal of economics, commerce and management*.

Tabassum, M., Mathew, K., et al. (2014). A genetic algorithm analysis towards optimization solutions. *International Journal of Digital Information and Wireless Communications (IJDIWC)*.

Telikani, A., Gandomi, A. H., e Shahbahrami, A. (2020). A survey of evolutionary computation for association rule mining. *Information Sciences*.

Vasconcelos, L. M. R. e Carvalho, C. L. (2018). Aplicação de regras de associação para mineração de dados na web. *Revista Telfract*.

Vlachou, A., Doulkeridis, C., Kotidis, Y., e Nørvåg, K. (2010). Reverse top-k queries. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. IEEE.

Vlachou, A., Doulkeridis, C., Rocha-Junior, J. B., e Nørvåg, K. (2022). On decisive skyline queries. In *ICBDAKD*. Springer.

Vo, B. e Le, B. (2011). Mining minimal non-redundant association rules using frequent itemsets lattice. *International journal of intelligent systems technologies and applications*.

Yin, M., Wang, W., Liu, Y., e Jiang, D. (2018). An improvement of FP-Growth association rule mining algorithm based on adjacency table. In *MATEC Web of Conferences*. EDP Sciences.

Zaki, M. J. (2000). Scalable algorithms for association mining. *IEEE transactions on knowledge and data engineering*.

Zaki, M. J. e Gouda, K. (2003). Fast vertical mining using diffsets. In *Proc. of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*.

Zaki, M. J., Parthasarathy, S., Li, W., e Ogihara, M. (1997). Evaluation of sampling for data mining of association rules. In *International Workshop on Research Issues in Data Engineering. High Performance Database Management for Large-Scale Applications*. IEEE.

Zeng, Y., Yin, S., Liu, J., e Zhang, M. (2015). Research of improved FP-Growth algorithm in association rules mining. *Scientific Programming*.