UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

JÚLIO CÉSAR BORBA NANDI

A TÉCNICA DE ASSOCIAÇÃO PELO ALGORITMO *FREQUENT PATTERN-GROWTH (FP-GROWTH)* NA *SHELL ORION DATA MINING ENGINE*

CRICIÚMA

2013

JÚLIO CÉSAR BORBA NANDI

A TÉCNICA DE ASSOCIAÇÃO PELO ALGORITMO *FREQUENT PATTERN-GROWTH (FP-GROWTH)* NA *SHELL ORION DATA MINING ENGINE*

Trabalho de Conclusão de Curso, apresentado para a obtenção do grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientadora: Profª. MSc. Merisandra Côrtes de Mattos Garcia

CRICIÚMA

2013

JÚLIO CÉSAR BORBA NANDI

A TÉCNICA DE ASSOCIAÇÃO PELO ALGORITMO *FREQUENT PATTERN-GROWTH (FP-GOWTH)* NA *SHELL ORION DATA MINING ENGINE*

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel, no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Inteligência Computacional.

Criciúma, 27 de Novembro de 2013.

BANCA EXAMINADORA

Profª. Merisandra Côrtes de Mattos Garcia - MSc. - (UNESC) - Orientador

Prof. Cristian Cechinel - Dr. - (UFPEL)

Prof. Kristian Madeira - MSc. - (UNESC)

RESUMO

O crescimento acelerado na coleta de dados nos mais variados campos dos negócios e da área científica tem despertado o interesse dos profissionais a atentar para a necessidade de se desenvolver tecnologias que permitam a análise e exploração de conhecimentos úteis implícitos nesses dados. Neste contexto, o *data mining* se destaca, pois automatiza o processo da extração de conhecimentos por meio de ferramentas e algoritmos computacionais. Essas ferramentas, por sua vez, na grande maioria são proprietárias e possuem um alto custo de aquisição. Considerando isso, o Grupo de Pesquisa em Inteligência Computacional Aplicada do Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, desenvolve diversos métodos e tarefas de *data mining* por meio do projeto da *Shell Orion Data Mining Engine*. Dentre as tarefas de *data mining*, a descoberta de regras de associação tem recebido atenção por parte de pesquisadores tanto da área acadêmica como das organizações. Na área acadêmica, as pesquisas desenvolvidas estão gerando bons resultados. Esses resultados, por sua vez, estão sendo utilizados pelas organizações em aplicações práticas. Objetivando ampliar as funcionalidades da *Shell Orion*, essa pesquisa consiste na demonstração do funcionamento e na implementação do algoritmo *Frequent Pattern-Growth*, desenvolvido para superar as limitações presentes no algoritmo Apriori e melhorar a eficiência da descoberta de regras de associação por meio de uma abordagem diferente, sem a geração do conjunto de candidatos. Também não concorda com o paradigma de gerar e testar do Apriori, uma vez, que codifica a base de dados em uma estrutura compacta em forma de árvore chamada *Frequent Pattern tree* e extrai os conjuntos de itens frequentes diretamente desta estrutura. Isso possibilita uma melhor eficiência na geração das regras de associação, pois evita constantes acessos na base de dados. Tendo-se concluída a implementação, diversos testes foram efetuados tendo por objetivo a análise de desempenho do algoritmo implementado. Fez-se isso por meio de medidas de qualidade para regras de associação, análise dos resultados obtidos e análise do tempo de processamento por meio de cálculos estatísticos. Os resultados das regras identificadas, comparadas com a Weka, mostraram que a *Shell Orion* encontrou corretamente as regras de associação úteis e fortes e também calculou corretamente as medidas de qualidade. Já quanto ao tempo de processamento verificou-se que a Weka obteve tempos um pouco melhores, porém do ponto de vista estatístico, a diferença não foi significativa.

Palavras-chave: Inteligência Computacional. *Data Mining*. Associação. Algoritmo *FP-Growth*. Árvore *FP-tree*.

ABSTRACT

The rapid growth in data collection in various fields of business and scientific area has piqued the interest of professionals to attend to the need to develop technologies that enable the analysis and exploitation of useful knowledge implicit in these data. In this context, data mining stands out because it automates the process of extracting knowledge through computational tools and algorithms. These tools, in turn, the vast majority are proprietary and have a high cost. Considering this, the Research Group on Applied Computational Intelligence Course of Computer Science, University of the Extreme South of Santa Catarina, develops various methods of data mining tasks and through the Shell Orion Data Mining Engine project. Among the tasks of data mining, the discovery of association rules has received attention from researchers in academia as both organizations. In academia, the researches developed are generating good results. These results, in turn, are used by organizations in practical applications. Aiming to extend the functionality of the Shell Orion, this research is to demonstrate the operation and implementation of the Frequent Pattern-Growth algorithm, developed to overcome the limitations present in the Apriori algorithm and improve the efficiency of the discovery of association rules by means of a different approach without generating the set of candidates. Also disagrees with the paradigm of Apriori generate and test once, encoding the database in a compact structure as a tree called Frequent Pattern tree and extract the sets of frequent items directly from this structure. This enables better efficiency in the generation of association rules, because it avoids constant access to the database. Having completed the implementation, several tests were performed with the purpose of performance analysis of the implemented algorithm. This was done by means of quality measures for association rules, analysis of results and analysis of the processing time through statistical calculations. The results of the identified rules, compared with Weka, showed that Shell Orion correctly found the rules of votes and strong association and also correctly calculated the quality measures. As for the processing time it was found that the Weka times got a little better, but from a statistical standpoint, the difference was not significant.

Keywords: Computational Intelligence. Data Mining. Association. FP-Growth Algorithm. Tree FP-tree.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A²CA	*Adaptive Ant-Clustering Algorithm*
API	*Application Programming Interface*
CART	*Classification and Regression Trees*
DBSCAN	*Density-Based Spatial Clustering of Applications with Noise*
DHP	*Direct Hashing and Pruning*
DM	*Data Mining*
FP-Growth	*Frequent Pattern-Growth*
FP-tree	*Frequent Pattern-tree*
GSP	*Generalized Sequential Patterns*
HSQLDB	*HyperSQL Database*
ICMC-USP	Instituto de Ciências Matemáticas e de Computação-Universidade de São Paulo
ID3	*Induction of Decision Tree*
JDBC	*Java Database Connectivity*
KDD	*Knowledge Discovery in Databases*
LHS	*Left Hand Side*
RBF	*Radial Basis Function*
RCP	*Robust C-Prototypes*
RHS	*Right Hand Side*
RNA	*Redes Neurais Artificiais*
SACA	*Standard Ant Clustering Algorithm*
SAMiRA	Sistema de Apoio à Mineração de Regras de Associação
SGBD	Sistemas Gerenciadores de Banco de Dados
SPSS	*Statistical Package for the Social Sciences*
SQL	*Structured Query Language*
TCC	Trabalho de Conclusão de Curso
TI	Tecnologia da Informação
UML	*Unified Modeling Language*
UNESC	Universidade do Extremo Sul Catarinense
URCP	*Unsupervised Robust C-Prototypes*
USP	Universidade de São Paulo
WEKA	*Waikato Environment for Knowledge Analysis*

SUMÁRIO

1 INTRODUÇÃO

Constantes avanços no armazenamento de dados e no conhecimento sobre as mais diversas áreas contribuíram para a formação de grandes e múltiplos repositórios de dados, tornando imperativo o desenvolvimento de tecnologias destinadas à análise das informações contidas nos mesmos, viabilizando a obtenção de novos conhecimentos.

A necessidade de ampliar as capacidades de análise dos humanos para manusear o número de *bytes* que se pode recolher é ao mesmo tempo econômica e científica. Científica porque os computadores permitiram aos seres humanos armazenar mais dados do que sua capacidade de entendê-los em tempo hábil. Econômica porque as empresas utilizam os dados para ganhar vantagem na competitividade e aumentar a eficiência, fornecendo serviços valiosos aos seus clientes (FAYYAD; PIATETSKY-SHAPIRO; SMYTH,1996a, tradução nossa; TAN; STEINBACH; KUMAR, 2009).

A análise das relações úteis entre os dados é conhecida como Descoberta de Conhecimento em Bases de Dados, *Knowledge Discovery in Databases (KDD*), esse processo inclui limpeza, integração, seleção e transformação de dados; *data mining*; avaliação de padrões e apresentação do conhecimento (HAN; KAMBER, 2006, tradução nossa). Sendo o *data mining* a mais destacada etapa desse processo.

O *data mining* se destaca por proporcionar descoberta rápida e automática, ou pelo menos parcialmente automática, de informações úteis e relevantes em grandes repositórios de dados em comparação com os métodos tradicionais, por meio de ferramentas e algoritmos computacionais. Constituindo-se em um campo interdisciplinar, que proporciona a confluência de um conjunto de disciplinas que incluem, entre outras, sistemas de banco de dados, estatísticas, aprendizado de máquina, visualização e ciência da informação (HAN; KAMBER, 2006, tradução nossa). Estas áreas reunidas visam um fim específico, ou seja, a descoberta de conhecimentos implícitos em repositórios de dados.

No contexto da aplicação de KDD a etapa de *data mining* é a responsável pela busca dos conhecimentos relevantes, envolvendo, para isso, a execução de métodos e algoritmos nas bases de dados. A escolha desses depende, em grande parte, do tipo de tarefa a ser realizada e do problema a ser resolvido

(GOLDSCHMIDT; PASSOS, 2005).

Devido as ferramentas de KDD serem em sua maioria comerciais, o Grupo de Pesquisa em Inteligência Computacional Aplicada do Curso de Ciência da Computação da UNESC, desenvolve diversos métodos e tarefas de *data mining* por meio do projeto da *Shell Orion Data Mining Engine*. A *Shell Orion* possui implementadas as tarefas de classificação (algoritmos ID3, C4.5, CART, RBF e *Naive Bayes*), clusterização (algoritmos *K-Means*, *Kohonen*, *Fuzzy C-Means, Gustafson-Kessel, Gath-Geva, Robust C-Prototypes* (RCP), *Unsupervised Robust C-Prototypes* (URCP), *Density-Based Spatial Clustering of Applications With Noise* (DBSCAN), *Standard Ant Clustering Algorithm* (SACA), *Ant-Based Clustering* e *Adaptive Ant-Clustering Algorithm* (A²CA)) e associação (algoritmo *Apriori*).

Na tarefa de associação a geração de regras tem sido reconhecida na literatura como um problema importante no campo do *data mining*. Aplicações que visam à descoberta de regras de associação estendem a procura por padrões úteis no comportamento do consumidor, marketing de alvo, e comércio eletrônico, entre outros (AGARWAL; AGGARWAL; PRASAD, 2000, tradução nossa; TAN; STEINBACH; KUMAR, 2009).

A tarefa de geração de regras de associação foi inicialmente apresentada por Agrawal et al (1993, tradução nossa) a partir da análise dos itens presentes em compras de um supermercado, objetivando a descoberta de relações do tipo: “Um cliente que compra os produtos $A_1, A_2,..., A_n$, também comprará os produtos $B_1, B_2,..., B_n$ com probabilidade p%”.

De uma forma geral, a tarefa de associação permite identificar o quanto a presença de um conjunto de itens nos registros de uma base de dados implica na presença de algum outro conjunto distinto de itens nos mesmos registros (MOTTA, 2010).

Com o objetivo de resolver o problema da descoberta de regras de associação diversos algoritmos foram desenvolvidos, dentre os quais se destaca o algoritmo *Apriori* (AGRAWAL; SRIKANT, 1994) por ser um algoritmo pioneiro e clássico; e o algoritmo *FP-Growth* (HAN et al, 2004; HAN; PEI; YIN, 2000) que gera o conjunto completo de regras sem para isso gerar candidato. A complexidade de um sistema de descoberta de regras de associação é dependente do algoritmo que está sendo utilizado (GOLDSCHMIDT; PASSOS, 2005; MELANDA, 2004).

Algoritmos tradicionais de associação adotam uma abordagem igual ou semelhante a do algoritmo *Apriori*, que se baseia na seguinte regra: se qualquer padrão de comprimento *k* não é frequente na base de dados, seu comprimento (*k* + 1) não será frequente. A ideia é, por meio de um processo iterativo, gerar o conjunto de padrões de candidatos de comprimento (*k* +1) a partir do conjunto de padrões de frequência de comprimento *k* (para $k \geq 1$), e verificar suas frequências de ocorrência na base de dados. No entanto, a geração de conjunto candidato é ainda dispendiosa especialmente quando há um grande número de padrões e/ou estes são longos, ou seja, padrões formados por um número expressivo de itens. Também é dispendioso percorrer repedidas vezes a base de dados para verificar e testar todos os conjuntos de candidatos e seus padrões correspondentes (HAN et al, 2004, tradução nossa).

Com o objetivo de superar essas limitações e melhorar a eficiência da descoberta de regras de associação diversos métodos e algoritmos alternativos têm sido desenvolvidos. Como por exemplo o algoritmo *FP-Growth* que utiliza uma abordagem diferente do *Apriori*, ou seja, sem a geração do conjunto de candidatos. Também não concorda com o paradigma de gerar e testar do *Apriori*, ao contrário disso, codifica o conjunto de dados em uma estrutura de dados compacta em forma de árvore chamada *Frequent Pattern tree* (*FP-tree*) e extrai os conjuntos de itens frequentes diretamente desta estrutura. Isso possibilita uma melhor eficiência na geração das regras de associação, pois evita constantes acessos na base de dados (TAN; STEINBACH; KUMAR, 2009).

Como parte do projeto da *Shell Orion Data Mining Engine*, este trabalho acadêmico visa o estudo do algoritmo *FP-Growth* e a implementação do mesmo no módulo de associação da *Shell Orion*.

1.1 OBJETIVO GERAL

Disponibilizar o algoritmo *FP-Growth* na tarefa de associação da *Shell Orion Data Mining Engine*.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos desta pesquisa são:

a) compreender os conceitos de *data mining* e a tarefa de associação;

b) entender o algoritmo *FP-Growth* e suas técnicas para não gerar candidato;
c) aplicar o algoritmo *FP-Growth* na tarefa de associação de itens frequentes na *Shell Orion Data Mining Engine*;
d) demonstrar matematicamente o funcionamento do algoritmo *FP-Growth*;
e) analisar a qualidade da regra gerada pelo algoritmo desenvolvido por meio de medidas de qualidade em *data mining*;
f) analisar o desempenho do algoritmo desenvolvido por meio de medidas estatísticas.

1.3 JUSTIFICATIVA

O crescimento acelerado na coleta de dados nos mais variados campos dos negócios e da área científica tem despertado o interesse dos profissionais a atentar para a necessidade de analisar e explorar conhecimentos úteis implícitos nesses dados (WITTEN; FRANK; HALL, 2011, tradução nossa; ZAKI, 2000, tradução nossa).

Para atender a essa crescente necessidade o *data mining* foi concebido, seu principal objetivo é automatizar o processo de extração de conhecimentos em bases de dados, para que novos conhecimentos sejam descobertos em tempo hábil. Objetivando tudo isso o *data mining* é implementado em ferramentas e algoritmos computacionais. Essas aplicações, por sua vez, na grande maioria são proprietárias e possuem um alto custo de aquisição. Este cenário justifica o projeto da *Shell Orion Data Mining Engine*, que visa implementar tarefas, métodos e algoritmos importantes do processo de *data mining*, em um software acessível a comunidade em geral e sem custo de aquisição.

Dentre as tarefas de *data mining*, a descoberta de regras de associação tem recebido atenção por parte de pesquisadores tanto da área acadêmica como das organizações. Na área acadêmica, as pesquisas desenvolvidas estão gerando bons resultados. Esses resultados estão sendo utilizados pelas organizações em aplicações práticas. Um exemplo da utilização prática da tarefa de associação pode ser vista na área comercial em páginas de vendas de produtos na internet, que apresentam sugestões do tipo: “Acessórios com preços reduzidos. Economize

adquirindo acessórios junto com seu *Notebook* X" (HAN; KAMBER, 2006, tradução nossa; MELANDA, 2004).

A tarefa de associação em *data mining* consiste em identificar os conjuntos de itens frequentes em uma base de dados, e em seguida, formar regras de implicação condicional entre tais itens. Por exemplo, de uma base de dados de uma grande rede de supermercados pode-se obter a seguinte regra: 80% dos clientes que compram pão e leite, também compram manteiga e ovos (WITTEN; FRANK; HALL, 2011, tradução nossa; ZAKI et al, 1997, tradução nossa). Com esses conhecimentos em mãos as empresas podem melhorar os serviços prestados aos seus clientes, prever cenários futuros e disponibilizar novos serviços.

Dentre os algoritmos que descobrem regras de associação em uma base de dados tem-se o algoritmo *FP-Growth* que disponibiliza um método eficiente que determina o conjunto completo de itens frequentes (HAN; KAMBER, 2006, tradução nossa; TAN; STEINBACH; KUMAR, 2009). Segundo Han et al (2004, tradução nossa) e Han, Pei e Yin (2000, tradução nossa) a eficiência do algoritmo *FP-Growth* é conseguida com três técnicas:

a) a base de dados original é comprimida em uma condensada estrutura de dados (*FP-tree*), isso evita buscas repetidas na base de dados;
b) a estrutura *FP-tree-based* adota um método, baseado em um padrão de crescimento fragmentado contínuo, que evita a dispendiosa geração de um grande número de conjuntos de candidatos;
c) um método de particionamento baseado na técnica de dividir para conquistar é usado para decompor a tarefa da geração dos N *itemsets* frequentes implícitos na *FP-tree* em um conjunto de pequenas tarefas, confinados em bases de dados condicionais, o que reduz o espaço de busca.

Considerando as vantagens proporcionadas pelo algoritmo *FP-Growth* proposto por Han et al (2004) e por Han, Pei e Yin (2000), justifica-se o desenvolvimento do mesmo na *Shell Orion*. Esta pesquisa tem por objetivo dar continuidade ao desenvolvimento da ferramenta, acrescendo o algoritmo *FP-Growth* ao seu módulo de associação.

1.4 ESTRUTURA DO TRABALHO

Este trabalho acadêmico é composto por seis capítulos, sendo que o Capítulo 1 apresenta uma visão geral da pesquisa, descrevendo também os objetivos pretendidos, e justificativa do trabalho.

Os principais conceitos relacionados ao processo de descoberta do conhecimento (KDD), a etapa de *data mining* e a *Shell Orion Data Mining Engine* são contextualizados no Capítulo 2.

A tarefa de associação em *data mining* é o tema do Capítulo 3, bem como os algoritmos que implementam a descoberta de regras de associação. O algoritmo *FP-Growth* é detalhado no Capítulo 4, onde o método que cria a *FP-tree* e o que gera os N *itemsets* frequentes a partir da *FP-tree* criada são descritos.

O Capítulo 5 apresenta alguns trabalhos correlatos que implementam o algoritmo *FP-Growth*.

No Capítulo 6 são descritas as etapas do trabalho desenvolvido, a metodologia utilizada e os resultados obtidos por meio da implementação do algoritmo *FP-Growth* no módulo de associação da *Shell Orion*.

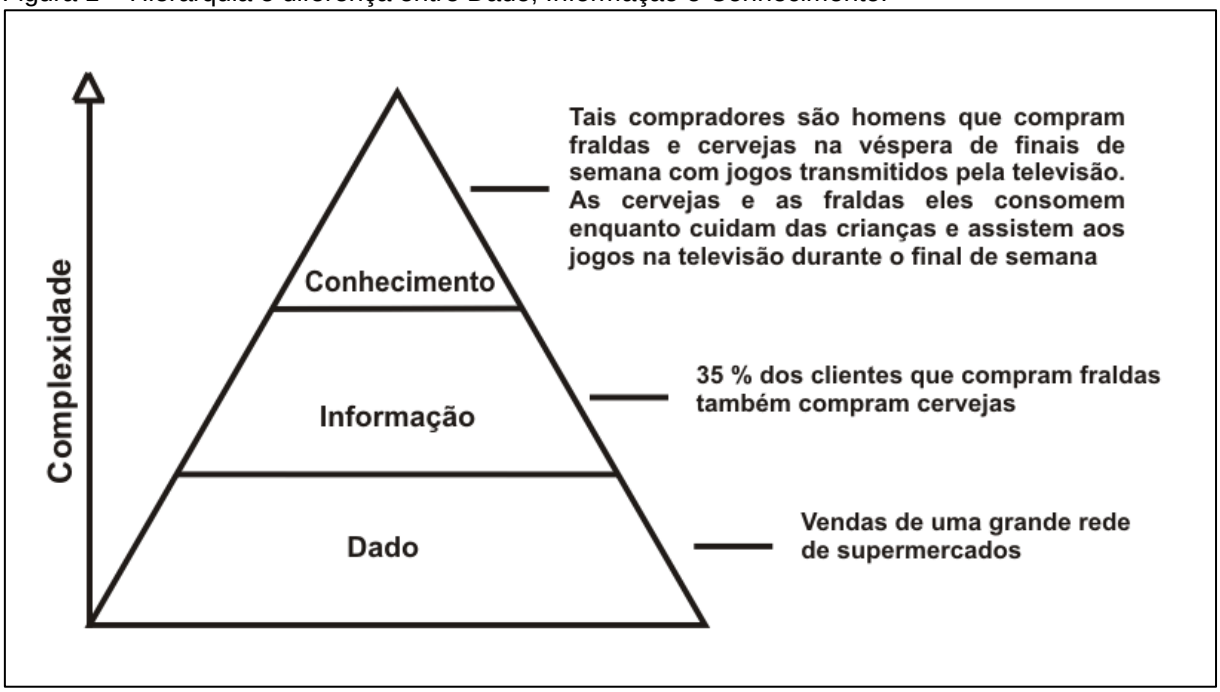
Finalizando, tem-se a conclusão deste trabalho acadêmico e as sugestões de trabalhos futuros.

2 DESCOBERTA DE CONHECIMENTO EM BASES DE DADOS

Constantes progressos na coleta automatizada de dados, e na disponibilidade de armazenamento barato, proporcionaram a coleta de grandes quantidades de dados (TAN; STEINBACH; KUMAR, 2009). De fato a capacidade de analisar e entender os conjuntos de dados é inferior à capacidade de reunir e armazenar os mesmos (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996a, tradução nossa; HAN; KAMBER, 2006, tradução nossa). Portanto a criação de técnicas e ferramentas computacionais é necessária para proporcionar agilidade na extração de conhecimento útil dos volumes de dados em rápida expansão.

Conforme Goldschmidt e Passos (2005), para atender esta necessidade, surgiu a área denominada Descoberta de Conhecimento em Bases de Dados (*Knowledge Discovery in Databases - KDD*), pois o problema e a resolução do mesmo vêm despertando o interesse das comunidades científica, industrial e comercial. Para um entendimento melhor do problema, é importante compreender as diferenças e a hierarquia entre dado, informação e conhecimento, ilustrado na figura 1.

Figura 1 – Hierarquia e diferença entre Dado, Informação e Conhecimento.



Fonte: Adaptado de Goldschmidt e Passos (2005).

Na base da pirâmide, os dados, são os itens fundamentais, captados e

armazenados por recursos da Tecnologia da Informação (TI). As informações representam os dados processados, com significados bem definidos no contexto. Recursos de TI facilmente processam os dados e geram informações. O conhecimento, por sua vez, está no topo da pirâmide, trata-se de um padrão ou conjunto de padrões onde a formulação da regra de negócio pode relacionar e envolver informações e dados (GOLDSCHMIDT; PASSOS, 2005).

Fayyad, Piatetsky-Shapiro e Smyth (1996a, tradução nossa) definem o KDD como um processo, composto por várias etapas, não trivial, interativo e iterativo, de identificação de padrões válidos, novos, potencialmente úteis e em última análise compreensíveis a partir de grandes conjuntos de dados.

Examinar em mais detalhes a definição do processo de KDD se faz necessário para o entendimento do processo como um todo:

a) **em última análise compreensíveis:** é um dos objetivos do KDD tornar padrões compreensíveis para os seres humanos, a fim de facilitar o entendimento dos dados armazenados (HAN; KAMBER, 2006, tradução nossa);
b) **potencialmente úteis:** são aqueles que quando aplicados proporcionam benefícios ao contexto da aplicação de KDD (GOLDSCHMIDT; PASSOS, 2005);
c) **novos:** pode ser medida analisando-se as mudanças nos resultados obtidos, por meio de comparações com os resultados atuais ou esperados, ou ainda, como uma nova descoberta está relacionada com a antiga (TAN; STEINBACH; KUMAR, 2009);
d) **padrões válidos:** são padrões descobertos e validados com o objetivo de adquirirem certo grau de confiabilidade, ou seja, são verdadeiros, fidedignos e adequados (FAYYAD; PIATETSKY-SHAPIRO; SMYTH,1996a, tradução nossa; HAN; KAMBER, 2006, tradução nossa);
e) **iterativo:** durante a execução, pode haver a possibilidade de repetições parciais ou integrais na busca de resultados satisfatórios por meio de sucessivos refinamentos (GOLDSCHIMIDT; PASSOS, 2005);
f) **interativo:** necessita da atuação do homem intervindo como responsável pelo controle do processo. Para isso, o homem, utiliza os recursos computacionais a sua disposição com o intuito de analisar e

interpretar os resultados obtidos (GOLDSCHIMIDT; PASSOS, 2005);

g) **não trivial:** a análise automática para a extração de informação útil mostra-se desafiadora. Isso ocorre devido ao tamanho muito grande da base dados, ou, às vezes, a base de dados é pequena, mas os dados não são triviais, e em outras ocasiões as técnicas existentes não são suficientes para se analisar os dados, necessitando o desenvolvimento de novos métodos (TAN; STEINBACH; KUMAR, 2009);
h) **várias etapas:** trata-se de uma série de etapas operacionais, do pré-processamento dos dados até o pós-processamento dos resultados obtidos (FAYYAD; PIATETSKY-SHAPIRO; SMYTH,1996a, tradução nossa; HAN; KAMBER, 2006, tradução nossa).

2.1 ETAPAS DO PROCESSO DE KDD

O KDD é um processo, interativo e iterativo, que envolve várias etapas com decisões a serem feitas pelo especialista do domínio de aplicação. As etapas devem ser desenvolvidas para atingir um fim específico, no caso, a descoberta de conhecimento implícito em repositórios de dados. Cada etapa possui uma intersecção com as demais, ou seja, os resultados coletados em uma etapa são utilizados para melhorar os resultados das próximas etapas (FAYYAD; PIATETSKY-SHAPIRO; SMYTH,1996a, tradução nossa; SASSI, 2006; HAN; KAMBER, 2006, tradução nossa).

A figura 2 mostra as etapas do processo de KDD e procura dar uma visão prática de sua natureza interativa e iterativa. As etapas do processo de KDD são:

a) **seleção:** nesta etapa se desenvolve a compreensão do domínio de aplicação e a relevância do conhecimento que se pretende adquirir, ou seja, o objetivo do processo de KDD. Para isso é criado um conjunto de dados alvo ou dados selecionados (FAYYAD; PIATETSKY-SHAPIRO; SMYTH,1996b, tradução nossa). Esta etapa também é denominada redução de dados, porque envolve a identificação de quais informações, dentre as disponíveis nas bases de dados, devem realmente ser consideradas durante o processo de KDD (GOLDSCHIMIDT; PASSOS, 2005);
b) **pré-processamento:** tem por objetivo transformar os dados brutos em

um formato adequado para as etapas seguintes do processo de KDD. O pré-processamento envolve a fusão de dados de múltiplas fontes, a limpeza dos dados para a remoção de ruídos e informações duplicadas, a seleção de registros e atributos que sejam relevantes, ou seja, abrange todo o tratamento realizado sobre os dados selecionados de forma a assegurar a qualidade dos fatos que eles representam (GOLDSCHIMIDT; PASSOS, 2005; TAN; STEINBACH; KUMAR, 2009);

c) **transformação:** também chamada de Codificação dos Dados, visa a conversão de um conjunto de dados brutos selecionados e pré-processados, em um formato padrão de uso. Pode ser necessário transformar a forma em que os dados estão representados, a fim de superar possíveis limitações existentes no algoritmo de extração de conhecimento que será executado[1]. As vantagens desta fase são: aumentar a compreensão do conhecimento descoberto, diminuir o tempo de processamento na etapa de *Data Mining* (DM), entre outras (BATISTA, 2003; PYLE, 1999, tradução nossa; SASSI, 2006);

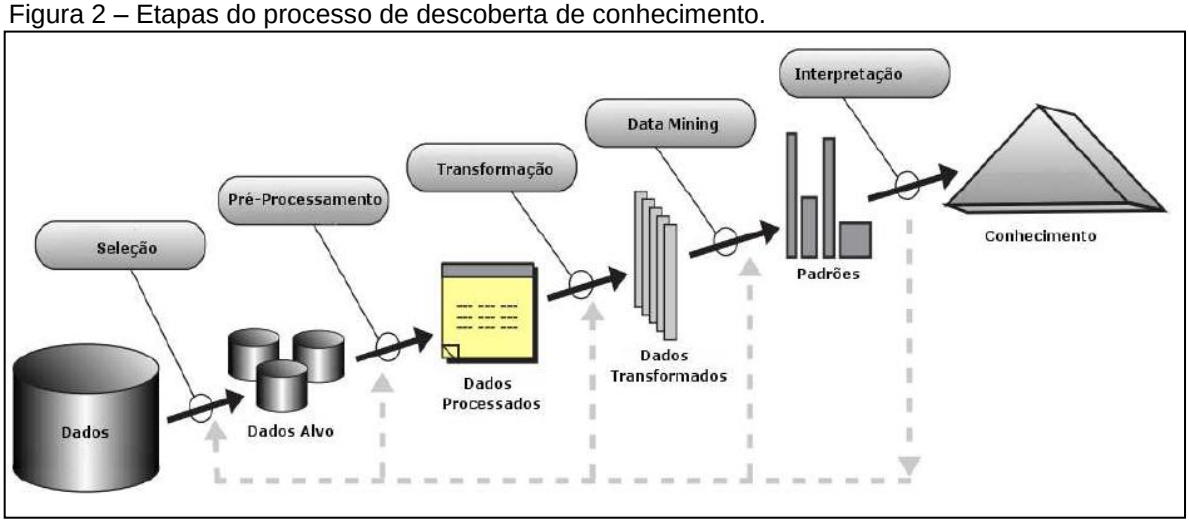
d) ***data mining*:** o termo DM derivou das semelhanças entre a procura de informações importantes em bases de dados e o ato de mineração para encontrar ouro. Os dois processos sugerem a seleção de um imenso amontoado de material, e uma análise inteligente e cuidadosa desse material, para encontrar o valor desejado. O DM é a principal etapa do processo de KDD, caracterizada pela existência do algoritmo minerador. Nesta etapa ocorre a aplicação de algoritmos sobre os dados, resultantes das etapas anteriores, visando a descoberta de conhecimentos implícitos e úteis (GOLDSCHIMIDT; PASSOS, 2005; SASSI, 2006);

e) **pós-processamento:** envolve a visualização, interpretação e análise do conhecimento descoberto na etapa do DM. É nesta etapa que o especialista no domínio de aplicação e o especialista em KDD analisam os resultados gerados e definem novas alternativas de investigação

[1] Um exemplo de limitação é a incapacidade de analisar alguns formatos de dados como atributos de data e hora por exemplo. Assim, costuma-se transformar esses tipos de dados em outro formato, capaz de ser analisado pelo algoritmo na fase de *data mining*. Um atributo do tipo data, por exemplo, pode ser transformado em um atributo do tipo inteiro, onde representa o número de dias decorridos a partir de uma data fixa (BATISTA, 2003).

dos dados. A principal meta dessa fase é melhorar o entendimento do conhecimento descoberto, para isso os resultados são validados por meio de medidas de qualidade e da percepção dos especialistas envolvidos (GOLDSCHIMIDT; PASSOS, 2005; SASSI, 2006).

Figura 2 – Etapas do processo de descoberta de conhecimento.



Fonte: Adaptado de Fayyad; Piatetsky-Shapiro; Smyth (1996a).

Dentre todas as etapas do processo de KDD, nesta pesquisa acadêmica será dada maior ênfase ao DM.

2.2 *DATA MINING*

O DM refere-se a uma etapa no processo de KDD, onde ocorre a aplicação de algoritmos específicos para extrair padrões ou modelos dentro de grandes bases de dados. A distinção entre o processo de KDD e a etapa do DM deve existir, pois a aplicação do DM sem a execução das demais etapas do processo de KDD pode facilmente levar a descoberta de conhecimento sem sentido e ou de padrões inválidos (HAN; KAMBER, 2006, tradução nossa).

O DM se caracteriza por ser um processo essencial que combina métodos tradicionais de análise de dados com algoritmos sofisticados para processar grandes volumes de dados. Esse cenário criou oportunidades interessantes para se explorar e analisar novos tipos de dados e técnicas antigas de novas maneiras. Objetivando a automação da descoberta de padrões ou modelos úteis e recentes que poderiam, de outra forma, permanecer desconhecidos (TAN; STEINBACH; KUMAR, 2009).

O DM pode ser amplamente aplicado em distintos campos de pesquisa,

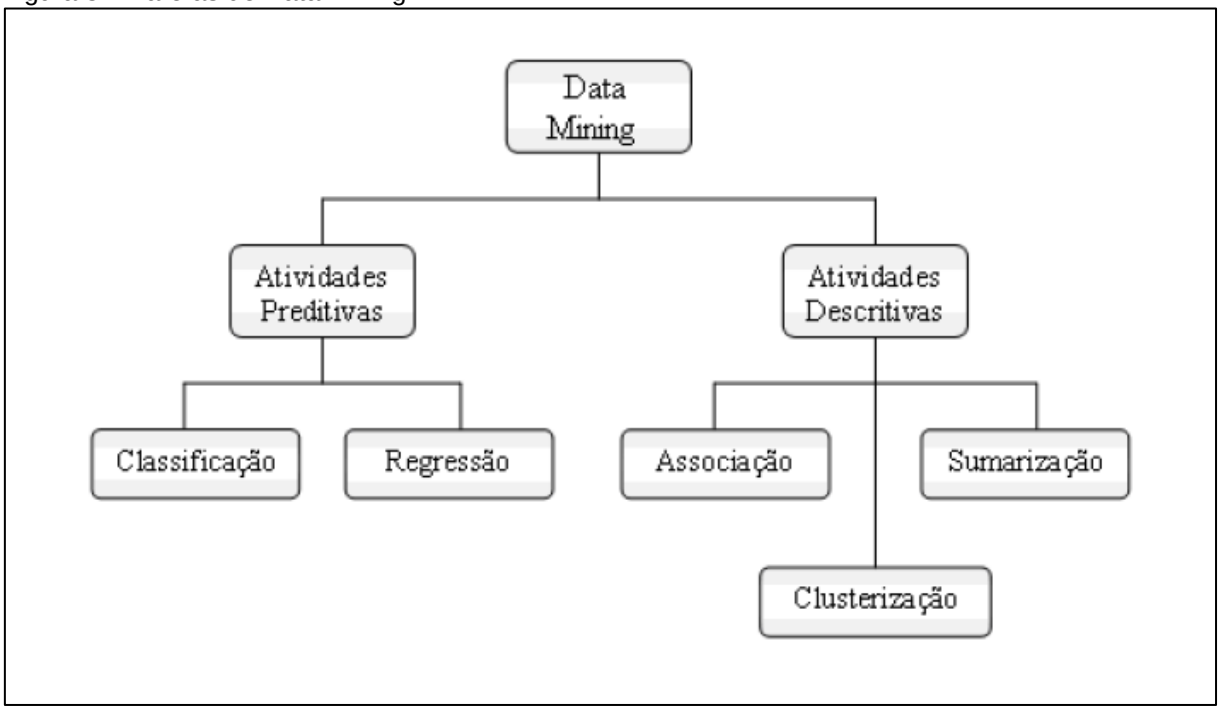
como por exemplo: diagnósticos, segmentação de imagens de satélite, previsão de carga de sistemas elétricos, decisões que envolvem julgamento, mineração de dados na web, marketing e vendas, entre outros (WITTEN; FRANK; HALL, 2011, tradução nossa).

Considerando os vários campos de aplicação do DM, e que os especialistas do processo de KDD podem estar procurando tipos distintos de padrões em certas bases de dados, existem diversas tarefas de DM, sendo que a escolha de uma determinada tarefa depende do problema que se pretende resolver e do conhecimento que se deseja descobrir (GOLDSCHIMIDT; PASSOS, 2005).

2.2.1 Tarefas e Métodos de Data Mining

As tarefas de DM são aplicadas de acordo com o padrão de conhecimento que se deseja descobrir, sendo geralmente classificadas em duas categorias: preditivas e descritivas (figura 3). As tarefas preditivas têm por objetivo prever o valor de um atributo futuro ou desconhecido baseando-se em outros atributos existentes. As tarefas descritivas por sua vez procuram padrões que simplifiquem os relacionamentos entre os dados, isso possibilita a identificação e interpretação de padrões por seres humanos (FAYYAD; PIATETSKY-SHAPIRO; SMYTH,1996a, tradução nossa).

Figura 3 – Tarefas de Data Mining.



Fonte: Adaptado de Rezende (2005).

Para se atingir as metas de predição e ou descrição são utilizadas tarefas de DM. As tarefas centrais, seus objetivos e exemplos de aplicação são:

a) **associação:** trata-se da descoberta de conjuntos de itens que aparecem simultaneamente e com alta frequência. Encontrar padrões frequentes desempenha um papel essencial no DM. Aplicações úteis de análise de associação incluem entre outros: descoberta de genes que possuem funcionalidade associada; identificação de páginas Web que sejam acessadas simultaneamente; o entendimento dos relacionamentos entre os diferentes elementos do sistema climático da Terra; a identificação de quais produtos em um determinado estabelecimento comercial são vendidos em conjunto, entre outros (TAN; STEINBACH; KUMAR, 2009);

b) **clusterização:** também denominada agrupamento, esta tarefa divide a base de dados em grupos ou *clusters* que sejam úteis e ou tenham significados. Ao contrário da tarefa de classificação, onde as classes são pré-definidas, a clusterização identifica automaticamente os grupos, utilizando medidas de similaridade (GOLDSCHIMIDT; PASSOS, 2005). Esta tarefa pode ser aplicada para compreender o clima da Terra que requer a busca de padrões na atmosfera e no

oceano, ou ainda como pré-processamento para aplicação de outras tarefas (TAN; STEINBACH; KUMAR, 2009);

c) **regressão:** se caracteriza por ser uma tarefa preditiva onde a variável a ser avaliada é contínua, ou seja, uma função, uma vez que a tarefa de regressão se restringe a valores numéricos. Exemplos de aplicações de regressão incluem, entre outros: a previsão do preço futuro de uma ação na bolsa de valores; definição do limite de crédito de um cliente; a projeção das vendas de uma empresa baseada na quantidade gasta em publicidade; a estimativa da probabilidade de um paciente sobreviver, de acordo com os resultados de exames e diagnósticos (HAN; KAMBER, 2006, tradução nossa);

d) **classificação:** tarefa preditiva que organiza um conjunto de dados em uma dentre várias classes pré-definidas pelo analista do processo de KDD. Esta tarefa pode ser aplicada para classificar clientes em índice de baixo, médio ou alto risco de empréstimo; classificar ações da bolsa de valores; identificar transações fraudulentas; entre outras aplicações (FAYYAD; PIATETSKY-SHAPIRO; SMYTH,1996a, tradução nossa);

e) **sumarização:** também chamada de descrição de conceitos, esta tarefa descritiva, consiste em identificar e apresentar as características principais dos dados de um conjunto de dados. Para atingir este objetivo a sumarização utiliza métodos para encontrar uma descrição compacta para um subconjunto de dados, porém, isso não significa que a sumarização é uma simples enumeração de informações. Pelo contrário nesta tarefa os dados são caracterizados de forma resumida. Um exemplo de aplicação é descrever o perfil comum de boa parte dos clientes de uma empresa. Com tal informação o departamento de marketing pode direcionar a oferta de produtos e ou serviços (GOLDSCHIMIDT; PASSOS, 2005).

Para auxiliar na implementação das tarefas de DM existem métodos, entre os quais detacam-se:

a) **Redes Neurais Artificiais (RNA):** é uma método computacional que visa a construção de um modelo matemático baseado em um sistema nervoso biológico, seu funcionamento tende a ser semelhante a alguns procedimentos humanos. A capacidade de generalizar a informação e

aprender por meio de exemplos é o quem tem despertado o interesse para a solução de problemas complexos por meio de RNA (BRAGA; CARVALHO; LUDERMIR, 2000);

b) **lógica fuzzy:** permite a construção de sistemas capazes de tratar informações imprecisas ou subjetivas. Ao contrário da lógica clássica onde um registro pertence a apenas uma classe de dados, o método baseado em lógica *fuzzy* permite que os registros pertençam simultaneamente a mais de uma classe (REZENDE, 2005);

c) **algoritmos genéticos:** segundo Han e Kamber (2006, tradução nossa) algoritmos genéticos são métodos computacionais que possuem a capacidade de adaptação, e se baseiam nos processos genéticos de organismos biológicos. São utilizados para solução de problemas complexos que envolvem classificação, simulação, previsão, otimização e avaliação de outros algoritmos de DM;

d) **árvores de decisão:** se baseia no princípio de “dividir para conquistar”. É utilizada para comprimir uma base de dados em uma estrutura compacta de tamanho menor, ou seja, uma forma de representação mais simples e concisa da base de dados original (WITTEN; FRANK; HALL, 2011, tradução nossa).

A implementação das tarefas e métodos descritos anteriormente solucionam dificuldades decorrentes do processo de KDD, como a necessidade de manipulação de grandes e heterogêneos volumes de dados, a dificuldade de integração de vários algoritmos específicos, e o tratamento de resultados representados em formatos diferentes (GOLDSCHIMIDT; PASSOS, 2005).

Ferramentas específicas que implementam o processo de KDD em ambientes integrados são denominadas *shells*. A tabela 1 descreve algumas ferramentas de DM disponíveis bem como as tarefas e métodos implementados.

Tabela 1. Exemplos de ferramentas de Data Mining

Nome	Técnicas Disponíveis	Fabricante	Tipo
Clementine	Classificação, regras de associação, clusterização, e padrões sequenciais.	Data-Miner PTy Ltd www.data-miner.com	Comercial
Cubist	Regressão	Rule Quest www.rulequest.com	Comercial
Darwin	Classificação, regressão e clusterização	Oracle Corp. www.oracle.com	Comercial
DataMite	Regras de associação	Dr. Phillip Varsey do LPA Prolog	Comercial
Intelligent Miner	Regras de associação, padrões sequenciais, classificação, clusterização, sumarização e modelagem de dependência	IBM Corp. www.ibm.com	Comercial
Microssoft data Analyzer	Classificação e clusterização	Microsoft Corp. www.microsoft.com	Comercial
Oracle Data Mining	Classificação e regras de associação	Oracle Corp. www.oracle.com	Comercial
Orion Data Mining Engine	Associação, classificação e clusterização	Grupo de Pesquisa em Ingeligencia Computacional Aplicada – Unesc http://www.unesc.net	Gratuita
PolyAnalyst	Classificação, regressão, associação, clusterização, sumarização e modelagem de dependência	Megapunter Intelligence www.megaputer.com	Comercial
WEKA	Classificação, regressão e regras de associação	University of Waikato http://www.cs.waikato.ac.nz	Gratuita

Fonte: Adaptado de Rezende (2005).

Grande parte das ferramentas disponíveis é comercial, sabendo disso o Grupo de Pesquisa em Inteligência Computacional Aplicada, do Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense (UNESC), formado por professores e acadêmicos da instituição, mantém em desenvolvimento o projeto da *Shell Orion Data Mining Engine*. Seus principais objetivos são: aplicar os conceitos de KDD descritos anteriormente, e disponibilizar no futuro uma *shell* gratuita para a comunidade acadêmica e sociedade em geral.

2.3 SHELL ORION DATA MINING ENGINE

O projeto da *Shell Orion* teve início no ano de 2005, e possui como característica o fato de que, todas as tarefas e métodos foram implementados por acadêmicos em seus respectivos Trabalhos de Conclusão de Curso (TCC).

A ferramenta possui atualmente três tarefas implementadas: associação, classificação e clusterização. A tabela 2 descreve a evolução da ferramenta

desenvolvida bem como os algoritmos e métodos implementados até o momento na *Shell Orion*.

Tabela 2. Evolução da Shell Orion Data Mining Engine

Ano	Tarefa	Método	Algoritmo	Atributos	Referência
2005	Associação	Regra de Associação	Apriori	Numérico	(CASAGRANDE, 2005)
2005	Classificação	Árvore de Decisão	ID3	Nominais	(PELEGRIM, 2005)
2007	Classificação	Árvore de Decisão	CART	Nominais e numéricos	(RAIMUNDO, 2007)
2007	Clusterização	Particionamento	K-Means	Numéricos	(MARTINS, 2007)
2007	Clusterização	Redes Neurais	Kohonen	Numéricos	(BORTOLOTTO, 2007)
2008	Clusterização	Lógica Fuzzy	Gustafson-kessel	Numéricos	(CASSETARI JUNIOR, 2008)
2009	Clusterização	Lógica Fuzzy	Gath-Geva	Numéricos	(PEREGO, 2009)
2009	Classificação	Árvore de Decisão	C4.5	Nominais e Numériocs	(MONDARDO, 2009)
2009	Classificação	Redes Neurais	RBF	Numéricos	(SCOTTI, 2010)
2010	Clusterização	Lógica Fuzzy	RCP	Numéricos	(CROTTI JUNIOR, 2010)
2010	Clusterização	Lógica Fuzzy	URCP	Numéricos	(CROTTI JUNIOR, 2010)
2010	Clusterização	Lógica Fuzzy	FCM	Numéricos	(CROTTI JUNIOR, 2010)
2011	Clusterização	Densidade	DBSCAN	Numéricos	(GAVA, 2011)
2012	Clusterização	Enxame – Colônia de formigas	SACA	Numéricos	(GHELLERE, 2012)
2012	Clusterização	Enxame – Colônia de formigas	Ant-Based Clustering	Numéricos	(GHELLERE, 2012)
2012	Clusterização	Enxame – Colônia de formigas	A²CA	Numéricos	(GHELLERE, 2012)
2012	Classificação	Bayesiano	Naive Bayes	Numéricos	(NOVASKI, 2012)

Fonte: Adaptado de Novaski (2012).

O desenvolvimento da *Shell Orion Data Mining Engine* é realizado utilizando-se a linguagem de programação Java, pois essa linguagem permite reutilização de código, possui ambientes de desenvolvimentos gratuitos, sendo independente de plataforma (PELEGRIM, 2005).

Outra vantagem considerável da utilização da plataforma Java para o desenvolvimento da *Shell Orion* é sua Interface de Programação de Aplicações (*Application Programming Interface* - API) denominada *Java Database Connectivity* (JDBC). Essa API permite a conexão a qualquer banco de dados que possua um *driver* disponível para tal Sistema Gerenciador de Banco de Dados (SGBD). Isso torna a ferramenta bastante flexível quanto a qual SGBD utilizar.

Desde 2005 vários TCC fizeram parte do projeto da *Shell Orion*, tendo-se dentre as tarefas implementadas a associação. O foco dessa pesquisa acadêmica é implementar o algoritmo *FP-Growth* no módulo de associação dessa ferramenta, com o intuito de ampliar as suas funcionalidades.

3 A TAREFA DE ASSOCIAÇÃO

A descoberta de regras de associação tem recebido atenção por parte de pesquisadores tanto na área acadêmica como em aplicações práticas. Isso ocorre devido sua aplicabilidade em problemas de negócio somado com sua compreensibilidade inerente, ou seja, até mesmo não especialistas em DM compreendem as regras geradas (MELANDA, 2004; SCHONHORST, 2010).

A tarefa de regras de associação foi apresentada inicialmente por Agrawal et al (1993, tradução nossa) a partir da análise dos itens presentes em compras de um supermercado, objetivando a descoberta de relações do tipo: “Um cliente que compra os produtos $A_1, A_2,..., A_n$, também comprará os produtos $B_1, B_2,..., B_n$ com probabilidade p%”. Esta tarefa busca quantificar o quanto a presença de um conjunto de atributos nos registros de uma base de dados implica na presença de outro conjunto distinto de atributos nos mesmos registros (AGRAWAL; SRIKANT, 1994, tradução nossa). Isso permite analisar os dados de indústrias e comércios varejistas para aprender sobre o comportamento de compras de seus clientes. O conhecimento adquirido pode ser usado para apoiar uma diversidade de aplicações relacionadas ao negócio como gerência de estoque; relacionamento com os clientes; disposição dos produtos nas prateleiras e gôndolas ou em um catálogo; e também em promoções de vendas (TAN; STEINBACH; KUMAR, 2009).

As regras de associação não estão restritas a aplicações no varejo, pois podem ser aplicadas em diversas áreas como na medicina, comércio eletrônico, geoprocessamento, detecção de fraudes, bioinformática, análise de dados científicos, entre outras (SCHONHORST, 2010).

Um exemplo de aplicação da tarefa de associação é na análise dos dados das ciências da Terra, onde os padrões descobertos podem revelar conexões importantes entre o oceano, a terra e os processos atmosféricos. Tais informações podem ajudar cientistas a desenvolver uma compreensão mais refinada de como os diferentes elementos do sistema da Terra interagem entre si (TAN; STEINBACH; KUMAR, 2009).

3.1 CONCEITOS E DEFINIÇÕES

A representação de uma regra de associação ocorre da seguinte forma: *LHS* ⇒ *RHS*, onde *LHS* e *RHS* são respectivamente o antecedente (*Left Hand Side*) e o consequente (*Right Hand Side*) da regra descoberta. Agrawal e Srikant (1994, tradução nossa) definem formalmente as regras de associação:

a) seja uma base de dados *D* composta por um conjunto de transações *T* = $\{t_1, t_2,..., t_n\}$ contendo *n* transações e por um conjunto de itens *A* = $\{a_1, a_2,..., a_n\}$, na qual cada transação $t_1 \in T$ é composta por um conjunto de itens (chamado *itemset*) tal que $t_i \subseteq A$. Uma transação t_i suporta o *itemset Y* se $Y \subseteq t_i$. O suporte *P(Y)* de um *itemset Y* representa a probabilidade da ocorrência do evento *Y*;

b) como já dito a regra de associação é uma implicação da forma *LHR* ⇒ *RHS*, em que *LHS*, *RHS* ⊂ *A* e *LHS* ∩ *RHS* = ∅. *A* regra *LHS* ⇒ *RHS* ocorre no conjunto de transações *T* com suporte, *sup*, se o percentual das transações em *T* que ocorre *LHS* ∪ *RHS* é atingido ou ultrapassado. A regra *LHS* ⇒ *RHS* tem confiança, *conf*, se em *conf%* das transações de *T* em que ocorre *LHS* ocorre também *RHS*, ou seja, a probabilidade condicional de *LHS* dado *RHS* é atingida ou ultrapassada.

O objetivo geral da tarefa de associação em DM é encontrar, a partir de um conjunto de transações *T*, todas as regras que associem a presença de um *itemset*, *X*, por exemplo, com qualquer outro *itemset* (*Y, Z,* etc). Entretanto, como *A* possui um total de *m* itens, o espaço de busca para a geração de todas as regras tende a ser exponencial (O(2^m)), uma vez que todos os itens podem constituir *itemsets*. Na prática, porém, nem todos os itens de *A* estão presentes nas transações de *T* e outros ocorrem em um número muito baixo de transações. Essa espacialidade é aproveitada pelas tarefas e métodos de DM, tornando-os eficientes e viáveis (MOTTA, 2010).

Segundo Alves (2007) para o entendimento do problema da procura de itens frequentes, os principais fundamentos precisam ser definidos, os quais são:

a) ***itemset*:** o tamanho de qualquer conjunto *X* é o número de elementos que possui. Sendo *N* o conjunto de *n* números naturais {*1, 2, 3, ..., n*}.

Então cada $x \in N$ pode ser chamado de item. Um subconjunto não vazio de *N* é chamado de um *itemset*. Por fim, chama-se conjunto de itens *X* com cardinalidade *K*= |*X*| de um *K-itemset*;

b) **transação:** dado um conjunto $I = \{i_1, i_2, ..., i_n\}$ contendo itens, uma transação *T* é definida como qualquer subconjunto de itens em *I*, ou seja, $T \subseteq I$. Como *T* é um conjunto, nele não pode haver elementos duplicados;

c) ***tid:*** uma base de dados *D* é um conjunto que possui *n* transações, então denomina-se *TID* o identificador único que rotula cada transação nesta base de dados;

d) **suporte:** é uma medida para a regra descoberta, calculada por meio da frequência da ocorrência dos *itemsets* envolvidos. Suporte de *X*, sup(*X*), é definido como a fração de todas as transações em *D* que suportam *X*. E suporte mínimo (**sup_min**) um valor pré-definido que representa o limite mínimo para *X*. Estabelecer valor mínimo para o suporte produz um filtro, onde as regras com valores suficientes para o suporte podem ser interessantes e dignas de atenção, pois se destacam quantitativamente das demais. Por outro lado, as regras com suporte abaixo do valor mínimo, podem representar uma ocorrência ao acaso.

   O suporte de uma regra de associação *X*, $A \Rightarrow B$, é a porcentagem das transações que contêm $A \cup B$ em relação ao total de transações *n* de *T*. Calcula-se o suporte com a fórmula (1):

$$sup(A \Rightarrow B) = sup(A \cup B) = \frac{\mathrm{n}(A \cup B)}{N} * 100 \qquad (1)$$

   sendo $n(A \cup B)$ o número de transações onde *A* e *B* ocorrem juntos e *N* o número total de transações analisadas;

e) **confiança:** trata-se de outra medida para a regra descoberta, esta calcula a força da regra. Assim, sendo *C* a confiança de uma regra de associação, $A \Rightarrow B$, *C* é na verdade a porcentagem das transações que contêm $A \cup B$ em relação a todas as transações de *T* que contêm *A*. Portanto, *C* é calculado pela probabilidade condicional *P*(*B*|*A*), ou seja,

a probabilidade de ocorrência de *B*, quando *A* ocorre. Seu cálculo é feito utilizando-se a fórmula (2):

$$conf(A \Rightarrow B) = prob(A\,|\mathrm{B}) = \frac{\sup(A \cup B)}{\sup(A)} = \frac{\mathrm{n}(A \cup B)}{\mathrm{n}(A)} * 100 \qquad (2)$$

sendo *n*(*A*) o número de transações nas quais *A* ocorre. A confiança representa a frequência de ocorrência de *A* dado *B*;

f) **regra forte:** enquanto que o suporte de uma regra indica o quanto esta é confiável, a confiança revela o quanto esta regra é aplicada. Para uma regra ser interessante é preciso ter suporte e confiança suficiente. Uma regra de associação é considerada forte quando possui suporte e confiança maiores ou iguais, respectivamente, aos limites pré-estabelecidos pelo usuário de suporte mínimo (**sup_min**) e de confiança mínima (**conf_min**).

3.1.1 Simplificando o Entendimento

A tarefa de descoberta de regras de associação consiste em encontrar conjuntos de itens que ocorram simultaneamente, de forma frequente e com certa confiança. Considere o exemplo de uma aplicação dessa tarefa na tabela 3. Os dados podem ser representados em um formato binário, onde cada coluna corresponde a um item e cada linha corresponde a uma transação. Por sua vez um item corresponde a uma variável binária cujo valor é um se o item estiver presente e zero se não estiver presente. Nesse exemplo a tarefa de associação tem por objetivo descobrir os produtos que são vendidos frequentemente juntos (TAN; STEINBACH; KUMAR, 2009).

Tabela 3. Exemplos de transações de cestas de compras

TID	Leite	Café	Cerveja	Pão	Manteiga	Arroz	Feijão
1	0	1	0	1	1	0	0
2	1	0	1	1	1	0	0
3	0	1	0	1	1	0	0
4	1	1	0	1	1	0	0
5	0	0	1	0	0	0	0
6	0	0	0	0	1	0	0
7	0	0	0	1	0	0	0
8	0	0	0	0	0	0	1
9	0	0	0	0	0	1	1
10	0	0	0	0	0	1	0

Fonte: Adaptado de Goldschimidt e Passos (2005).

Outra forma de representar a tabela 4 é no formato *basket* (tabela 5), onde a quantidade de itens que pode constar em um TID não é limitada ao número de atributos da relação (AGRAWAL et al, 1993, tradução nossa).

Tabela 4. Exemplos de transações de cestas de compras no formato *basket*

TID	Itens
1	{Café, Pão, Manteiga}
2	{Leite, Cerveja, Pão, Manteiga}
3	{Café, Pão, Manteiga}
4	{Leite, Café, Pão, Manteiga}
5	{Cerveja}
6	{Manteiga}
7	{Pão}
8	{Feijão}
9	{Arroz, Feijão}
10	{Arroz}

Fonte: Adaptado de Tan, Steinbach e Kumar (2009).

O formato *basket* otimiza o espaço que os itens não presentes deixam. Isso é aproveitado pela tarefa de associação em DM, e torna os algoritmos eficientes e viáveis (MOTTA, 2010).

Analise a regra {Pão, Manteiga} ⇒ {Café}. A frequência do suporte para {Pão, Manteiga, Café} é 3 e o total de transações (TID) é 10, portanto o suporte para a regra é assim calculado:

$$sup(\{\text{Pão}, \text{Manteiga}\} \Rightarrow \{\text{Café}\}) = sup(\{\text{Pão}, \text{Manteiga}, \text{Café}\}) = \frac{3}{10} * 100 = 30\%$$

se o suporte mínimo (**sup_min**) estabelecido na aplicação for menor ou igual a 30%

então a regra {Pão, Manteiga} ⇒ {Café} é considerada frequente. Já a confiança é obtida dividindo-se o contador de suporte para {Pão, Manteiga, Café} pelo contador de suporte de {Pão, Manteiga}. A confiança da regra se calcula assim:

$$conf(\{\text{Pão, Manteiga}\} \Rightarrow \{\text{Café}\}) = prob(\{\text{Pão, Manteiga}\}|\{\text{Café}\})$$
$$= \frac{\sup(\{\text{Pão, Manteiga}\} \Rightarrow \{\text{Café}\})}{\sup(\{\text{Pão, Manteiga}\})} = \frac{3}{4} * 100 = 75\%$$

se o valor estabelecido na aplicação para a confiança mínima for menor ou igual a 75% então a regra de associação {Pão, Manteiga} ⇒ {Café} é considerada válida. A medida de confiança qualifica uma regra, e indica o quanto a ocorrência do antecedente da regra assegura a ocorrência do consequente. Defini-se *K-itemset* todo conjunto de itens com exatamente *K* elementos. Assim como a regra {Pão, Manteiga} ⇒ {Café} possui 3 elementos ela corresponde a 3-*itemsets* (GOLDSCHIMIDT; PASSOS, 2005; TAN; STEINBACH; KUMAR, 2009).

3.2 DEFINIÇÃO DO PROBLEMA

O problema da descoberta de regras de associação em DM é definido da seguinte forma: dado um conjunto de transações T, encontre todas as regras que tenham suporte >= *min_sup* e confiança >= *min_conf*, onde *min_sup* e *min_conf* são os limites de confiança e suporte pré-estabelecidos (SCHONHORST, 2010).

Uma forma de procurar as regras de associação em uma base de dados é por meio do calculo do suporte e confiança para cada possível regra. Porém, essa abordagem devido ao grande número de regras que podem ser extraídas, é custosa e, portanto não deve ser usada. Para se ter uma ideia, o número total de regras possíveis extraídas de uma base de dados que contenha *n* itens é calculada por meio da fórmula (3):

$$R = 3^n - 2^{n+1} + 1 \quad (3)$$

até mesmo para o conjunto de dados pequeno mostrado na tabela 4, esta abordagem necessita que seja calculado o suporte e a confiança para:

$$R = 3^7 - 2^{7+1} + 1 = 1932$$

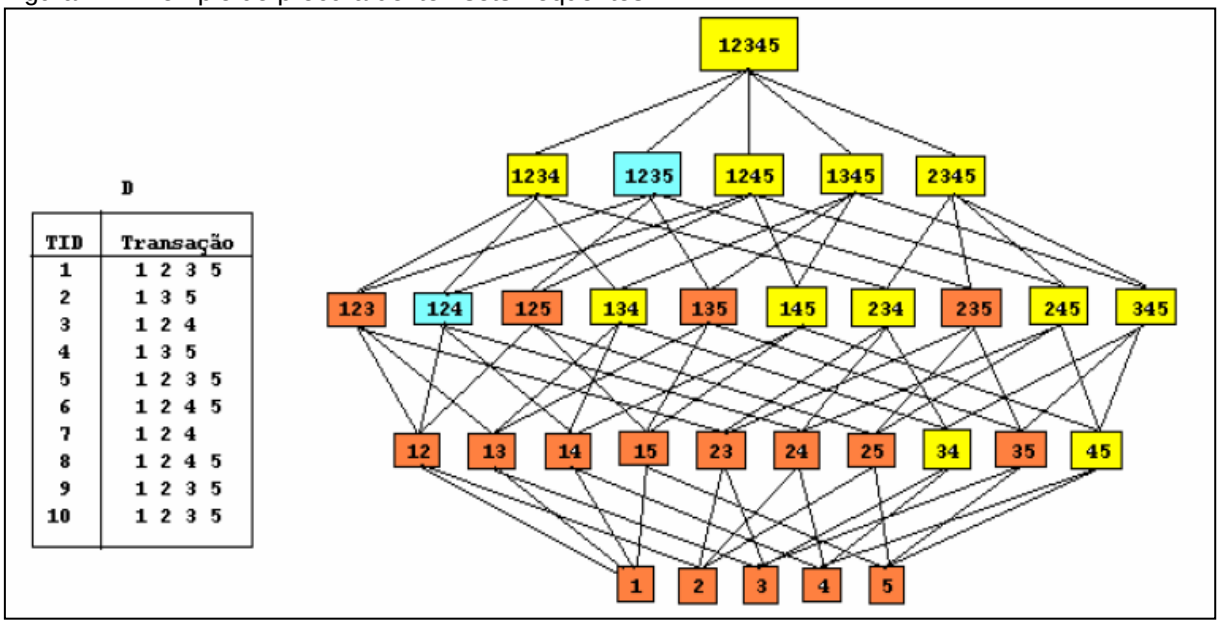
regras. Após a aplicação de min_sup = 30% e min_conf = 60%, verifica-se que mais de 70% das regras são descartadas, ou seja, a maioria dos cálculos feitos para se medir a confiança da regra gerada são desnecessários. Visando a melhoria do desempenho dos algoritmos de descoberta de regras de associação os requisitos de suporte e confiança são calculados separadamente, isso evita a execução de cálculos desnecessários, pois permite a poda de regras que não satisfazem o suporte mínimo (**min_sup**), sem ter que calcular os seus respectivos valores de confiança (TAN; STEINBACH; KUMAR, 2009).

Agrawal et al (1993, tradução nossa) define o problema de extração de regras de associação em duas fases: as quais compreendem a geração de conjuntos de *itemsets* frequentes e de regras de associação.

3.2.1 Geração de Conjuntos de *Itemsets* Frequentes

A figura 4 ilustra e ratifica o conceito já mencionado: o número de candidatos a conjuntos frequentes cresce exponencialmente com o número de itens considerados. Essa fase requer mais recursos computacionais, pois é mais custosa. Os algoritmos de associação realizam uma busca exaustiva e testam todos os componentes do espaço de busca para verificar se estes são ou não frequentes (ALVES, 2007).

Figura 4 – Exemplo de procura de itemsets frequentes.



Fonte: Alves (2007).

De acordo com Motta (2010) encontrar todas as regras que satisfaçam o limite do *min_sup* é essencial, pois a simples verificação de uma regra em particular pode desconsiderar regras novas e ou mudanças nas tendências do universo que se está analisando. Também outros métodos, como por exemplo, relacionados a alguma forma de amostragem, podem facilmente não corresponder a uma correta representação do universo global, comprometendo os resultados.

Nessa fase a maioria dos algoritmos cria um conjunto de itens prováveis, chamados de conjunto de candidatos[2], que podem ou não ser frequentes. Para descobrir quais destes candidatos são realmente frequentes são realizadas contagens sobre a base de dados. Tal tarefa requer um tempo de processamento considerável e normalmente grande quantidade de memória, por isso os algoritmos são desenvolvidos ou melhorados objetivando a redução do número de candidatos gerados. A quantidade de candidatos, a frequência e a sequencia que são criadas dependem, em grande parte, da solução dada por cada algoritmo (ALVES, 2007).

3.2.2 Geração de Regras de Associação

Após a contagem e o cálculo dos respectivos suportes estarem disponíveis, o conjunto de *itemsets* gerados no passo anterior (item 3.2.1) pode ser

[2] Conjuntos candidatos são conjuntos de itens com a possibilidade de serem frequentes e fortes. São gerados a partir de conjuntos frequentes até então encontrados (ALVES, 2007).

avaliado e suas confianças determinadas. Dependendo do nível de confiança mínima estabelecido na aplicação, a regra é classificada como forte ou, então, é simplesmente descartada. O calculo da confiança em comparação com a procura e determinação dos N *itemsets* frequentes é direto, pois nessa fase não há necessidade de se ler o conjunto de dados inteiro novamente e sim que se executem os cálculos para se obter o valor de confiança para cada N *itemset* (SCHONHORST, 2010).

Utiliza-se diferentes algoritmos para se obter de uma base de dados os N *itemsets* frequentes e gerar regras de associação fortes, entre os principais algoritmos (TAN; STEINBACH; KUMAR, 2009) tem-se: *Apriori* (AGRAWAL; SRIKANT, 1994) e *FP-Growth* (HAN et al, 2004; HAN; PEI; YIN, 2000).

3.3 O ALGORITMO *APRIORI*

O nome *Apriori* se baseia no fato de que o algoritmo usa o conhecimento prévio de propriedades do conjunto de itens frequentes. O *Apriori* é um algoritmo clássico e considerado um dos maiores avanços na tecnologia de descoberta de regras de associação em DM (HAN; KAMBER, 2006, tradução nossa). Diversos algoritmos foram inspirados no funcionamento do *Apriori*, os principais são: *Generalized Sequential Patterns*[3] (GSP); *Direct Hashing and Pruning*[4] (DHP); *Partition*[5]*; Eclat*[6] (GOLDSCHIMIDT; PASSOS, 2005; HAN; KAMBER, 2006, tradução nossa).

O funcionamento do *Apriori* se baseia no princípio da antimonotonicidade do suporte. O princípio é definido resumidamente assim: um *k-itemset* somente pode ser frequente se todos os seus *(k-1)-itemsets* também forem frequentes, ou seja, a

[3] O algoritmo GSP foi desenvolvido para encontrar padrões sequenciais em um conjunto de dados. Cada sequência é uma lista de transações efetuadas durante um período de tempo, e cada transação é um conjunto de *itemsets*. O problema consiste em descobrir todos os padrões sequenciais com um suporte mínimo especificado pelo usuário, onde o suporte de um padrão é o número de sequências que contêm tal padrão (SRIKANT; AGRAWAL, 1996, tradução nossa).

[4] DHP também tem como base o algoritmo *Apriori.* Esse algoritmo possui duas características principais que objetivam resolver alguns problemas do *Apriori*: uma é a geração de candidatos utilizando uma tabela *hash* e a outra é a redução do numero de candidatos por meio de poda (PARK; CHEN; YU, 1995, tradução nossa).

[5] O algoritmo *Partition* é mais um algoritmo semelhante ao *Apriori*. Ele utiliza intersecções entre conjuntos para calcular os valores de suporte. *Partition* também divide a base de dados em várias partes que são tratadas de maneira independente, isso é feito objetivando um melhor aproveitamento da memória física disponível (SAVARESE; OMIECINSKI; NAVATHE, 1995).

[6] *Eclat* combina busca em profundidade com intersecção entre conjuntos. Desta forma, não é necessário dividir a base de dados como no caso do algoritmo *Partition* (ZAKI et al, 1997).

combinação de *itemsets* para gerar um novo *itemset* ocorre somente quando estes já são frequentes. Tal princípio possibilitou ao *Apriori* reduzir significativamente o número de conjuntos candidatos, pois quanto maior o número de conjuntos candidatos maior será também a necessidade de processamento para encontrar os conjuntos de itens frequentes (AGRAWAL; SRIKANT, 1994, tradução nossa).

Para uma explicação detalhada do algoritmo *Apriori* veja o trabalho de conclusão de curso do acadêmico Diego Paz Casagrande que implementou esse algoritmo no módulo de associação da *Shell Orion* no ano de 2005 (CASAGRANDE, 2005).

A complexidade de um sistema de descoberta de regras de associação é dependente do algoritmo que está sendo utilizado (MELANDA, 2004; TAN; STEINBACH; KUMAR, 2009).

3.3.1 Complexidade Computacional

Conforme Tan, Steinback e Kumar (2009), apesar dos avanços significativos na melhora de desempenho, a complexidade computacional do algoritmo *Apriori* ainda pode ser afetada por fatores relevantes, dentre os quais se destacam:

a) **limite de suporte:** diminuir o limite de suporte muitas vezes acarreta em mais conjuntos de itens candidatos sendo declarados frequentes. Isto tem um efeito indesejado sobre a complexidade computacional do algoritmo porque mais *itemsets* devem ser gerados, contados e testados. Também o tamanho máximo dos conjuntos de itens frequentes tende a ser maior com limites de suportes mais baixos, com isso o algoritmo *Apriori* terá que fazer mais passagens sobre o conjunto de dados;

b) **dimensionalidade:** refere-se ao número de itens, quanto maior este for, maior será o espaço necessário para armazenar os contadores de suporte dos itens. Se o número de *itemsets* candidatos gerados também crescer, os custos computacionais aumentarão;

c) **número de transações:** como o algoritmo *Apriori* executa passagens repetidas pela base de dados, o tempo necessário para a sua execução aumenta com um maior número de transações;

d) **extensão média da transação:** em conjuntos de dados densos, a extensão média da transação pode ser grande o bastante para afetar a complexidade do algoritmo *Apriori* por dois motivos: um é o tamanho máximo dos conjuntos de dados frequentes que tende a aumentar à medida que a extensão média da transação aumenta. Isso resulta em mais conjuntos de *itemsets* candidatos sendo examinados durante a geração de candidatos e a contagem de suporte; o outro ocorre devido a extensão da transação ser maior, onde mais conjuntos de dados estão contidos na mesma. Isso implicará em um número maior de percursos na árvore *hash* durante a contagem de suporte.

Com o objetivo de superar essas limitações e melhorar a eficiência da descoberta de regras de associação diversos métodos e algoritmos alternativos têm sido desenvolvidos. Aí surge uma pergunta: pode-se criar um método ou um algoritmo que gera o conjunto completo de *itemsets* frequentes a partir de uma base de dados sem para isso gerar candidatos? (HAN; KAMBER, 2006, tradução nossa). O próximo capítulo responderá essa pergunta, e apresentará o algoritmo *FP-Growth*.

4 O ALGORITMO *FP-GROWTH*

Este capítulo apresenta o algoritmo *FP-Growth* desenvolvido por Han, Pei e Yin (2000). O algoritmo *FP-Growth* utiliza uma abordagem diferente do *Apriori* para procurar e descobrir conjuntos de *itemsets* frequentes, já que, não concorda com o paradigma de gerar e testar do *Apriori*. Portanto ele cria uma estrutura de dados para codificar todo o conjunto de dados de forma compacta. Após isso, o *FP-Growth* extrai conjuntos de itens frequentes diretamente desta estrutura (TAN; STEINBACH; KUMAR, 2009).

De acordo com Han, Pei e Yin (2000, tradução nossa) a eficiência do algoritmo *FP-Growth* é atingida, e o problema da descoberta de regras de associação é resolvido, por meio da aplicação das seguintes técnicas:

a) inicialmente é construída uma estrutura de dados, um tipo especializado de árvore compacta, chamada *frequent-pattern tree* (*FP-tree*), que resume a base de dados original. Nela são armazenadas informações que quantificam os padrões frequentes. Com o intuito de garantir que a estrutura de dados seja informativa e compacta, somente os 1-*itemsets* irão compor os nós da árvore e esses serão organizados de tal maneira que os nós que ocorrem mais frequentemente terão maiores possibilidades de serem compartilhados com vários outros nós em seus ramos. Experimentos mostram que *FP-tree* é realmente compacta, sendo menor que a base de dados original. Outra vantagem é que após a construção da árvore o algoritmo *FP-Growth* não precisa mais efetuar leituras da base de dados, pois irá extrair os conjuntos de itens frequentes diretamente da *FP-tree*;
b) durante a construção da *FP-tree*, o algoritmo *FP-Growth* cria um método de organização das informações analisadas. Tal método se baseia no crescimento contínuo e segue um padrão onde a estrutura de dados cresce em fragmentos, ou seja, cada transação é analisada e acrescentada à estrutura. Isso ocorre até a leitura da última transação. Devido a ordenação dos itens ocorrer de forma decrescente pelo valor de suporte, cria-se uma tendência de que os itens de transações com maior número de ocorrência na árvore fiquem nos ramos de forma *top-down*, isso contribui para um melhor aproveitamento dos ramos

inseridos na estrutura e torna a árvore compacta. Também evita a onerosa etapa de geração e teste de candidatos, pois não necessita que um *itemset* descoberto seja armazenado para consultas em iterações posteriores;

c) por fim o algoritmo *FP-Growth* emprega uma técnica de busca que se baseia no uso da árvore *FP-tree* explorando-a de baixo para cima, ou seja, no formato *bottom-up*. Com isso, ele encontra todos os conjuntos de itens frequentes, terminando com um prefixo específico e empregando uma outra estratégia: a de dividir para conquistar. Faz isso, fracionando o problema em subproblemas, uma vez que a *FP-tree* é dividida em sub-árvores condicionais. Logo após, o *FP-Growth* realiza a análise de cada FP-*tree* condicional gerada fazendo chamadas recursivas na mesma. Ao término da análise todos os conjuntos de itens frequentes são encontrados.

O Algoritmo *FP-Growth* apresenta maior complexidade de implementação em comparação com os algoritmos da família *Apriori*, no entanto esta complexidade adicional é compensada pelo seu melhor desempenho, pois o *FP-Growth* é considerado o algoritmo de associação mais eficiente já desenvolvido, sendo a base dos melhores algoritmos desenvolvidos posteriormente (BORGELT, 2005, tradução nossa; SOUSA, 2009). Os motivos de tal complexidade e do melhor desempenho serão descritos em detalhes nas seções seguintes. A próxima descreve a estrutura *FP-tree* e os passos para a sua construção.

4.1 *FP-TREE*: PROJETO E CONTRUÇÃO

A construção da estrutura *FP-tree* usa uma estratégia que faz com que os itens de todas as transações percorridas fiquem organizados da forma mais compacta possível, pois evita a criação de ramos desnecessários na árvore FP. Isto é possível devido a forma que os itens frequentes são inseridos, onde cada transação é mapeada em um caminho na árvore. Como diferentes transações podem possuir diversos itens em comum, seus caminhos podem se sobrepor, considerando que para cada caminho ainda não mapeado na *FP-tree* é criado um novo ramo, quanto mais os caminhos se sobrepõem, mais são aproveitados ramos

existentes colaborando assim para uma maior compressão da estrutura criada (HAN et al, 2004, tradução nossa; TAN; STEINBACH; KUMAR, 2009; XAVIER, 2010).

Para construir a *FP-tree* são necessárias duas leituras de todas as transações da base de dados. Na primeira leitura são levantados os valores de suporte de todos os itens de tamanho 1 (1-*itemset*). Os itens frequentes são posicionados em ordem decrescente pelo valor de suporte, enquanto que os infrequentes são descartados, uma vez que eles não podem ser parte de um conjunto de itens frequentes (HAN et al, 2004, tradução nossa; HAN; PEI; YIN, 2000, tradução nossa).

Na tabela 5 tem-se uma representação dessa primeira leitura, considerando-se para este exemplo, uma base de dados contendo transações e um valor para o suporte mínimo igual a três. A esquerda tem-se os itens comprados que compõe a base de dados original. Na parte do meio são listados os itens com seus respectivos valores de suporte ordenados de forma decrescentemente. E a direita é listada as transações com os itens ordenados em forma decrescente em relação ao valor de suporte e os itens infrequentes {f} e {g} são descartados.

Tabela 5. Exemplos de transações.

TID	Itens Comprados			Itens frequentes (Ordenados)
100	a, d, f			d, a
200	a, c, d, e	d	8	d, c, a, e
300	b, d	b	7	d, b
400	b, c, d	c	5	d, b, c
500	b, c	a	4	b, c
600	a, b, d	e	3	d, b, a
700	b, d, e	f	2	d, b, e
800	b, c, e, g	g	1	b, c, e
900	c, d, f			d, c
1000	a, b, d			d, b, a

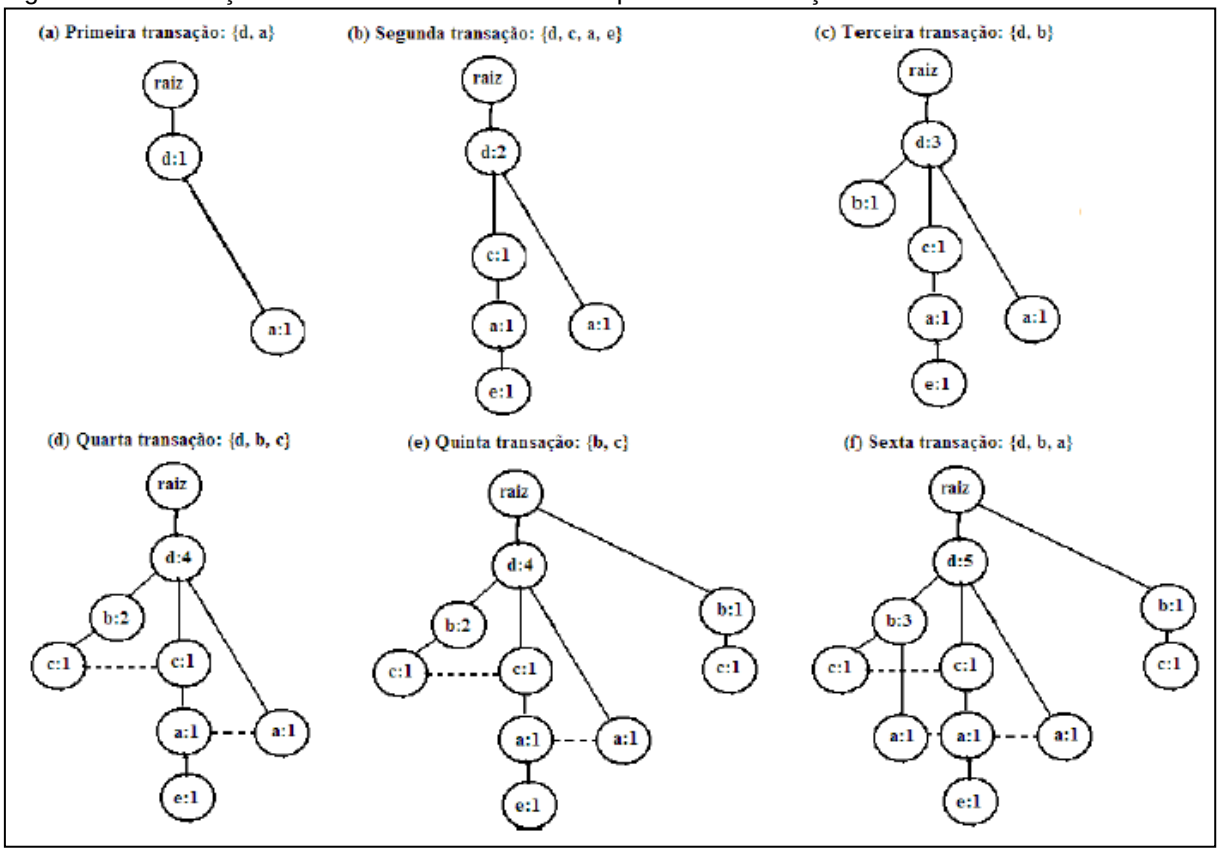
Fonte: Adaptado de Borgelt (2005, tradução nossa).

Finalizado a primeira leitura o método que monta a *PF-tree* cria o nó raiz e o rotula com *null*. Então, inicia-se a segunda leitura da base de dados para cada transação, a qual é acrescentada a estrutura *FP-tree*. Ao final desta leitura o método retorna a árvore construída.

Baseando-se na tabela 5 a figura 5 mostra a segunda leitura das seis primeiras transações. A árvore *FP-tree* é então estendida da seguinte forma:

a) a leitura da primeira transação leva à construção do primeiro ramo da árvore: (*d:1*), (*a:1*). Note que os itens frequentes são listados de acordo com a sua respectiva frequência, conforme mostra a figura 5(a);
b) para a segunda transação, figura 5(b), tem-se a sua respectiva lista de itens frequentes ordenados: *d, c, a, e*. O método então verifica que existe prefixo em comum ao caminho já existente: *d, a*. Por isso a contagem de cada nó ao longo do prefixo é incrementado em um, neste caso somente o item d é incrementado. Para os demais itens desta operação um novo nó é criado para cada item, tendo-se o (*c:1*) que ligado como filho de (*d:2*), o nó (*a:1*) vinculado como filho de (*c:1*), e o nó (*e:1*) vinculado a (*a:1*);
c) na terceira transação, figura 5(c), os itens frequentes ordenados são: *d, b*. Assim como na transação anterior, apenas o item *d* possui prefixo em um caminho da árvore, por isso seu contador é incrementando, para o item (*b:1*) um novo nó é criado e vinculado como filho de (*d:3*);
d) para os itens da quarta transação, *d, b, c*, existe prefixo em comum para os itens {*d, b*}, assim seus contadores são incrementados em um, ficando apenas o último item *c* sem caminho existente, consequentemente é criado um novo nó (*c1*) e ligado como um filho de (*b:2*), conforme a figura 5(d);
e) na quinta transação o método verifica que não existe prefixo na árvore para o conjunto de itens *b, c*. Portanto, um novo ramo a partir da raiz da árvore é criado e iniciam-se os contadores com um (figura 5(e));
f) na sexta transação, figura 5(e), o método identifica a existência de prefixo para os itens {*d, b*}, então incrementa seus contadores em um, para o item {*a*} um novo nó (*a:1*) é criado e vinculado como filho de (*b:3*).

Figura 5 – Construção da *FP-Tree*: Leitura das seis primeiras transações.

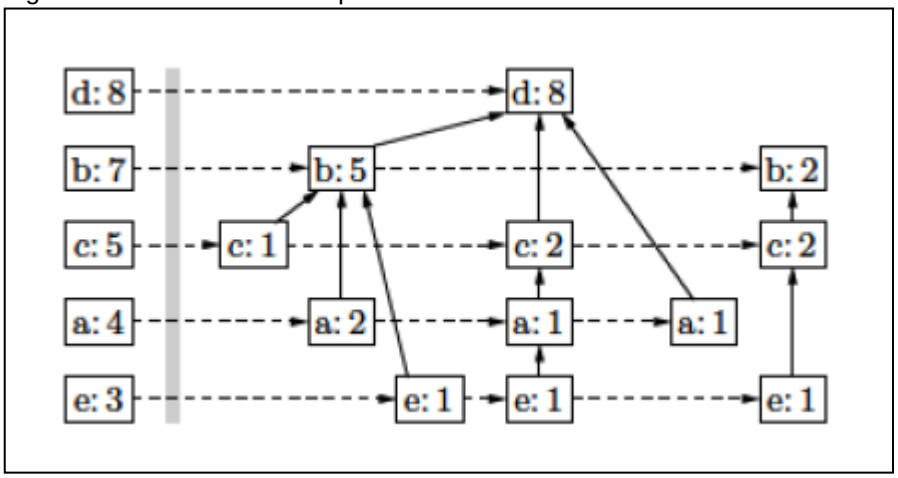


Fonte: Xavier (2010).

O método de construção da árvore *FP-tree* usa a mesma lógica até o final da leitura da última transação. O algoritmo *FP-Growth* cria uma tendência onde os itens com maior ocorrência ficam nos ramos da árvore de forma *top-down*, isso ocorre em virtude da ordenação dos itens pelo seu respectivo valor de suporte em ordem decrescente, consequentemente contribui para que sejam melhores aproveitados os ramos da estrutura criada de forma que fiquem mais compactas.

Afim de facilitar o percurso do algoritmo *FP-Growth* pela árvore, uma tabela de cabeçalho é construída. Sua função é armazenar o endereço da primeira ocorrência do respectivo *itemset* na árvore. Isso permite ao algoritmo encontrar os outros *itemsets* com o mesmo item nome devido ao *itemset* na árvore FP ter um campo que armazena o endereço do próximo item com o mesmo item nome ou nulo caso não tenha mais nenhum (HAN et al, 2004, tradução nossa; HAN; PEI; YIN, 2000, tradução nossa). A figura 6 mostra a *FP-tree* criada a partir da tabela 5.

Figura 6 – *FP-Tree* criada a partir da tabela 5.



Fonte: Borgelt (2005).

Conforme Han et al (2004, tradução nossa) e Han, Pei e Yin (2000, tradução nossa) com base em um exemplo que explique todo o processo de criação de uma árvore *PF-tree*, seu processo pode ser projetado e definido como segue:

a) trata-se de uma estrutura de dados com um nó raiz rotulado com *null*, um conjunto de sub-árvores de itens, os quais possuem prefixos em comum e são ligados como filho do nó raiz, e uma tabela de cabeçalho que armazena os itens frequentes;

b) cada nó N na *FP-tree* é constituído por três campos distintos:

– **nome do item**: identifica o item que este nó representa,

– **contador**: memoriza o número de transações em que tal item aparece na base de dados,

– **nó *link***: faz uma ligação para o próximo nó na *FP-tree* que leva o mesmo nome item;

c) cada registro da tabela de cabeçalhos de itens frequentes consiste em dois campos: um que armazena o nome do item e outro o cabeçalho do nó link. Assim, armazena um ponteiro para o primeiro nó na *FP-tree* que possui o mesmo nome do item a que este ponteiro se refere.

Baseando-se nesses conceitos o método responsável pela criação da árvore *FP-tree* do algoritmo *FP-Growth* foi desenvolvido (figura 7).

Figura 7 – Algoritmo *FP-GROWTH*: método de construção da *FP-tree.*

```
Algoritmo: FP-Growth_ConstroiArvore
Entradas: Base de dados D; minsup
Saída: FP-tree T

1)   F := 1-itemsets_frequentes( D, minsup );
2)   L = ordena_decrescente( F );
3)   criaNoh( T , null );
4)   para cada transação Trans em D {
5)        [p|P] = ordena( Trans, L );
6)        insere_tree( [p|P], T );
7)   }
8)   retorne T;
```

Fonte: Han et al (2004, tradução nossa) e Han, Pei e Yin (2000, tradução nossa).

O processo de construção da estrutura *FP-tree* começa com a primeira leitura completa da base de dados, transação por transação, o objetivo é levantar todos os *1-itemsets* frequentes (linha 1). Na linha 2 uma lista ordenada *L* é criada com os *itemsets* em ordem decrescente de suporte. Após a definição do conjunto de 1-*itemsets*, inicia-se o processo de inclusão dos mesmos na árvore, para isso é criado o nó raiz e associado *null* ao seu rótulo (linha 3). A seguir, a segunda leitura na base de dados é realizada. Cada transação é então processada em duas etapas (linhas 4 a 7). A primeira etapa ordena os *itemsets* de acordo com a lista *L* (linha 5). E a outra etapa, linha 6, faz uso da função *insere_tree* (figura 8), que é responsável pela inserção dos *itemsets* da transação atual na *FP-tree*.

Figura 8 – Algoritmo *FP-GROWTH*: função *insere_tree*.

```
Função: insere_tree( [p|P], N )
Entradas: Conjunto ordenado [p|P], Nó N
Saída: -

1)   se ( N tem um filho F tal que F.nome-item = p.nome-item ) {
2)        F.contador++;
3)        Noh_filho = F;
4)   }
5)   else {
6)        Noh_filho = criaNoh( T, p.nome-item );
7)        Noh_filho.contador = 1;
8)        Noh_filho.pai = N;
9)        atualizaTabelaCabeçalhos( Noh_filho );
10)  }
11)  P.remove( p );
12)  se ( P é não-vazio )
13)       insere_tree( P, Noh_filho );
```

Fonte: Adaptado de Han et al (2004, tradução nossa) e Han, Pei e Yin (2000, tradução nossa).

A chamada da função *insere_tree* é feita pelo algoritmo responsável pela construção da *FP-tree* (figura 7), passando como parâmetros o conjunto de *itemsets* da transação atual representado por *[p|P]*, onde *p* é o primeiro elemento da lista e *P* é o conjunto de elementos remanescentes, e o nó raiz da árvore *N*. Na linha 1, a função *insere_tree* verifica se *p* (o primeiro elemento da lista) é um dos filhos do nó raiz *N*. Se for, seu contador de suporte é incrementado (linha 2). Caso contrário, é necessária a criação de um novo nó na *FP-tree* para tal *itemset* e ligá-lo como filho do nó *N*. Além disso, é feita a atualização da tabela de cabeçalhos (*Header Table*) para ligá-lo aos outros *itemsets* que possuem o mesmo nome item (linhas 5 a 10). Caso ainda existam itens em *P*, a função *insere_tree* é chamada recursivamente, e terá como parâmetros os itens remanescentes e o nó que fora atualizado ou adicionando como nó pai da nova chamada recursiva (linhas 12 e 13). Quando não houver mais itens remanescentes, ou seja, o conjunto *P* estiver vazio, a recursão encerra indicando que todos os itens da transação em questão foram processados e corretamente inseridos na *FP-tree*.

Ao termino da leitura e processamento a estrutura *FP-tree* estará criada, representando de forma compacta a base de dados original, podendo esta estrutura ser utilizada pelo algoritmo *FP-Growth* na etapa de descoberta dos padrões frequentes.

4.2 EXTRAÇÃO DE PADRÕES FREQUENTES UTILIZANDO A *FP-TREE*

O algoritmo *FP-Growth* extrai padrões frequentes presentes em uma base de dados a partir da estrutura *FP-tree,* explorando-a de baixo para cima. Seu funcionamento é baseado na seguinte regra: sejam *X* e *Y* dois conjuntos de *itemsets*. O suporte de $X \cup Y$ será igual ao suporte de *Y*, considerando apenas as transações da base de dados que contêm *X*. Essa projeção da base de dados recebe o nome de base de dados condicional de X (*X's conditional pattern base*) e à *FP-tree* que corresponde a essa projeção da base de dados é dada o nome de *FP-tree* condicional de X (*X's conditional FP-tree*).

De um modo geral, o processo consiste no desenvolvimento de uma estratégia baseada na técnica de dividir para conquistar, onde o problema é fracionado em subproblemas. O algoritmo faz isso considerando cada item da tabela de cabeçalhos. O primeiro item a ser processado é o último, o segundo o penúltimo item e assim sucessivamente até o término do processamento de todos os itens. Para cada *itemset* presente na tabela de cabeçalhos, bases de padrões condicionais são geradas, as quais são utilizadas para a construção das *FP-tree* condicionais, que relacionam os caminhos frequentes que se conectam aos nós correspondentes ao *itemset* em questão. Uma vez criadas, as *FP-tree* condicionais são utilizadas para encontrar os padrões frequentes que apresentam o *itemset* como sufixo (HAN et al, 2004, tradução nossa; HAN; PEI; YIN, 2000, tradução nossa).

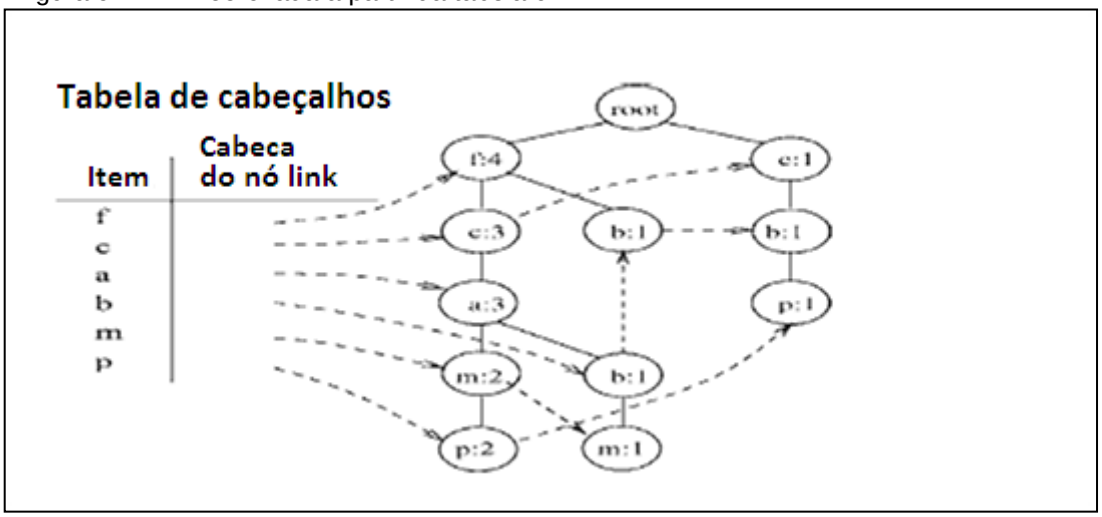
Tabela 6. Exemplos de transações de uma base de dados.

TID	Itens Comprados	Itens frequentes (Ordenados)
100	f,a,c,d,g,i,m,p	f,c,a,m,p
200	a,b,c,f,l,m,o	f,c,a,b,m
300	b,f,h,j,o	f,b
400	b,c,k,s,p	c,b,p
500	a,f,c,e,l,p,m,n	f,c,a,m,p

Fonte: Han et al (2004, tradução nossa) e Han, Pei e Yin (2000, tradução nossa).

A seguir, o funcionamento do algoritmo *FP-growth* é ilustrado por meio de um exemplo. Considere a *FP-tree* extraída da tabela 6 e apresentada na figura 9, onde o suporte mínimo é igual a três transações.

Figura 9 – *FP-Tree* criada a partir da tabela 6.



Fonte: Han et al (2004, tradução nossa) e Han, Pei e Yin (2000, tradução nossa).

O algoritmo procura conjuntos de *itemsets* frequentes terminando em *p* primeiro, seguido por *m*, *b*, *a*, *c*, e finalmente, *f*. Faz isso examinando apenas os caminhos que estão ligados ao nó da tabela de cabeçalhos que está sendo analisado. Estes caminhos podem ser acessados rapidamente usando os ponteiros associados ao nó em questão.

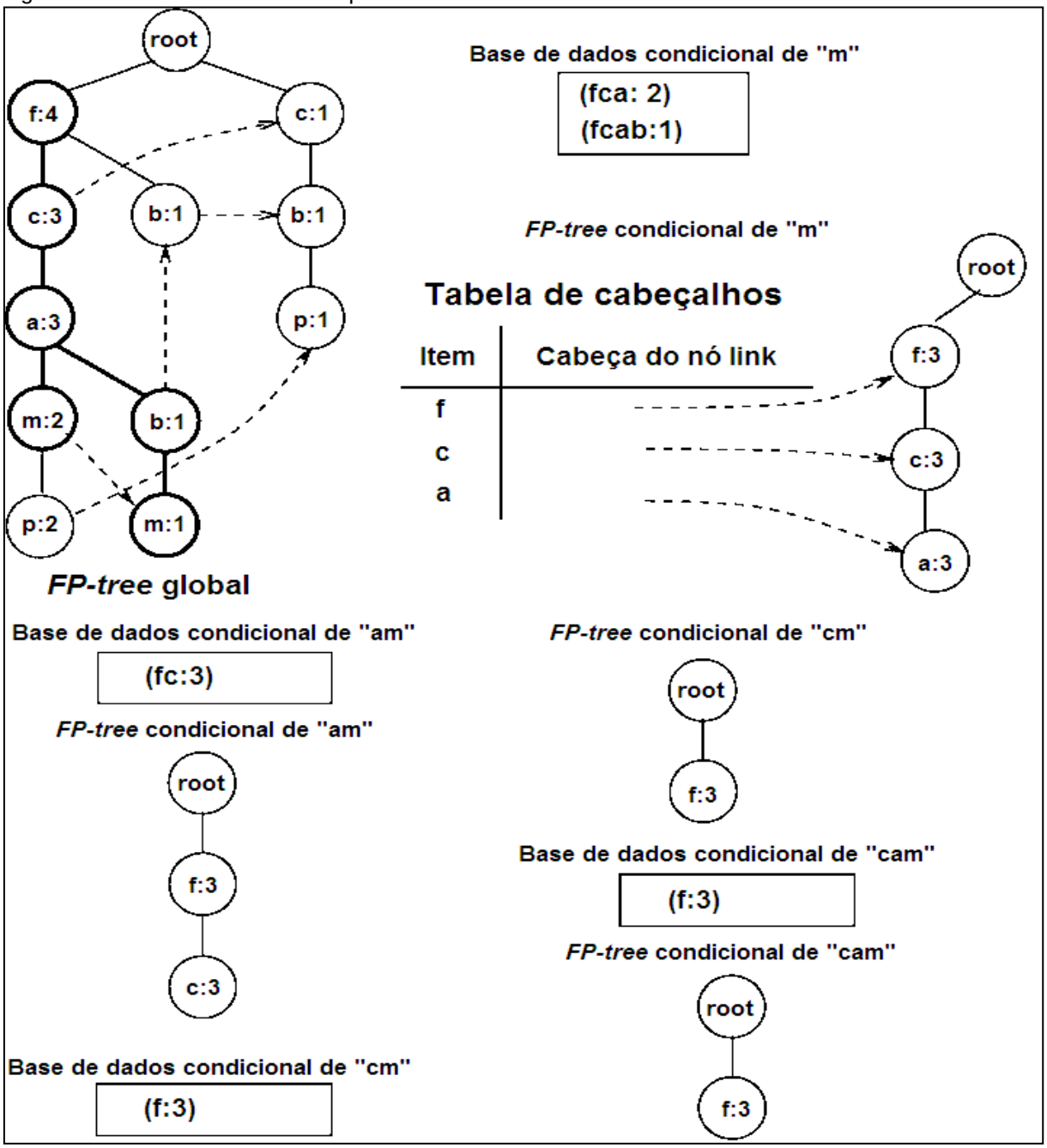
Para o nó *p*, o algoritmo verifica que ele é frequente (p:3), e que possui dois caminhos na *FP-tree*: {*f:4, c:3, a:3, m:2, p:2*} e {*c:1, b:1, p:1*}. O primeiro caminho indica que a sequência {*f, c, a, m, p*} aparece duas vezes na base de dados. Observe também que o caminho indica que os *itemsets* {*f, c, a*} aparecem três vezes e o *itemset* {*f*} aparece quatro vezes. Porém, só aparecem duas vezes em conjunto com *p*. Assim, para analisar os *itemsets* que aparecem simultaneamente com o *itemset p*, os contadores de suporte do caminho de prefixo devem ser atualizados para: {*f:2, c:2, a:2, m2*}. Da mesma forma, o segundo caminho indica que os *itemsets* {*c, b, p*} aparecem uma vez em conjunto nas transações da base de dados, ou seja, o caminho de prefixo para *p* é {*cb: 1*}. Estes dois caminhos de prefixo para *p* – {(fcam: 2), (cb: 1)} – formam a base de dados condicional de *p* (*p's conditional pattern base*), ou seja, a base de sub padrões sob a condição da existência de *p*. A *FP-tree* que corresponde a projeção de *p* (*p's conditional FP-tree*)

tem apenas um ramo {*c:3*}. Portanto deriva um único *itemset* {cp:3}, tendo-se a busca por padrões frequentes associados a *p* finalizada (HAN et al, 2004, tradução nossa; HAN; PEI; YIN, 2000, tradução nossa).

Agora o algoritmo *FP-Growth* passa a analisar o nó *m*, verifica imediatamente que o seu padrão é frequente {m, 3}, e que tem dois caminhos, {*f:4, c:3, a:3, m:2*} e {*f:4, c:3, a:3, b:1, m:1*}. Observa-se que *p* aparece juntamente com *m*, no entanto, *p* não é incluído nesta análise uma vez que quaisquer padrões frequentes envolvendo *p* já foram analisados no passo anterior. Semelhante à análise anterior, a base de dados condicional de *m* (*m's conditional pattern base*) é {(fca: 2), (fcab: 1)}. Construindo uma *FP-tree* condicionada sobre essa projeção, se obtém *m* (*m's conditional FP-tree*), um único caminho {*f:3, c:3, a:3*}. Esta *FP-tree* condicional é então analisada de forma recursiva chamando analise ({f:3, c:3, a:3} | m).

A figura 10 mostra que *analise ({f: 3, c: 3, a:3}* | *m)* envolve analisar três itens {a, c, f} de baixo para cima. O primeiro resulta no padrão frequente {am: 3}, uma base de dados condicional {(fc: 3)}, e, em seguida, uma chamada *analise ({f: 3, c: 3}* | *am)*; o segundo gera {cm: 3} como padrão frequente, e {(f: 3)} como base de dados condicional, em seguida, chama analise ({f:3} | cm); por último o terceiro gera apenas um padrão frequente {fm: 3}. Além disso a chamada de *analise ({f: 3, c: 3}* | *am)* deriva dois padrões frequentes {cam: 3} e {fam: 3}, e uma base de dados condicional {(f: 3)}, o que conduz então para uma outra chamada *analise ({f: 3* | *cam)*, que encontra o padrão mais longo {fcam: 3}. Do mesmo modo, a chamada *analise ({f:3}* | *cm)* deriva o padrão {fcm: 3}. Com isso o conjunto de padrões frequentes envolvendo *m* é definido: {(m, 3); (am: 3); (cm: 3); (fm, 3); (cam: 3); (fam: 3); (fcam: 3); (fcm: 3)}. Isto indica que de um único caminho da *FP-tree* pode-se extrair todas as combinações dos itens presentes em tal caminho. Desta forma, quando o algoritmo *FP-growth* identifica que a *FP-tree* tem somente um caminho, ele gera os conjuntos frequentes por meio da combinação dos itens presentes no caminho e inicia o processamento do próximo item da tabela de cabeçalhos (HAN et al, 2004, tradução nossa; HAN; PEI; YIN, 2000, tradução nossa).

Figura 10 – A *FP-tree* condicional para análise do *itemset m*.



Fonte: Adaptado de Han et al (2004, tradução nossa) e Han, Pei e Yin (2000, tradução nossa).

Da mesma forma, o nó *b* deriva {b, 3} e tem três caminhos: {*f:4, c:3, a:3, b:1*}; {*f:4, b:1*}; e {*c:1, b:1*}. Uma vez que a base de dados condicional de b {(fca, 1), (f, 1), (c: 1)} não gera item frequente, a análise do *itemset b* termina. Já o nó *a* gera um padrão frequente {(a, 3)} e uma base de dados condicional {(fc: 3)}, que culmina em uma *FP-tree* de caminho único. Assim, o seu conjunto de padrões frequentes pode ser gerado, com todas as combinações possíveis. Concatenando com {a, 3}, tem-se {(fa, 3), (ca: 3), (fca: 3)}. O nó c deriva {c: 4} e uma base de dados condicional {(f: 3)}, e o conjunto de padrões frequentes associado a {c: 3} é {(fc: 3)}.

Por fim, o nó *f* gera somente {f: 4}, e como nenhuma base de dados condicional pode ser gerada a execução da extração de padrões frequentes usando a *FP-tree* termina. A tabela 7 mostra resumidamente o processo de extração de padrões frequentes usando a estrutura *FP-tree*.

Tabela 7. Processo de extração de padrões frequentes usando a estrutura *FP-tree*.

Item	Base de dados condicional	*FP-tree* condicional	Padrões frequentes
p	{(fcam: 2), (cb: 1)}	{(c: 3)} | p	{(p: 3); (cp: 3)}
m	{(fca: 2), (fcab: 1)}	{(f: 3, c: 3, a: 3)} | m	{(m: 3); (am: 3); (cm: 3); (fm: 3); (cam: 3); (fam :3); (fcm: 3); (fcam: 3)}
b	{(fca: 1), (f :1), (c:1)}	Ø	{(b: 3)}
a	{(fc: 3)}	{(f: 3, c: 3)} | a	{(a: 3); (fa: 3); (ca: 3); (fca: 3)}
c	{(f: 3)}	{(f: 3)} | c	{(c: 4): (fc: 3)}
f	Ø	Ø	{(f: 4)}

Fonte: Adaptado de Han et al (2004, tradução nossa) e Han, Pei e Yin (2000, tradução nossa).

De acordo com Han et al (2004, tradução nossa) e Han, Pei e Yin (2000, tradução nossa) as propriedades e regras descritas são a base para a construção do algoritmo *FP-Growth*. A figura 11 apresenta o algoritmo *FP-Growth* responsável pela extração de padrões frequentes implícitos em uma base dados utilizando-se para isso a estrutura *FP-tree*.

Figura 11 – Algoritmo *FP-Growth*: extração de padrões frequentes usando a *FP-tree*.

```
Algoritmo: FP-Growth( Tree, α )
Entradas: FPTree Tree, padrão α
Saída: Itemsets frequentes L

1)  se ( Tree contém um único caminho P )
2)      para cada combinação β de nós em P
3)          gere padrão β U α com suporte = suporte mínimo em β
4)  senão
5)      para cada a_i da tabela header de Tree
6)          gere padrão β = a_i U α com suporte = a_i.suporte
7)          construa as bases de padrões condicionais de β
8)          construa a FP-tree condicional Tree_β
9)          se ( Tree_β ≠ Ø )
10)             FP-Growth( Tree_β, β );
```

Fonte: Han et al (2004, tradução nossa) e Han, Pei e Yin (2000, tradução nossa).

A execução desta parte ou método do algoritmo começa com uma chamada na forma *FP-Growth(T, null)*, sendo *T* a *FP-tree* que se deseja analisar com o intuito de extrair dela os padrões frequentes. O método inicia os seus

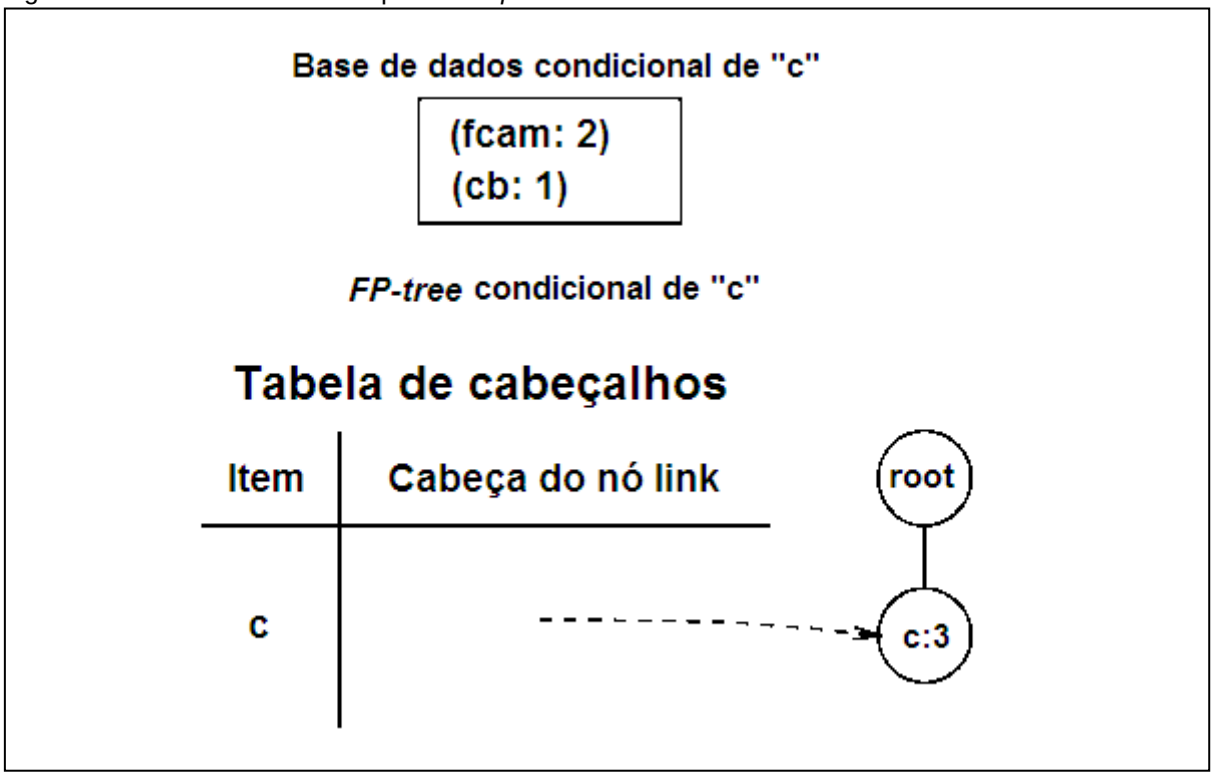
trabalhos verificando se a *FP-tree* passada como parâmetro possui um único caminho. Se a mesma possuir mais de um caminho a rotina irá fazer uma consulta na tabela de cabeçalhos da *FP-tree*. Seu objetivo é selecionar o último elemento da tabela, ou seja, aquele que possui o menor valor de suporte. Isso simplifica as operações posteriores, uma vez que o elemento não precisará ser considerado e analisado como parte do sufixo durante a análise dos outros *itemsets* presentes na tabela de cabeçalhos, pois todos os seus respectivos *itemsets* já terão sido encontrados (HAN; KAMBER, 2006).

Tomando como exemplo a *FP-tree* da figura 14 e considerando que o valor do suporte mínimo (*min_sup*) previamente informado é igual a três, pode-se exemplificar o funcionamento do método de extração de padrões frequentes.

Inicialmente, na linha 1, o algoritmo *FP-Growth* verifica se a estrutura apresenta mais de um caminho, como a condição da linha 1 é falsa, o algoritmo executa o bloco “senão” (linhas 5 a 10). O primeiro elemento da tabela de cabeçalhos a ser analisado é o *p*. Nesse caso, o padrão β será o próprio *p*, pois α foi inicializado com valor *null*. O próximo passo é construir a base de dados condicional de β, que é composta pelos caminhos existentes na *FP-tree* onde o sufixo é o próprio β. Se β = *p*, então, por meio da *FP-tree* é possível identificar os caminhos e seus respectivos valores de suporte que levam ao nó *p*, os quais são: {*f:4, c:3, a:3, m:2, p:2*} e {*c:1, b:1, p:1*}. Considerando que *p* é o sufixo desses caminhos, obtém-se a sua respectiva base condicional {(fcam: 2), (cb: 1)}.

No próximo passo, linha 8, a base de dados condicional é utilizada para construir a *FP-tree* condicional *(*$Tree_{\beta}$). Esse processo é idêntico ao de construção da estrutura *FP-tree* apresentado na seção 4.1. O primeiro caminho da base de dados condicional, {fcam: 2}, é então inserindo ordenadamente na $Tree_{\beta}$. A seguir, repete-se a inserção para o caminho {cb: 1}, esse processo resulta na *FP-tree* condicional mostrada na figura 12. Note que os nós {f, a, m} não se encontram na *FP-tree* condicional, visto que seu respectivo valor de suporte é menor que o suporte mínimo (*min_sup* < *3*).

Figura 12 – *FP-tree* condicional para o nó *p*.



Fonte: Adaptado de Han, Pei e Yin (2000, tradução nossa) e Han et al (2004, tradução nossa).

Após a construção da *FP-tree* condicional ($Tree_\beta$), o algoritmo verifica que a mesma não está vazia (linha 9), então o algoritmo aplica uma chamada recursiva na forma *FP-Growth*($Tree_\beta$, *β*) para processar a $Tree_\beta$ recém-criada para o nó *p*, representado por *β* (linha 10). Agora, a condição da linha 1 é verdadeira, pois $Tree_\beta$ apresenta um único caminho. Então, os padrões frequentes relacionados a *p* são gerados, estes consistem em todas as combinações possíveis envolvendo os nós de P acrescidos do sufixo *β*. Assim, os padrões cujo sufixo é β = *p* são encontrados. Como a $Tree_\beta$ de *p* possui apenas um único ramo, somente um padrão {cp: 3} é encontrado.

A seguir, o algoritmo inicia a análise do nó *m*, que é o próximo nó (m) presente na tabela de cabeçalhos (linha 5). O algoritmo *FP-growth* repete os passos anteriores, recursivamente, para a análise do nó *m* e depois para os demais nós presentes na tabela de cabeçalhos, até que todos tenham sido analisados.

Tabela 8. Padrões frequentes gerados.

Item	Padrões frequentes
p	{(p: 3); (cp: 3)}
m	{(m: 3); (am: 3); (cm: 3); (fm: 3); (cam: 3); (fam :3); (fcm: 3); (fcam: 3)}
b	{(b: 3)}
a	{(a: 3); (fa: 3); (ca: 3); (fca: 3)}
c	{(c: 4): (fc: 3)}
f	{(f: 4)}

Fonte: Adaptado de Han, Pei e Yin (2000, tradução nossa) e Han et al (2004, tradução nossa).

Após a análise de todos os *itemsets* presentes na tabela de cabeçalhos o algoritmo *FP-Growth* mostra os padrões frequentes gerados, juntamente com o *itemset* que os gerou. Na tabela 8 é ilustrada uma forma de se mostrar o resultado da execução do algoritmo *FP-Growth*.

5 TRABALHOS CORRELATOS

Dentre as tarefas de *data mining*, a descoberta de regras de associação tem se mostrado como um dos campos de pesquisa que desperta interesse, sendo considerada uma das mais importantes tarefas de DM (HAN; KAMBER, 2006, tradução nossa).

Na área acadêmica, as pesquisas desenvolvidas estão gerando bons resultados. Esses resultados, por sua vez, estão sendo utilizados pelas organizações em aplicações práticas. Um exemplo da utilização prática da tarefa de associação pode ser vista na área comercial em páginas de vendas de produtos na internet, que apresentam sugestões do tipo: “Quem comprou este produto que você está comprando também comprou estes outros produtos.” (MELANDA, 2004).

De uma forma geral, a tarefa de descoberta de regras de associação permite identificar o quanto a presença de um conjunto de itens nos registros de uma base de dados implica na presença de algum outro conjunto distinto de itens nos mesmos registros (MOTTA, 2010).

Com o objetivo de resolver o problema da descoberta de regras de associação diversos algoritmos foram desenvolvidos, dentre os quais se destaca o algoritmo *FP-Growth*. Ele se destaca por resolver o problema sem a necessidade de geração de um conjunto de *itemsets* candidatos, o que faz dele um algoritmo menos custoso que os algoritmos da família *Apriori*. A seguir se encontram alguns exemplos relacionados da utilização do algoritmo *FP-Growth*.

5.1 UMA IMPLEMENTAÇÃO DO ALGORITMO *FP-GROWTH*

Este artigo foi desenvolvido por Christian Borgelt e foi apresentado no *Workshop Open Source Data Mining Software* em Chicago nos Estados Unidos da América no ano de 2005.

Segundo o autor basicamente três motivos o fizeram implementar o algoritmo *FP-Growth*: o fato de o algoritmo estar sendo reconhecido como um dos algoritmos mais eficientes para resolver o problema da descoberta de regras de associação; a popularidade que o algoritmo *FP-Growth* alcançou sendo a base de diversos estudos científicos e aplicações práticas; e o desejo de compará-lo com

outros algoritmos de descoberta de regras de associação implementados pelo próprio Christian, que foram os algoritmos *Apriori*, *Eclat* e *Relim*.

Neste artigo o autor descreve a implementação do algoritmo *FP-Growth* feita na linguagem de programação C. E no final o compara com os algoritmos *Apriori*, *Eclat* e *Relim* implementados em suas versões mais otimizadas até aquele momento. Frente às comparações feitas, de acordo com o autor, o algoritmo *FP-Growth* foi claramente o mais eficiente (BORGELT, 2005, tradução nossa).

5.2 MELHORAR A EFICIÊNCIA DA *WEB USAGE MINING* USANDO OS ALGORITMOS *K-APRIORI* E *FP-GROWTH*

Este artigo foi escrito por R. Kousalya, K. Suguna, V. Saravanan e publicado no *International Journal of Scientific & Engineering Research Volume 4*, em março de 2013. Nele os autores falam da ferramenta desenvolvida para melhorar a eficiência da *Web Usage Mining*.

Web Usage Mining é a coleta e análise de informações do uso de páginas da web, objetivando encontrar padrões de navegação e do uso de seus recursos. Os dados binários são transformados em dados reais utilizando o algoritmo *Wiener*, desenvolvido pelos autores. Após, utilizam o algoritmo de clusterização *k-means,* onde os dados transformados são agrupados com base nas suas semelhanças. Em seguida, os algoritmos *K-Apriori* e *FP-Growth* são implementados e utilizados para a obtenção de padrões frequentes (KOUSALYA; SUGUNA; SARAVANAN, 2013, tradução nossa).

Segundo os autores os padrões de uso da Web descobertos são então analisados e podem ser utilizados para entender e consequentemente atender melhor as necessidades dos usuários que navegam na web. *Web Usage Mining* permite descobrir padrões de navegação, e prevê comportamentos que os usuários terão enquanto interagem com a web.

Os autores deste artigo também propõem uma melhoria no algoritmo *Apriori*, para resolver alguns de seus problemas conhecidos. Como por exemplo, os constantes acessos à base de dados e a dispendiosa geração do conjunto de *itemsets* candidatos. No trabalho os autores propõem o algoritmo *K-Apriori* que é aplicado sobre os *K* grupos de dados gerados pelo algoritmo *k-means*, com o intuito de se obter, de uma forma mais eficiente, os conjuntos de itens frequentes. Acredita-

se que uma grande base de dados dividida levará a um menor conjunto de *itemsets* candidatos e consequentemente menos acessos a base de dados, o que tende a aumentar a eficiência (KOUSALYA; SUGUNA; SARAVANAN, 2013, tradução nossa).

No final, o desempenho dos algoritmos *K-Apriori* e *FP-Growth* são comparados. O resultado foi que o algoritmo *FP-Growth* se mostrou mais eficiente do que o algoritmo *K-Apriori* (KOUSALYA; SUGUNA; SARAVANAN, 2013, tradução nossa).

5.3 DESCOBERTA DIRETA E EFICIENTE DE REGRAS DE ASSOCIAÇÃO ÓTIMAS

Esta dissertação foi apresentada ao Instituto de Ciências Matemáticas e de Computação (ICMC-USP), por Alinson Sousa de Assunção, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação e Matemática Computacional, na cidade de São Carlos, no estado de São Paulo, no ano de 2012.

Segundo o autor as medidas de interesse foram criadas para verificar a qualidade das regras de associações geradas, produzindo um filtro que elimina as regras desinteressantes. No entanto, elas não acarretam na diminuição do tempo de execução do algoritmo minerador, pois tais cálculos são aplicados nas regras geradas após as mesmas terem sido encontradas. Para vencer essa dificuldade, foram desenvolvidas técnicas que exploram diretamente regras de associação ótimas. Ainda de acordo com o autor sua dissertação segue a mesma direção apresentada e visa uma melhor abordagem deste tema para que se possam descobrir regras de associação ótimas.

As abordagens anteriores apresentam um entrave devido à utilização do algoritmo *Apriori*. Tal entrave ocorre devido ao *Apriori* realizar uma busca em largura sobre os conjuntos de dados. As técnicas que realizam busca em profundidade sobre o espaço a ser analisado mostram-se mais promissoras. Em virtude dessa característica foi adotado o algoritmo *FP-Growth*, pois ele realiza uma busca em profundidade sobre os conjuntos de itens a serem explorados (ASSUNÇÃO, 2012).

De acordo com o autor os resultados mostraram que a utilização da exclusiva metodologia adotada pelo algoritmo *FP-Growth* foi responsável pela melhor eficiência da implementação desenvolvida na grande maioria dos

experimentos realizados. Isso justificou sua utilização em detrimento a adoção da metodologia empregada pelo algoritmo *Apriori* (ASSUNÇÃO, 2012).

5.4 SAMIRA – UMA PROPOSTA DE SISTEMA DE APOIO À MINERAÇÃO DE REGRAS DE ASSOCIAÇÃO

Esta dissertação foi desenvolvida por Sandro Luz da Paixão Xavier e apresentada ao Instituto Federal de Educação, Ciência e Tecnologia do Ceará, como parte dos requisitos para a obtenção do título de Mestre em Computação Aplicada, na cidade de Fortaleza, no estado do Ceará, no ano de 2010.

Neste trabalho o autor desenvolveu um sistema de apoio à decisão chamado **S**istema de **A**poio à **Mi**neração de **R**egras de **A**ssociação (SAMiRA), que engloba todas as etapas do processo de *data mining*. Para validar o sistema desenvolvido, o mesmo foi aplicado em estudos de casos com bases de dados de duas empresas, onde foi possível fazer uma análise das regras geradas para as duas empresas e realizar uma comparação dos algoritmos *Apriori* e *FP-Growth* implementados (XAVIER, 2010).

Como resultados verificou-se que o algoritmo *Apriori* encontrou uma quantidade um pouco maior de regras de associação em alguns casos com suporte baixo, enquanto que o algoritmo *FP-Growth* apresentou um tempo de execução expressivamente menor (XAVIER, 2010).

6 O ALGORITMO *FP-GROWTH* NA TAREFA DE ASSOCIAÇÃO DA *SHELL ORION DATA MINING ENGINE*

A *Shell Orion Data Mining Engine,* um software livre que implementa o processo da descoberta de conhecimento em bases de dados, é um projeto de pesquisa vinculado ao Grupo de Pesquisa em Inteligência Computacional Aplicada, do Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense (UNESC). Suas funcionalidades são ampliadas por meio da inserção de algoritmos e métodos de DM, e melhoradas por meio de estudos de casos, desenvolvidos por alunos da universidade em seus respectivos Trabalhos de Conclusão de Curso.

Seguindo esse contexto, essa pesquisa se propõe a aumentar as funcionalidades presentes na *Shell Orion*, por meio da implementação do algoritmo *FP-Growth* para a tarefa de associação.

Na análise do desempenho e da qualidade dos algoritmos implementados na *Shell Orion*, são utilizadas bases de dados, sendo que nessa pesquisa optou-se pela utilização de duas bases de dados uma na área da saúde e outra na área de ciências políticas.

6.1 BASE DE DADOS

Na avaliação da implementação do algoritmo *FP-Growth*, utilizou-se duas bases de dados. Ambas foram selecionadas no *UCI Machine Learning Repository*[7].

A base de dados da área de ciências políticas disponibiliza informações obtidas a partir de votações dos congressistas nos Estados Unidos acerca de 16 questões importantes para o país.

Esta base de dados apresenta 435 transações e 17 atributos, dos quais 16 atributos representam projetos que foram votados pelos congressistas e um atributo classe identifica se o político é Republicano ou Democrata. Coletou-se a mesma em 1984 a partir de uma seção de votação de congressistas norte americanos. A tabela 9 apresenta as características e as considerações técnicas de cada atributo, tais informações foram extraídas do UCI *Machine Learning Repository*.

[7] *UCI Machine Learning Repository* é um repositório de bases de dados mantido pelo Departamento de Informação e Ciências da Computação da Universidade da Califórnia, Irvine, disponível gratuitamente no seguinte endereço eletrônico: http://archive.ics.uci.edu/ml/.

Tabela 9. Base de dados: *Congressional Voting Records Data Set.*

Atributo	Descrição	Valor
Class	Identifica o partido político do congressista	Nominal: Democrata ou Republicano
handicapped-infants	Projeto: crianças portadoras de deficiência	Nominal: Sim ou Não
water-project-cost-sharing	Projeto hídrico de partilha de custos	Nominal: Sim ou Não
adoption-of-the-budget-resolution	Projeto: aprovação da resolução do orçamento	Nominal: Sim ou Não
physician-fee-freeze	Projeto: médico taxa de congelamento	Nominal: Sim ou Não
el-salvador-aid	Projeto: ajuda a El Salvador	Nominal: Sim ou Não
religious-groups-in-schools	Projeto: grupos religiosos nas escolas	Nominal: Sim ou Não
anti-satellite-test-ban	Projeto: proibição antissatélite de teste	Nominal: Sim ou Não
aid-to-nicaraguan-contras	Projeto: contra ajudar a Nicarágua	Nominal: Sim ou Não
mx-missile	Projeto: míssil MX	Nominal: Sim ou Não
immigration	Projeto: imigração	Nominal: Sim ou Não
synfuels-corporation-cutback	Projeto: CUTBACK corporação de combustíveis sintéticos	Nominal: Sim ou Não
education-spending	Projeto: gastos com educação	Nominal: Sim ou Não
superfund-right-to-sue	Projeto: Fundo que possa ser acionado	Nominal: Sim ou Não
crime	Projeto: crime	Nominal: Sim ou Não
duty-free-exports	Projeto: exportações com isenção de impostos	Nominal: Sim ou Não
export-administration-act-south-africa	Projeto: exportação da administração que atua na África do Sul	Nominal: Sim ou Não

Fonte: Adaptado de *UCI Machine Learning Repository* (2013).

A outra base de dados utilizada na presente pesquisa acadêmica contém dados clínicos de pacientes com diagnóstico para hepatite. Esta base de dados possui 155 transações distribuídas em duas classes, a dos pacientes que morreram composta por 32 registros e a outra com os que viveram com 123 casos e outra que identifica o sexo do paciente, 12 atributos são binários, enquanto 6 atributos apresentam valores contínuos, totalizando 20 registros. A tabela 10 apresenta as características e as considerações técnicas de 14 atributos, tais informações foram extraídas do UCI *Machine Learning Repository*.

Tabela 10. Base de dados: *Congressional Voting Records Data Set.*

Atributo	Descrição	Valor
Class	Identifica se o paciente viveu ou morreu	Nominal: Morto ou Vivo
SEX	Sexo do paciente	Nominal: Masculino ou Feminio
STEROID	Identifica se o paciente possui a presença de esteroides.	Nominal: Sim ou Não
ANTIVIRALS	Identifica se o paciente possui a presença de antivirais	Nominal: Sim ou Não
FATIGUE	Identifica se o paciente possui o sintoma de sentir fatiga	Nominal: Sim ou Não
MALAISE	Identifica se o paciente possui o sintoma de sentir mal-estar	Nominal: Sim ou Não
ANOREXIA	Identifica se o paciente possui o distúrbio alimentar da anorexia	Nominal: Sim ou Não
LIVER_BIG	Identifica se o paciente possui o fígado grande	Nominal: Sim ou Não
LIVER FIRM	Identifica se o paciente possui o fígado forte	Nominal: Sim ou Não
SPLEEN PALPABLE	Identifica se o paciente possui o baço palpável	Nominal: Sim ou Não
SPIDERS	Identifica se o paciente apresenta o sintoma de aranha vascular	Nominal: Sim ou Não
ASCITES	Identifica se o paciente apresenta o sintoma de ascite	Nominal: Sim ou Não
VARICES	Identifica se o paciente apresenta a presença de varizes	Nominal: Sim ou Não
HISTOLOGY	Identifica se o paciente teve tecidos estudados	Nominal: Sim ou Não

Fonte: Adaptado de *UCI Machine Learning Repository* (2013).

Os atributos que possuem valores contínuos (*AGE; BILIRUBIN; ALK PHOSPHATE; SGOT; ALBUMIN; PROTIME*) não foram considerados nesta etapa devido os mesmos não aparecem nas regras de associação uma vez que possuem índice de suporte baixo, interferindo apenas no custo computacional.

6.2 METODOLOGIA

Para a realização deste trabalho acadêmico foram empregadas as seguintes etapas metodológicas: levantamento bibliográfico; modelagem por meio do padrão UML; demonstração matemática do algoritmo *FP-Growth*; implementação e realização de testes; validação dos resultados obtidos; análise de desempenho; e análise estatística dos tempos de processamento da *Shell Orion* em comparação com a ferramenta Weka.

Durante a etapa do levantamento bibliográfico o foco foi a escrita da fundamentação teórica tendo-se entendido os temas envolvidos na pesquisa, tais

como o processo de descoberta de conhecimento em bases de dados, *data mining*, a tarefa de associação, o algoritmo *FP-Growth* e as medidas de qualidade para algoritmos de associação.

6.2.1 Modelagem UML do Módulo do Algoritmo *FP-GROWTH*

A implementação teve início pela modelagem dos processos executados pelo algoritmo *FP-Growth* por meio do padrão *Unified Modeling Language*[8] (UML). A modelagem UML tem por objetivo auxiliar os desenvolvedores de software a obterem um melhor entendimento das características do algoritmo e dos processos necessários para sua execução.

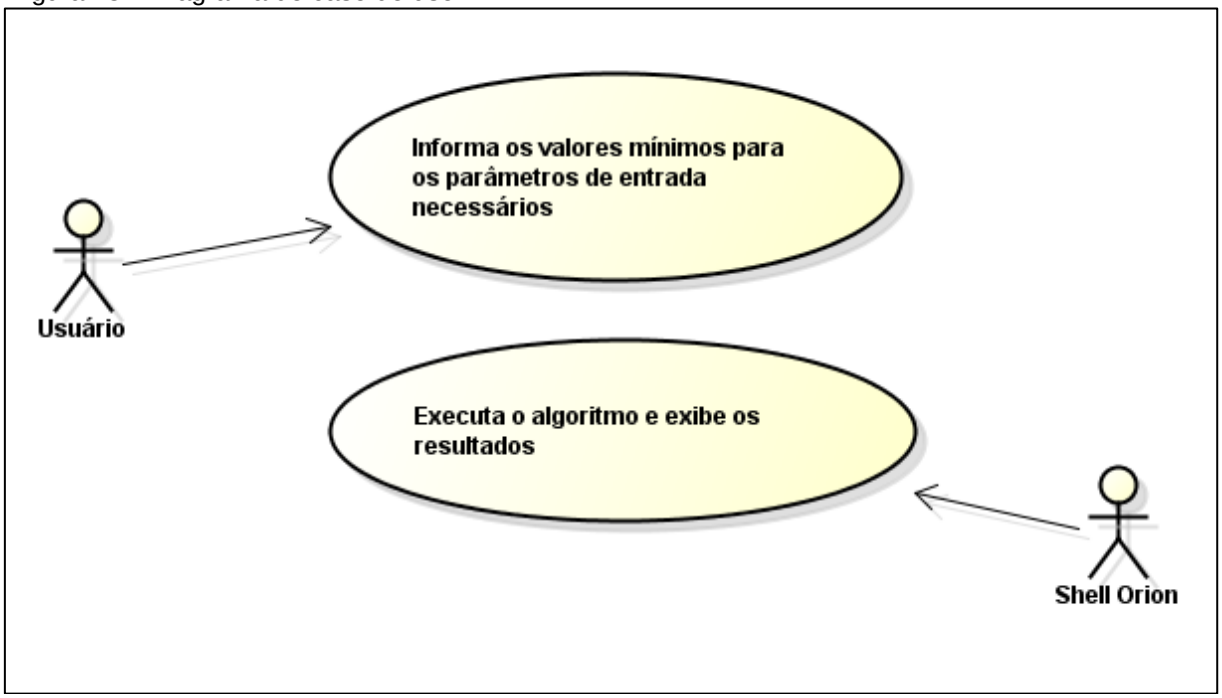
Foram desenvolvidos, com o auxilio da ferramenta Astah Community[9], os diagramas de caso de uso, de atividades e de sequências.

O diagrama de casos de uso é considerado o mais geral e informal da UML uma vez que permite que se tenha uma ideia de como o software irá se comportar. É utilizado na análise e levantamento dos requisitos, onde são levantadas as necessidades do(s) usuário(s) (GUEDES, 2008). Conforme mostra a figura 13, o diagrama de caso de uso destaca duas etapas principais:

a) **informar os parâmetros de entrada do algoritmo:** o usuário deve informar os valores mínimos para os parâmetros necessários para a execução do algoritmo. São informados os percentuais de suporte e confiança, bem como os valores para o *lift*, o *leverage* e a convicção. O usuário também deve informar se o algoritmo irá analisar os dados no formato *basket*;
b) **execução do algoritmo:** a *Shell Orion* recebe os parâmetros de entrada informados e selecionados pelo usuário e executa o algoritmo *FP-Growth* retornando os resultados obtidos.

[8] *Unified Modeling Language* ou Linguagem de Molagem Unificada é utilizada para modelar, especificar, visualizar, documentar e construir sistemas computacionais por meio do paradigma de Orientação a Objetos (GUEDES, 2008).
[9] Disponível para *download* gratuitamente em (http://astah.net/download).

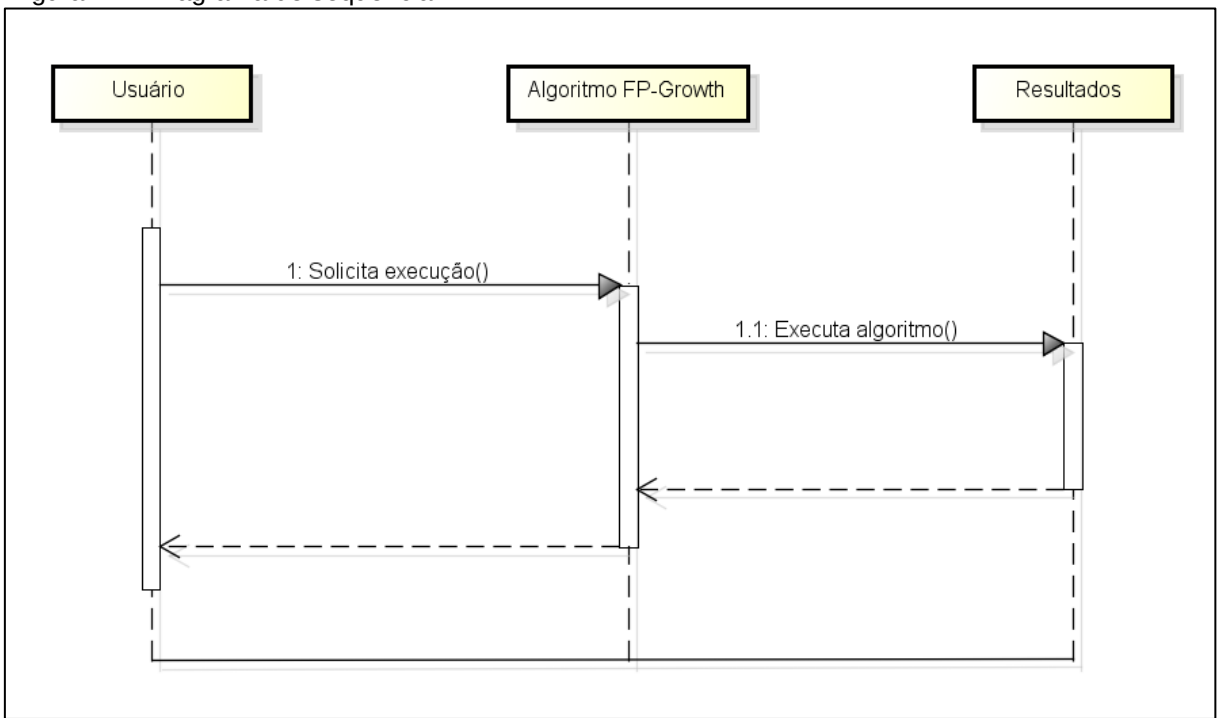
Figura 13 – Diagrama de caso de uso.



Fonte: Do Autor.

O diagrama de sequência tem por objetivo ilustrar a ordem em que as mensagens são trocadas entre os objetos envolvidos em um determinado processo. De uma forma geral, se baseia no diagrama de casos de uso para identificar os eventos geradores do processo modelado, bem como o processo deve ser executado e concluído (GUEDES, 2008).

Figura 14 – Diagrama de sequência.

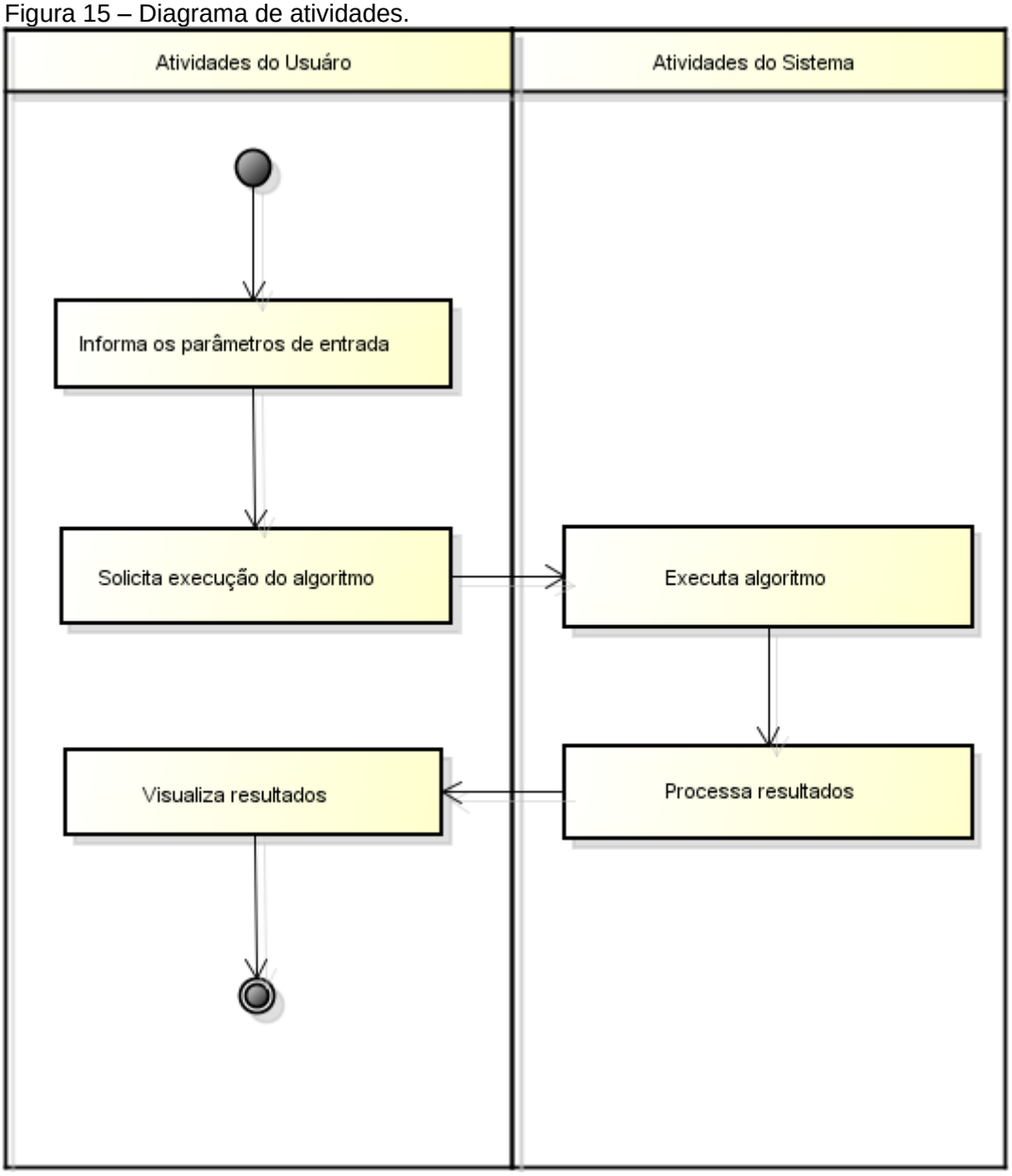


Fonte: Do Autor.

Por meio do diagrama de sequência (Figura 14) pode-se observar a interação do usuário com o algoritmo *FP-Growth*. O usuário informa os parâmetros de entrada na tela inicial e solicita a execução do algoritmo, o algoritmo *FP-Growth* será executado e os resultados são exibidos ao usuário solicitante da operação.

Já o diagrama de atividades concentra-se em representar o fluxo de controle de um processo, descrevendo os passos a serem executados para a sua conclusão (GUEDES, 2008).

Figura 15 – Diagrama de atividades.



Fonte: Do Autor.

É possível verificar, por meio da figura 15, que em uma única rotina está presente todo o fluxo de atividades necessárias para a execução do algoritmo *FP-Growth*.

As atividades estão divididas entre as realizadas pelo sistema (*Shell Orion*) e as efetuadas pelo usuário. O usuário é responsável por selecionar os parâmetros de entrada e executar o algoritmo na *Shell Orion*, a ferramenta executa o algoritmo, processa os resultados e mostra para o usuário.

A modelagem por meio da UML permitiu a compreensão do sistema a ser desenvolvido, bem como na documentação e visualização do funcionamento do algoritmo *FP-Growth* na *Shell Orion Data Mining Engine*.

6.2.2 Demonstração Matemática do Algoritmo *FP-GROWTH*

Esta etapa do trabalho tem por objetivo demonstrar o funcionamento do algoritmo *FP-Growth*, por meio da modelagem matemática, permitindo uma compreensão mais detalhada do seu funcionamento.

Os conceitos, formalismos e técnicas do algoritmo *FP-Growth* foram embasados no artigo escrito por Jiawei Han, Jian Pei e Yiwen Yin, publicado na *Conference on Management of Data* (SIGMOD'00), em Dallas no estado do Texas nos Estados Unidos da América no ano de 2000, intitulado *Mining Frequent Patterns without Candidate Generation*.

Os valores informados como parâmetros de entrada para o algoritmo são fundamentais, uma vez que a saída resultante é afetada em função desses parâmetros.

Na demonstração do funcionamento do algoritmo foram definidos os seguintes parâmetros de entrada:

a) **suporte:** é a métrica utilizada pelo algoritmo para encontrar todos os *N itemsets* frequentes. O suporte de uma regra de associação X, $A \Rightarrow B$, é a porcentagem das transações que contêm $A \cup B$ em relação ao número total de transações analisadas. Foi definido o valor de 20% com a finalidade de demonstrar detalhadamente o funcionamento do algoritmo;

b) **confiança:** trata-se de outra métrica utilizada pelo algoritmo *FP-Growth,* no entanto, esta é utilizada após o algoritmo ter descoberto todos os *N itemsets* frequentes. O motivo de não se calcular a confiança juntamente com o suporte foi detalhado na seção 3.2. A confiança calcula a força da regra. Assim, sendo C a confiança de uma regra de associação, $A \Rightarrow B$, C é na verdade a porcentagem das transações que contêm $A \cup B$ em relação a todas as transações que contêm A. Portanto, C é calculado pela probabilidade condicional $P(B|A)$, ou seja, a probabilidade de ocorrência de B, quando A ocorre. Definiu-se o valor de 60% para simplificar a demonstração do funcionamento do algoritmo.

Nesta demonstração matemática foi utilizada uma base de dados contendo 10 transações, onde a quantidade de itens que pode constar em um TID

não é limitada ao número de atributos da relação, conforme mostrado na tabela 11.

Tabela 11. Base de dados utilizada para a modelagem do algoritmo

TID	Itens
1	N, J, M
2	N, F, G
3	J, G, F, H
4	J, F, H
5	J, N, G
6	J, N, G, F
7	J, P
8	J, N, G
9	J, N, F
10	N, G, H, O

Fonte: Do Autor.

Definida a base de dados que será analisada, o primeiro passo consiste em criar a estrutura *FP-tree* que irá armazenar os 1 *itemsets* frequentes. Para isso é necessário ler a base de dados duas vezes para que a *FP-tree* seja corretamente criada. Durante a primeira leitura é identificado o conjunto de *itemsets* frequentes F, de tamanho um, e seus respectivos suportes. A tabela 12 mostra a primeira leitura da base de dados.

Tabela 12. Primeira leitura da base de dados utilizada para a modelagem do algoritmo

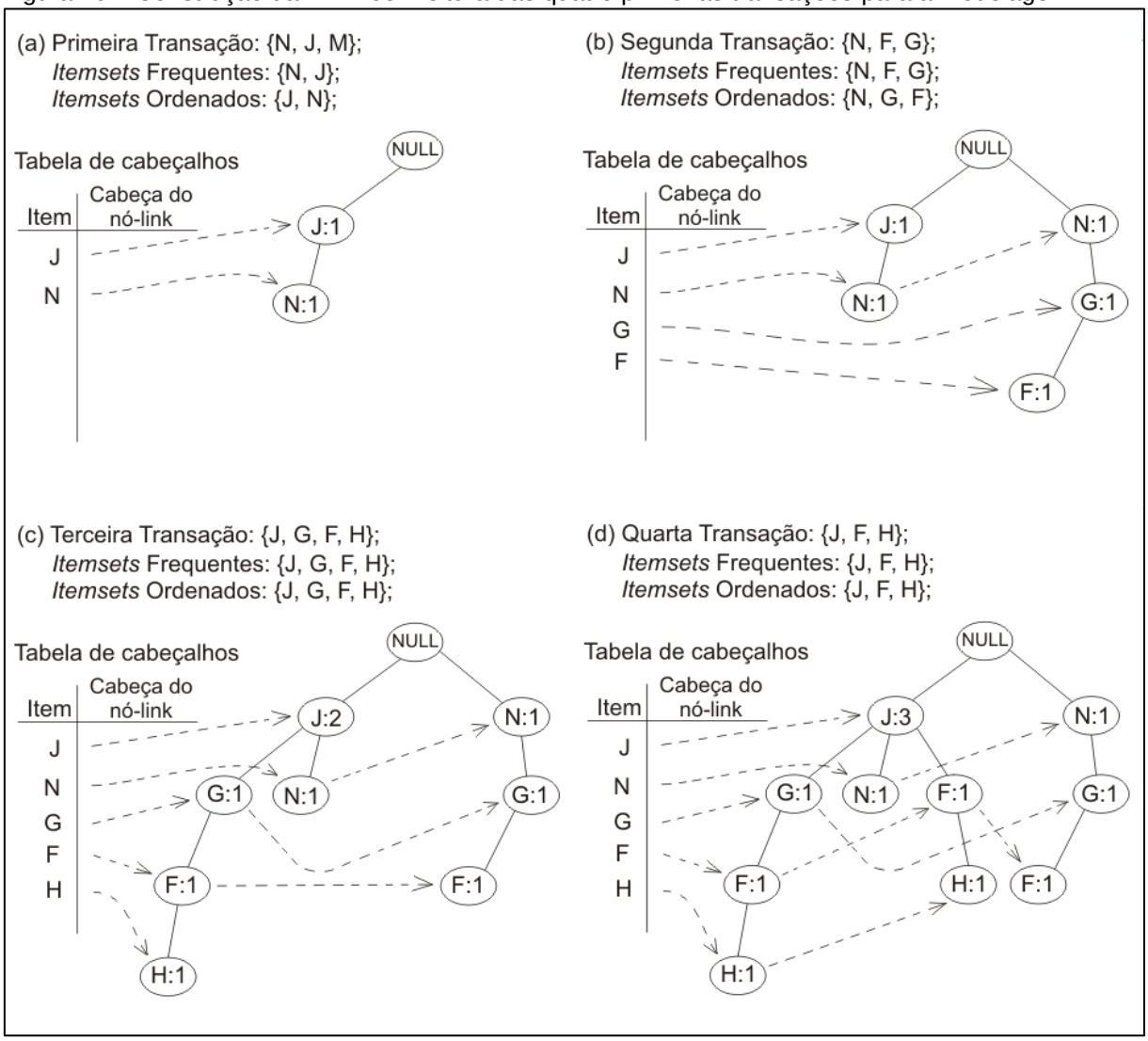
Transações	Itens	1 *Itemsets*
1	N, J, M	N:1, J:1, M:1
2	N, F, G	N:2, J:1, M:1, F:1, G:1
3	J, G, F, H	N:2, J:2, M:1, F:2, G:2, H:1
4	J, F, H	N:2, J:3, M:1, F:3, G:2, H:2
5	J, N, G	N:3, J:4, M:1, F:3, G:3, H:2
6	J, N, G, F	N:4, J:5, M:1, F:4, G:4, H:2
7	J, P	N:4, J:6, M:1, F:4, G:4, H:2, P:1
8	J, N, G	N:5, J:7, M:1, F:4, G:5, H:2, P:1
9	J, N, F	N:6, J:8, M:1, F:5, G:5, H:2, P:1
10	N, G, H, O	N:7, J:8, M:1, F:5, G:6, H:3, P:1, O:1

Fonte: Do Autor.

Logo após a primeira leitura da base de dados os 1 *itemsets* infrequentes, que possuem valor de suporte menor que 20%, {*M:1, P:1, O:1*} são removidos do conjunto F, o mesmo é ordenado em ordem decrescente conforme o valor do suporte, o conjunto F ordenado é armazenado em L, que irá possuir a lista de itens frequentes {J:8, N:7, G:6, F:5, H:3}. Quando o *itemset* possui o mesmo valor de suporte o algoritmo ordena em ordem alfabética.

O algoritmo *FP-Growth* cria então a árvore *FP-tree* e rotula o nó raiz como *NULL*. Começa a segunda leitura da base de dados onde cada transação é lida novamente, os *itemsets* infrequentes de tal transação serão desconsiderados e os frequentes serão armazenados em uma lista. A lista, por sua vez, é ordenada de acordo com o valor de suporte do *itemset*. Em seguida ela será passada como parâmetro para o método insere_tree(lista_ordenada, “NULL”). A figura 16 ilustra o processo de inserção das quatro primeiras transações da base de dados na estrutura *FP-tree*.

Figura 16 – Construção da *FP-Tree*: Leitura das quatro primeiras transações para a modelagem



Fonte: Do Autor.

A figura 16(a) mostra a leitura da primeira transação. Verifique que o item {M} é descartado, já os itens ordenados {J, N} são inseridos na árvore. Para inseri-los o algoritmo *FP-Growth* chama o método insere_tree({J, N}, “NULL”). O método

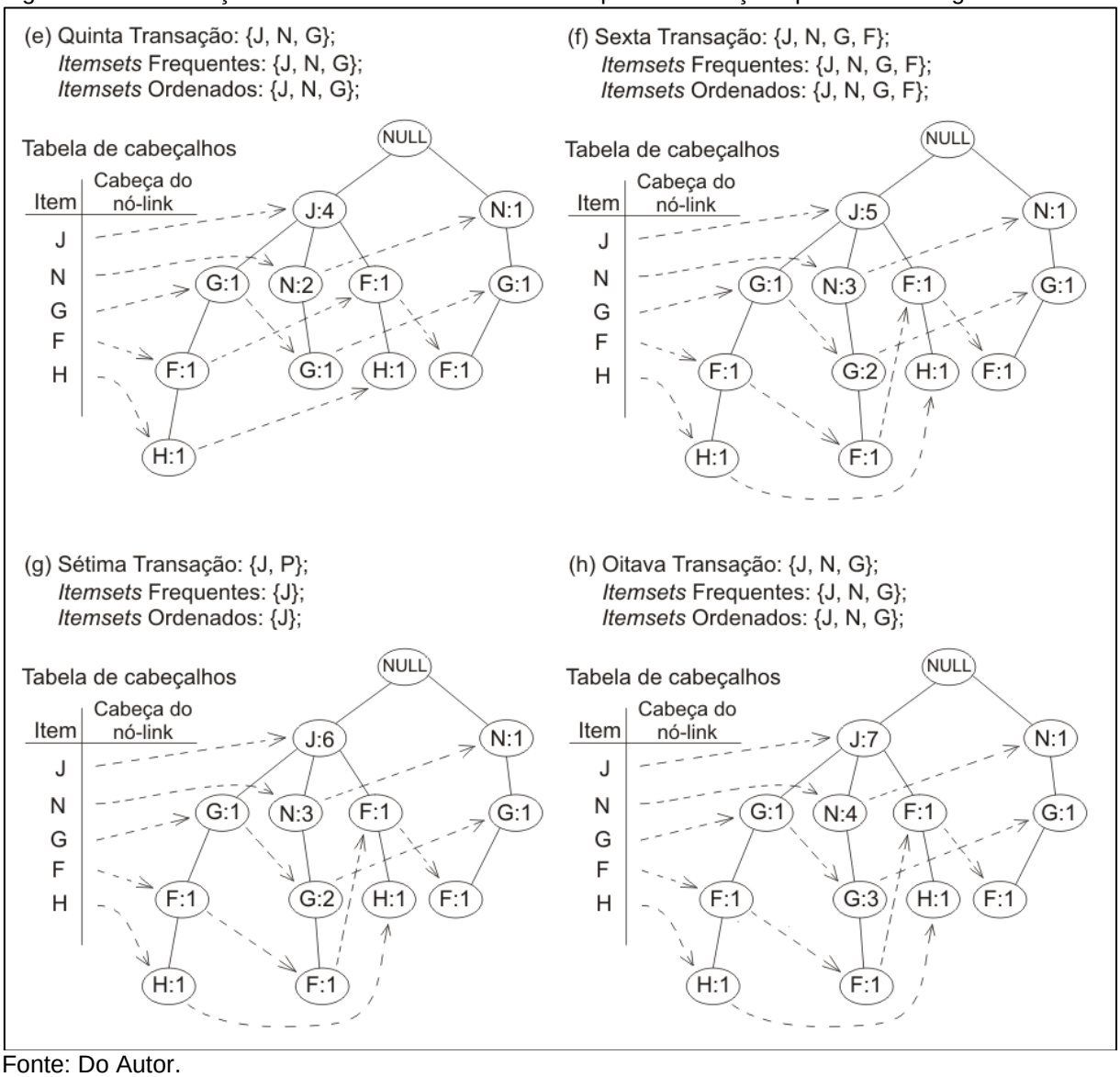
passa a inserir o primeiro elemento da lista que no caso é o item {J}, faz isso verificando se o nó pai (NULL) tem algum filho com o mesmo nome item igual a {J}. Como neste caso não possui o método cria um novo nó (J:1) e liga-o como um filho do nó raiz (NULL), a tabela de cabeçalhos é atualizada e o item {J} é excluído da lista. O método verifica a lista, como ela não está vazia o método faz uma chamada recursiva, insere_tree({N}, “J”), agora o método irá inserir o item {N} na *FP-tree* uma vez que ele é o primeiro elemento da lista. Como o nó pai (J:1) não possui nenhum nó filho com o mesmo nome item igual a {N}, um novo nó (N:1) é criado e vinculado como filho do nó (J:1), a tabela de cabeçalhos é atualizada e o item {N} é excluído da lista. Agora a lista está vazia, finalizando-se a recursão e inserindo-se, a primeira transação na *FP-tree*.

O algoritmo passa a ler a segunda transação {N, F, G}, figura 16(b). Todos os itens satisfazem o suporte mínimo de 20 %, então nenhum é descartado. Os itens ordenados {N, G, F} são passados por parâmetro para o método insere_tree({N, G, F}, “NULL”). Note que a inserção dos nós na árvore irá sempre partir do nó raiz (NULL) para baixo, ou seja, no formato *top-down*. O primeiro elemento a ser inserido na *FP-tree* é o item {N} uma vez que possui o maior valor de suporte. O método verifica se o nó pai (NULL) tem algum filho com o mesmo nome item igual a {N}, como não há, o método cria um novo nó (N:1) e liga-o como um filho do nó raiz (NULL). A tabela de cabeçalhos é atualizada e o item {N} é excluído da lista, o método verifica a lista, como ela não está vazia faz uma chamada recursiva, insere_tree({G, F}, “N”), agora o método irá inserir o item {G} na *FP-tree* uma vez que ele é o primeiro elemento da lista. Como o nó pai (N:1) não possui nenhum nó filho com o mesmo nome item igual a {G}, um novo nó (G:1) é criado e vinculado como filho do nó (N:1). A tabela de cabeçalhos é atualizada e o item {G} é excluído da lista. Verifica-se que a lista não está vazia, por isso uma nova chamada recursiva é realizada: insere_tree({F}, “G”). Desta vez o item a ser inserido na estrutura é o {F}. Como o nó (G:1) não possui nenhum nó filho com o mesmo nome item igual a {F} um novo nó (F:1) é criado e ligado como filho do nó (G:1)), a tabela de cabeçalhos é atualizada e o item {F} é excluído da lista. A recursão termina, pois a lista está vazia e a segunda transação também foi inserida na árvore *FP-tree*.

Agora é a vez da terceira transação {J, G, F, H}, figura 16(c). Nenhum item é descartado. Os mesmos são passados por parâmetro para o método insere_tree({J, G, F, H}, “NULL”). O primeiro elemento a ser analisado é o item {J}. O

método verifica se o nó pai (NULL) tem algum filho com o mesmo nome item igual a {J}, como há, o método incrementa o nó (J:1) para (J:2), a tabela de cabeçalhos é atualizada e o item {J} é excluído da lista, o método verifica se a lista está vazia, como a lista não está vazia o método faz uma chamada recursiva, insere_tree({G, F, H }, “J”), agora o método irá inserir o item {G} na *FP-tree* uma vez que ele é o primeiro elemento da lista. Como o nó pai (J:2) não possui nenhum nó filho com o mesmo nome item igual a {G}, um novo nó (G:1) é criado e vinculado como filho do nó (J:2), a tabela de cabeçalhos é atualizada e o item {G} é excluído da lista. Verifica-se que a lista não está vazia, por isso uma nova chamada recursiva é realizada: insere_tree({F, H}, “G”). O método segue o mesmo padrão para inserir os itens {F, H} e as demais transações da base de dados. A figura 17 mostra a leitura de outras quatro transações da base de dados.

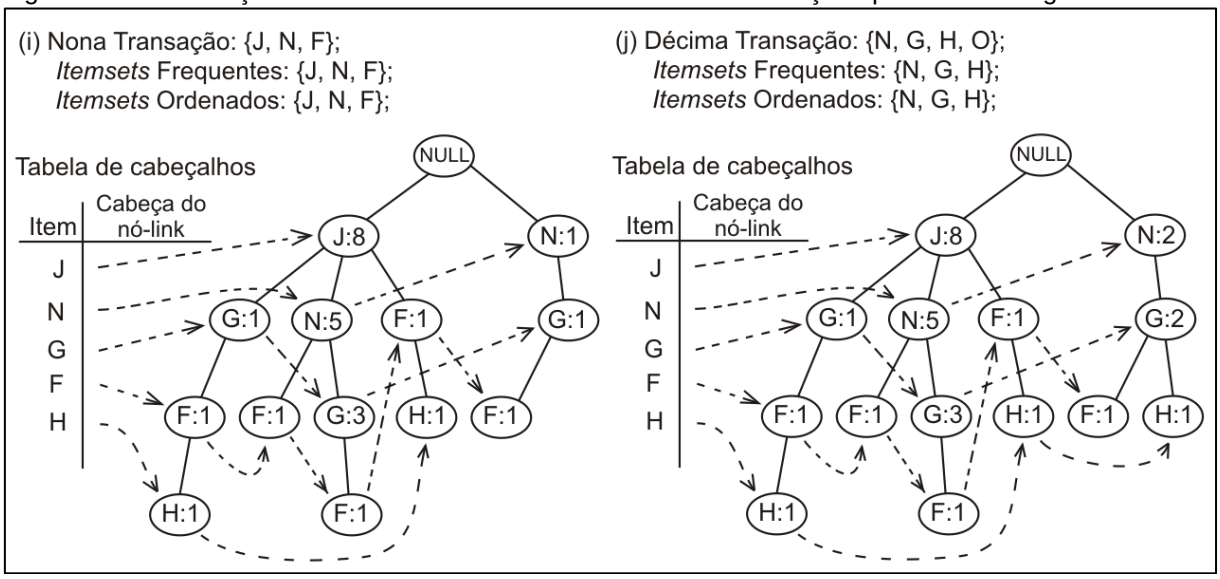
Figura 17 – Construção da *FP-Tree*: Leitura de outras quatro transações para a modelagem



Fonte: Do Autor.

Na construção da estrutura *FP-tree*, o algoritmo *FP-Growth* cria uma tendência. Isso ocorre devido à forma como as informações, que estão contidas nas transações que estão sendo lidas, são organizadas. Essa tendência, padrão frequente de crescimento, dá nome ao algoritmo; contribui para um melhor aproveitamento dos ramos inseridos na estrutura e a torna compacta; também evita a dispendiosa etapa da geração e teste de candidatos, pois não necessita que um *itemset* descoberto seja armazenado para consultas em iterações posteriores, não tendo-se a ideia de conhecimentos prévios. A figura 18 mostra a leitura das duas últimas transações da base de dados utilizada para a demonstração da modelagem matemática.

Figura 18 – Construção da *FP-Tree*: Leitura das duas últimas transações para a modelagem



Fonte: Do Autor.

A figura 18(j) mostra a estrutura *FP-tree* concluída com todas as transações da base de dados inseridas, podendo-se verificar o quanto a estrutura de dados é compacta. Por exemplo, a base de dados mostrada na tabela 9 possui 32 itens enquanto que a árvore *FP-tree*, figura 18(j), possui apenas 14 itens, isso representa uma compactação de 56,25%. A tendência é que a taxa de compactação aumente à medida que cresce o número de transações presentes na base de dados.

Ao término da segunda leitura da base de dados todas as transações estarão processadas e inseridas na estrutura *FP-tree* que estará corretamente criada. Ela irá representar a base de dados original e será passada como parâmetro para outro método do algoritmo *FP-Growth* responsável pela descoberta dos *N itemsets* frequentes.

A execução do método que analisa a estrutura *FP-tree* começa com uma chamada na forma analisa_tree(T, “NULL”), sendo T a *FP-tree* que se deseja analisar. Inicialmente o método verifica se a *FP-tree*, figura 18(j), possui um único caminho. Como a estrutura não possui um único caminho o método irá criar um laço de repetição na tabela de cabeçalhos, percorrendo-a de forma crescente, conforme o valor do suporte todos os itens presentes na mesma. Nesse caso, o primeiro item analisado da tabela de cabeçalhos é o {H}. O método concatena o item {H} com o segundo parâmetro passado na chamada. Como foi passado “NULL” o primeiro *itemset* frequente a ser identificado é o {(H:3)}. O próximo passo é construir a base de dados condicional do item {H}, que será composta pelos caminhos existentes na

FP-tree onde o sufixo é o próprio {H}. Por meio da tabela de cabeçalhos é possível identificar os caminhos que levam ao nó {H}, os quais são: {J:8, G:1, F:1}; {J:8, F:1} e {N:2, G:2}. Os contadores de suporte de cada caminho são atualizados para o mesmo valor do item {H}, obtém-se assim sua respectiva base condicional {(JGF: 1), (JF: 1), (NG: 1)}. A base condicional possui três transações e será utilizada para construir a *FP-tree* condicional do item {H}. A tabela 13 mostra a primeira leitura da base condicional do item {H}.

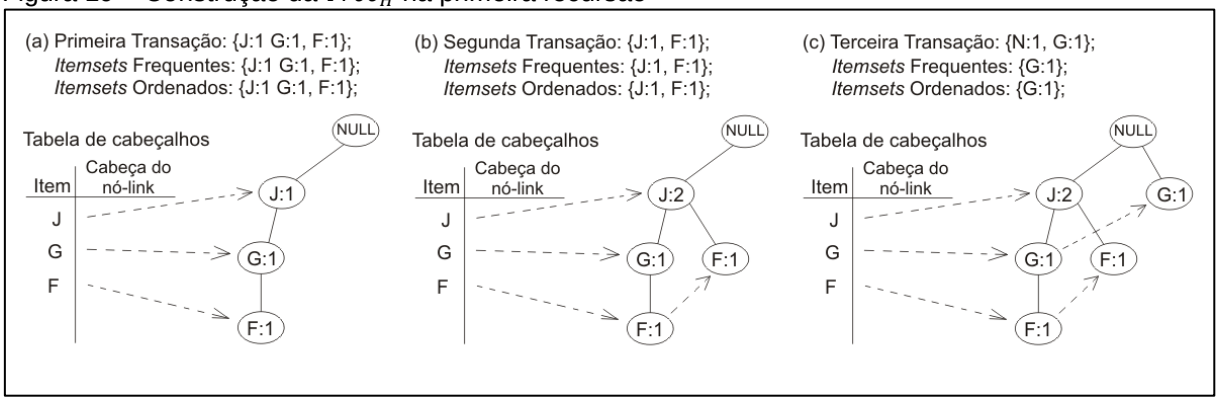
Tabela 13. Primeira leitura da base condicional do item {H}

Transações	Itens	1 *Itemsets*
1	(JGF: 1)	J:1, G:1, F:1
2	(JF: 1)	J:2, G:1, F:2
3	(NG: 1)	J:2, G:2, F:2, N:1

Fonte: Do Autor.

Dando continuidade o método começa a segunda leitura, onde o primeiro caminho da base de dados condicional, {JGF: 1}, é então lido e inserindo ordenadamente na $Tree_H$. A seguir, repete-se o processo para as transações {JF: 1} e {NG: 1}, esse processo resulta na *FP-tree* condicional mostrada na figura 19. Note que o nó {N} não se encontra na *FP-tree* condicional, visto que seu respectivo valor de suporte é menor que o suporte mínimo definido (*min_sup < 20%*).

Figura 19 – Construção da $Tree_H$ na primeira recursão



Fonte: Do Autor.

Verifica-se que esse processo é igual ao que cria a estrutura *FP-tree*, exceto pelo fato que aqui o *itemset* não é necessariamente incrementado em 1 ou inserido na árvore e atribuído o valor 1. Nesse caso o valor que é incrementado ou atribuído é o do *itemset* sufixo que neste caso é o {H}.

Criada a $Tree_H$ condicional, figura 19(c), o método a verifica para ver se ela está vazia. Nesse caso a estrutura condicional para o sufixo { H} não é vazia, então o método faz uma chamada recursiva analisa_tree($Tree_H$, “H”). O método verifica se a $Tree_H$, passada como parâmetro, possui um único caminho. Como a estrutura não possui um único caminho o método irá criar um laço de repetição na tabela de cabeçalhos da $Tree_H$. Nesse caso, o primeiro item da tabela de cabeçalhos que será analisado é o item {F}. O método concatena o item {F} com o segundo parâmetro passado na chamada, que no caso é o item “H”, assim o segundo *itemset* frequente é identificado: {(F,H: 2)}. O passo seguinte é construir a base de dados condicional do item {F}, usando-se a tabela de cabeçalhos que permite identificar os caminhos que levam ao nó {F}, os quais são: {J:2, G:1}; e {J:2}. Os contadores de suporte de cada caminho são atualizados para o mesmo valor do item {F}, obtém-se assim sua respectiva base condicional {(JG: 1), (J: 1)}. A base condicional possui duas transações, e será utilizada para construir a *FP-tree* condicional do item {F}. A tabela 14 mostra a primeira leitura da base condicional do item {F}.

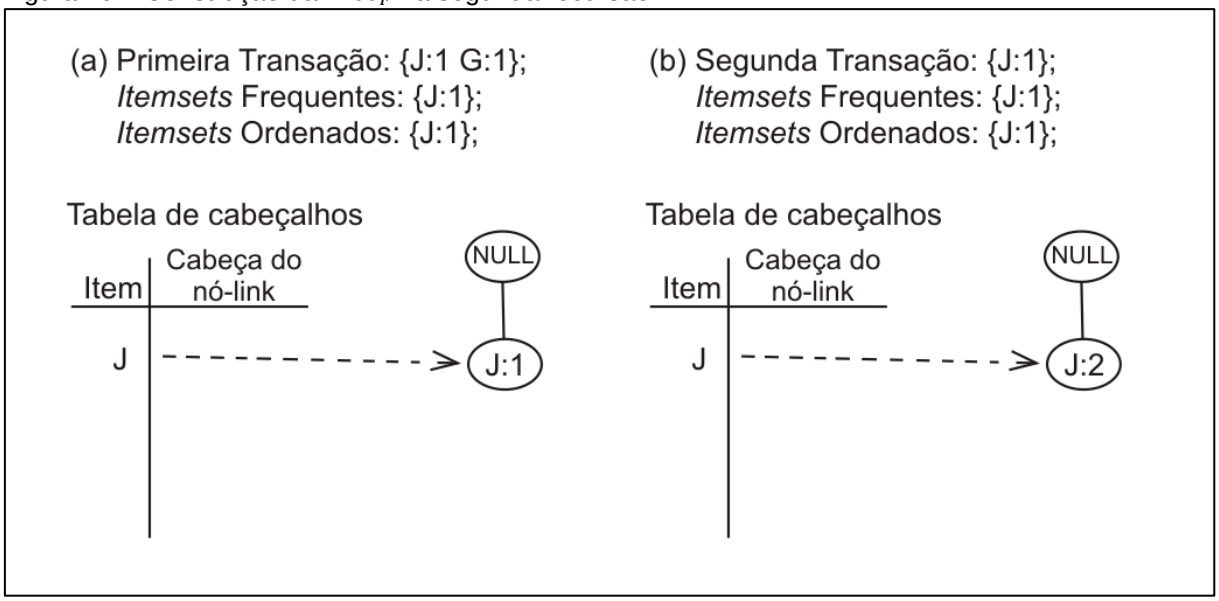
Tabela 14. Primeira leitura da base condicional do item {F}

Transações	Itens	1 *Itemsets*
1	(JG: 1)	J:1, G:1
2	(J: 1)	J:2, G:1

Fonte: Do Autor.

Começa a segunda leitura. O primeiro caminho da base de dados condicional, {JG: 1}, é então lido e somente o *itemset* {J:1} será inserindo na $Tree_F$. Pois o *itemset* {G:1} apresentou suporte menor que o suporte mínimo (*min_sup* < *20%*) no final da primeira leitura. Repete-se o processo para a transação {J: 1}, esse processo resulta na *FP-tree* condicional mostrada na figura 20.

Figura 20 – Construção da $Tree_F$ na segunda recursão



Fonte: Do Autor.

Criada a $Tree_F$ condicional, figura 22(b), o método verifica que a mesma não é vazia. Por isso uma chamada recursiva analisa_tree($Tree_F$, “HF”) é feita. Agora a $Tree_F$, passada como parâmetro, possui um único caminho. Então são gerados N *itemsets* para cada combinação dos nós no percurso. Como o caminho possui somente um nó, apenas um *itemset* é encontrado, o qual é: {(J,F,H: 2)}.

A presente recursão termina a rotina volta para a segunda recursão e continua a analisar a $Tree_H$. O laço de repetição da tabela de cabeçalhos termina de processar o item {F} e passa agora a analisar o item {G}. O método concatena o item {G} com o segundo parâmetro passado na chamada, que no caso é o item “H”, outro *itemset* frequente é identificado: {(G,H: 2)}. O passo seguinte é construir a base de dados condicional do item {G}. Por meio da tabela de cabeçalhos é possível identificar os caminhos que levam ao nó {G}, nesse caso somente um caminho é encontrado, {J:2}, porque o outro caminho que leva ao nó {G} é vazio devido ao nó raiz e ao sufixo não serem considerados. O contador de suporte do caminho encontrado é atualizado para o mesmo valor do item {G}, obtém-se assim sua respectiva base condicional {(J: 1)}. A base condicional possui uma transação, e será utilizada para construir a *FP-tree* condicional do item {F}. A tabela 15 mostra a primeira leitura da base condicional do item {G}.

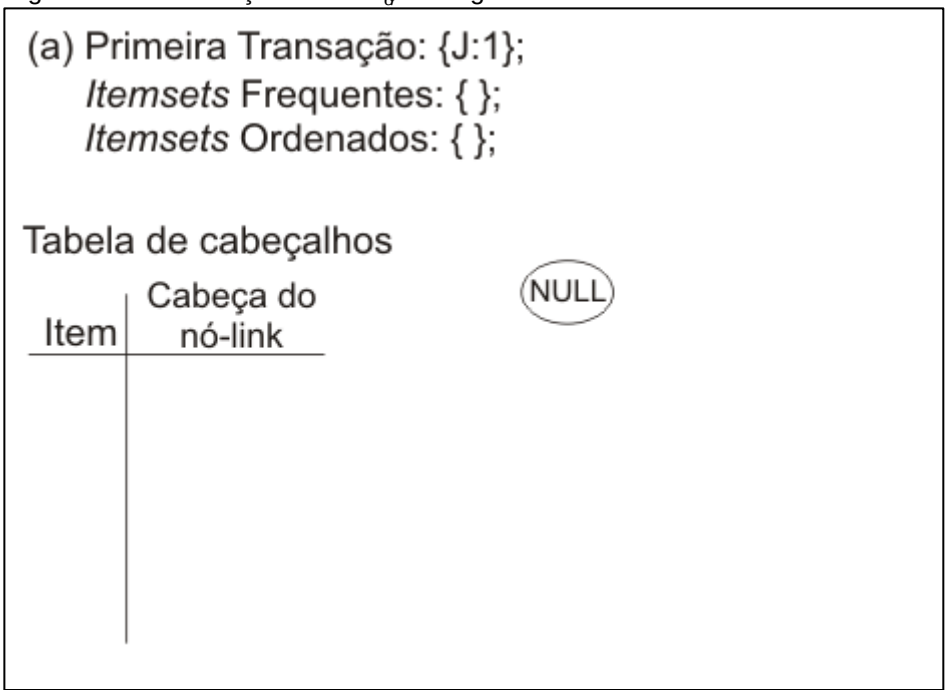
Tabela 15. Primeira leitura da base condicional do item {G}

Transações	Itens	1 *Itemsets*
1	(J: 1)	J:1

Fonte: Do Autor.

Começa a segunda leitura. A primeira transação da base de dados condicional, {J: 1}, é então lida e nenhum *itemset* será inserindo na $Tree_G$. Pois a base condicional não apresentou *itemset* com suporte maior ou igual ao suporte mínimo (*min_sup >= 20%*) no final da primeira leitura. Esse processo resulta na *FP-tree* condicional mostrada na figura 21.

Figura 21 – Construção da $Tree_G$ na segunda recursão



Fonte: Do Autor.

Criada a $Tree_G$ condicional, figura 23, o método verifica que a mesma é vazia. Por isso não é feita chamada recursiva. O laço de repetição da tabela de cabeçalhos termina de processar o item {G} da $Tree_H$ e passa agora a analisar o item {J}. O método concatena o item {J} com o segundo parâmetro passado na chamada, que no caso é o item “H”, outro *itemset* frequente é identificado: {(J,H: 2)}. O passo seguinte é construir a base de dados condicional do item {J}. Por meio da tabela de cabeçalhos é possível identificar os caminhos que levam ao nó {J}, nesse caso nenhum caminho é encontrado. A base condicional é vazia e a $Tree_J$ também o será. Devido a isso não é feita chamada recursiva. O laço de repetição da tabela de

cabeçalhos termina de processar o item {J} e como não tem mais itens na mesma. A análise da $Tree_H$, figura 21(c), termina.

O método volta para a primeira recursão e continua a analisar a *FP-tree* principal, figura 20(j). O laço de repetição da tabela de cabeçalhos termina de processar o item {H} e passa agora a processar o item {F}. Depois de processar o item {F} irá processar respectivamente os itens {G}, {N} e o {J} seguindo a mesma lógica. Após todos os 1 *itemsets* presentes na tabela de cabeçalhos serem processados o algoritmo *FP-Growth* mostra os N *itemsets* frequentes gerados. A tabela 16 mostra uma forma de apresentação do resultado da execução do algoritmo *FP-Growth*.

Tabela 16. Padrões frequentes gerados na modelagem.

Suporte	Padrões frequentes
80%	{J}
70%	{N}
60%	{G}
50%	{F}
30%	{H}
50%	{J, N}
40%	{J, G}
40%	{J, F}
20%	{J, H}
50%	{N, G}
30%	{N, F}
30%	{G, F}
20%	{G, H}
20%	{F, H}
30%	{J, N, G}
20%	{J, N, F}
20%	{J, G, F}
20%	{J, F, H}
20%	{N, G, F}

Fonte: Do Autor.

É importante salientar que caso o suporte mínimo fosse igual a 30% os N *itemsets* com suporte menor seriam considerados infrequentes e, portanto não seriam apresentados na tabela 13. Portanto quanto maior o suporte mínimo menor será o conjunto de N *itemsets* frequentes, o contrário também é verdadeiro.

Tendo como base o conjunto de N *itemsets* frequentes, tabela 16, o algoritmo *FP-Growth* gera as regras de associação. A tabela 17 mostra as regras geradas na presente modelagem matemática.

Tabela 17. Regras de associação geradas na modelagem.

Suporte	Regras de Associação
50%	{J} → {N}
50%	{N} → {J}
40%	{J} → {G}
40%	{G} → {J}
40%	{J} → {F}
40%	{F} → {J}
20%	{J} → {H}
20%	{H} → {J}
50%	{N} → {G}
50%	{G} → {N}
30%	{N} → {F}
30%	{F} → {N}
30%	{G} → {F}
30%	{F} → {G}
20%	{G} → {H}
20%	{H} → {G}
20%	{F} → {H}
20%	{H} → {F}
30%	{J} → {N, G}
30%	{N} → {J, G}
30%	{J, N} → {G}
30%	{G} → {J, N}
30%	{J, G} → {N}
30%	{N, G} → {J}
20%	{J} → {N, F}
20%	{N} → {J, F}
20%	{J, N} → {F}
20%	{F} → {J, N}
20%	{J, F} → {N}
20%	{N, F} → {J}
20%	{J} → {G, F}
20%	{G} → {J, F}
20%	{J, G} → {F}
20%	{F} → {J, G}
20%	{J, F} → {G}
20%	{G, F} → {J}
20%	{J} → {F, H}
20%	{F} → {J, H}
20%	{J, F} → {H}
20%	{H} → {J, F}
20%	{J, H} → {F}
20%	{F, H} → {J}
20%	{N} → {G, F}
20%	{G} → {N, F}
20%	{N, G} → {F}
20%	{F} → {N, G}
20%	{N, F} → {G}
20%	{G, F} → {N}

Fonte: Do Autor.

Analisando a tabela 14 verifica-se que os cinco primeiros itens da tabela 13 ({J}; {N}; {G}; {F}; {H}) não geraram regras de associação. Isso ocorre devido aos 1

itemsets frequentes não serem regras de associação e sim apenas *itemsets* frequentes de tamanho um.

Lembre-se que de uma forma geral, a tarefa de associação permite identificar o quanto a presença de um conjunto de itens nos registros de uma base de dados implica na presença de algum outro conjunto distinto de itens nos mesmos registros (MOTTA, 2010).

Outra conclusão facilmente identificada é o volume de conhecimento descoberto. De uma pequena base de dados com apenas 10 transações o algoritmo gerou 48 regras de associação. Em uma base de dados real o número de regras muitas vezes é tão grande que dificulta a análise das regras geradas, inviabilizando o seu uso no apoio a tomada de decisão. Para reduzir o número de regras geradas é necessário eliminar as redundâncias e as relações irrelevantes. Objetivando isso, técnicas de pós-processamento são aplicadas.

A partir deste momento começa a fase de pós-processamento das regras geradas. O algoritmo *FP-Growth* irá primeiro calcular a confiança de cada regra gerada, fará isso por meio da fórmula (4):

$$conf(A \Rightarrow B) = prob(A\,|\mathrm{B}) = \frac{\sup(A \cup B)}{\sup(A)} = \frac{\mathrm{n}(A \cup B)}{\mathrm{n}(A)} * 100 \qquad (4)$$

Considerando-se a tabela 14 e a fórmula (4) tem-se o calculo da confiança da seguinte forma:

$$conf(\{\mathrm{J}\} \Rightarrow \{\mathrm{N}\}) = prob(\{\mathrm{J}\}|\{\mathrm{N}\}) = \frac{\sup(\{\mathrm{J}\} \Rightarrow \{\mathrm{N}\})}{\sup(\{\mathrm{J}\})} = \frac{5}{8} * 100 = 62{,}50\%$$

$$conf(\{\mathrm{N}\} \Rightarrow \{\mathrm{J}\}) = prob(\{\mathrm{N}\}|\{\mathrm{J}\}) = \frac{\sup(\{\mathrm{N}\} \Rightarrow \{\mathrm{J}\})}{\sup(\{\mathrm{N}\})} = \frac{5}{7} * 100 = 71{,}43\%$$

$$conf(\{\mathrm{J}\} \Rightarrow \{\mathrm{G}\}) = prob(\{\mathrm{J}\}|\{\mathrm{G}\}) = \frac{\sup(\{\mathrm{J}\} \Rightarrow \{\mathrm{G}\})}{\sup(\{\mathrm{J}\})} = \frac{4}{8} * 100 = 50{,}00\%$$

$$conf(\{\mathrm{G}\} \Rightarrow \{\mathrm{J}\}) = prob(\{\mathrm{G}\}|\{\mathrm{J}\}) = \frac{\sup(\{\mathrm{G}\} \Rightarrow \{\mathrm{J}\})}{\sup(\{\mathrm{G}\})} = \frac{4}{6} * 100 = 66{,}67\%$$

Após calcular a confiança para cada regra o algoritmo elimina as regras que não satisfazem a confiança mínima de 60%, estipulada para a presente modelagem matemática. A tabela 18 mostra o resultado desta etapa.

Tabela 18. Regras de associação que satisfazem a confiança mínima.

Suporte	Regras de Associação	Confiança
50%	{J} → {N}	62,50%
50%	{N} → {J}	71,43%
40%	{G} → {J}	66,67%
40%	{F} → {J}	80,00%
20%	{H} → {J}	66,67%
50%	{N} → {G}	71,43%
50%	{G} → {N}	83,33%
30%	{F} → {N}	60,00%
30%	{F} → {G}	60,00%
20%	{H} → {G}	66,67%
20%	{H} → {F}	66,67%
30%	{J, N} → {G}	60,00%
30%	{J, G} → {N}	75,00%
30%	{N, G} → {J}	60,00%
20%	{N, F} → {J}	66,67%
20%	{G, F} → {J}	66,67%
20%	{H} → {J, F}	66,67%
20%	{J, H} → {F}	100,00%
20%	{F, H} → {J}	100,00%
20%	{N, F} → {G}	66,67%
20%	{G, F} → {N}	66,67%

Fonte: Do Autor.

Por meio do calculo da confiança, 27 regras de associação são descartadas, isso representa mais da metade, 21 regras satisfazem a confiança mínima estipulada. Porém, ainda verificam-se regras redundantes, e para uma base de dados com apenas 10 transações 21 regras de associação ainda é um número considerável. Portanto, é imprescindível identificar as regras que são, de fato, relevantes e úteis. Para isso outras medidas de qualidade têm sido propostas, dentre as quais as principais são (GONÇALVES, 2005; GUILLET; HAMILTON, 2007, tradução nossa; MELANDA, 2004):

a) ***Lift*:** também conhecida como *interest*, *lift* é utilizada para avaliar dependências, ou seja, esta medida de qualidade indica o quanto mais frequente torna-se B quando A ocorre. Pode-se dizer também que *lift* é a razão, em termos percentuais, entre a confiança obtida e a esperada (PIATETSKY-SHAPIRO; STEINGOLD, 2000, tradução nossa). É calculada por meio da fórmula (5):

$$Lift(A \Rightarrow B) = \frac{Conf(A \Rightarrow B)}{Sup(B)} \quad (5)$$

Considerando-se a tabela 15 e a fórmula (5) o *Lift* é calculado como segue:

$$Lift(J \Rightarrow N) = \frac{62{,}50}{0{,}7} = 89{,}29$$

$$Lift(N \Rightarrow J) = \frac{71{,}43}{0{,}8} = 89{,}29$$

$$Lift(G \Rightarrow J) = \frac{66{,}67}{0{,}8} = 83{,}33$$

$$Lift(F \Rightarrow J) = \frac{80{,}0}{0{,}8} = 100{,}00$$

Quando *lift*(A ⇒ B) é igual a 100, A e B são independentes. Se o valor do *lift*(A ⇒ B) é maior que 100, A e B são positivamente dependentes. E quando *lift*(A ⇒ B) for menor que 100, A e B são negativamente dependentes. O resultado desta medida varia entre 0 e +∞. Sua interpretação é simples: quanto maior for o valor do *lift*, mais interessante é a regra, pois A aumentou B em uma taxa maior (PIATETSKY-SHAPIRO; STEINGOLD, 2000, tradução nossa). Para a presente modelagem matemática utilizou-se o valor 100 como *lift* mínimo objetivando eliminar as regras negativamente dependentes;

b) ***Leverage:*** esta medida é utilizada com o objetivo de se descobrir o valor da diferença entre o suporte real e o esperado de uma regra de associação (PIATETSKY-SHAPIRO, 1991, tradução nossa). É calculado conforme a fórmula (6):

$$Leverage(A \Rightarrow B) = Sup(A \Rightarrow B) - (Sup(A) \times Sup(B)) \quad (6)$$

Considerando-se a tabela 15 e a fórmula (6) o *Leverage* é calculado assim:

$$Leverage(J \Rightarrow N) = 0{,}5 - (0{,}8 \times 0{,}7) = -0{,}06$$

$$Leverage(N \Rightarrow J) = 0{,}5 - (0{,}7 \times 0{,}8) = -0{,}06$$

$$Leverage(G \Rightarrow J) = 0{,}4 - (0{,}6 \times 0{,}8) = -0{,}08$$

$$Leverage(F \Rightarrow J) = 0{,}4 - (0{,}5 \times 0{,}8) = 0{,}00$$

Quando *leverage*(A ⇒ B) é igual a 0, então A e B são independentes. Se o valor do *leverage*(A ⇒ B) é maior que 0, então A e B são positivamente dependentes. Já quando *leverage*(A ⇒ B) < 0, A e B são negativamente dependentes. O valor do resultado dessa medida varia entre -0.25 e 0.25. Quanto maior o valor da medida, mais interessante é a regra (PIATETSKY-SHAPIRO, 1991, tradução nossa). Utilizou-se o valor 0 (zero) como valor mínimo para o *leverage*, na presente modelagem matemática, objetivando-se eliminar as regras negativamente dependentes;

c) ***Convicção:*** o *lift* e o *leverage* possuem como característica o fato de serem medidas simétricas, ou seja, *Lift*(A ⇒ B) = *Lift*(B ⇒ A) e *leverage*(A ⇒ B) = *leverage*(B ⇒ A). Isto ocorre porque estes índices possuem o objetivo de medir a dependência entre os itens, ao invés de mensurar implicação (o sentido da seta "⇒"). Esta medida foi proposta com o objetivo de avaliar uma regra de associação como uma verdadeira implicação (BRIN et al, 1997, tradução nossa). O valor da convicção é calculado por meio da fórmula (7):

$$Conv(A \Rightarrow B) = \frac{Sup(A) \times Sup(\neg B)}{Sup(A\ U\ \neg B)} \qquad (7)$$

Considerando-se a tabela 15 e a fórmula (7) a *Convicção* é assim calculada:

$$Conv(J \Rightarrow N) = \frac{0{,}8 \times (1 - 0{,}7)}{(0{,}8 - 0{,}5) + 0{,}1} = 0{,}60$$

$$Conv(N \Rightarrow J) = \frac{0{,}7 \times (1 - 0{,}8)}{(0{,}7 - 0{,}5) + 0{,}1} = 0{,}47$$

$$Conv(G \Rightarrow J) = \frac{0{,}6 \times (1 - 0{,}8)}{(0{,}6 - 0{,}4) + 0{,}1} = 0{,}40$$

$$Conv(F \Rightarrow J) = \frac{0{,}5 \times (1 - 0{,}8)}{(0{,}5 - 0{,}4) + 0{,}1} = 0{,}50$$

O valor da medida de qualidade convicção varia entre 0 e +∞. Os desenvolvedores da medida (BRIN et al, 1997, tradução nossa) concluíram que as regras mais interessantes apresentaram um valor de convicção entre 1,01 e 5. Concluíram também que muitas das regras com valor de Convicção acima de 5 representaram informações ilusórias ou óbvias. Na presente modelagem matemática optou-se pelo valor de 0,9 como valor mínimo para a medida convicção, pois representa um valor próximo do ideal, enquanto um valor dentro do ideal, para a presente modelagem matemática, não geraria nenhuma regra de associação, isto porque o valor máximo obtido foi 1 (tabela 16);

A fase de pós-processamento implementada no algoritmo *FP-Growth* na *Shell Orion* calcula a confiança, o *lift*, o *leverage* e a *convicção* para cada regra de associação gerada. Somente as regras de associação que satisfazerem todas as medidas de qualidade serão consideradas relevantes e úteis. A tabela 19 mostra as regras de associação que satisfazem os valores mínimos estipulados.

Tabela 19. Regras de associação relevantes e úteis.

Suporte	Regras de Associação	Confiança	*Lift*	*Leverage*	Convicção
50%	{N} → {G}	71,43%	119,05	0,08	0,93
50%	{G} → {N}	83,33%	119,05	0,08	0,90
20%	{H} → {J, F}	66,67%	166,67	0,08	0,90
20%	{J, H} → {F}	100,00%	200,00	0,10	1,00

Fonte: Do Autor.

Empregando-se as medidas de qualidade *Lift*, *Leverage* e *Convicção* mais 17 regras de associação são descartadas, e apenas quatro regras satisfazem os parâmetros estipulados.

Salienta-se que a mudança nos valores mínimos estipulados nos parâmetros de entrada do algoritmo, pode aumentar ou diminuir o número de regras de associação geradas, portanto, estas merecem uma análise especial.

O entendimento do funcionamento do algoritmo *FP-Growth* possibilitou sua implementação no módulo de associação da *Shell Orion Data Mining Engine*.

6.2.3 Implementação e Realização de Testes

O algoritmo *FP-Growth* foi implementado no módulo de associação da *Shell Orion Data Mining Engine*, por meio da linguagem de programação Java e do ambiente de programação integrado *NetBeans* 7.3.1[10].

A *Shell Orion* permite que sejam estabelecidas conexões com diversos Sistemas Gerenciadores de Bancos de Dados (SGBD), desde que esses SGBDs disponibilizem um driver JDBC para realizar a conexão. Já encontra-se a disposição do usuário os seguintes SGBDs: *Firebird*, HSQLDB, *MySQL*, *Oracle Express Edition*, *PostgreSQL* e *SQLAnywhere*.

Para a presente implementação e realização de testes do algoritmo *FP-Growth*, o SGBD escolhido foi o HSQLDB[11], por ser gratuito e não necessitar de instalação prévia o que o torna de simples portabilidade.

Concluída a definição do SGBD, foram realizados testes com as bases de dados selecionadas para verificar os resultados obtidos pelo algoritmo implementado. Dentre as bases de dados escolhidas uma contém dados clínicos de pacientes com hepatite e a outra contém dados da votação de uma seção do congresso dos Estados Unidos da América.

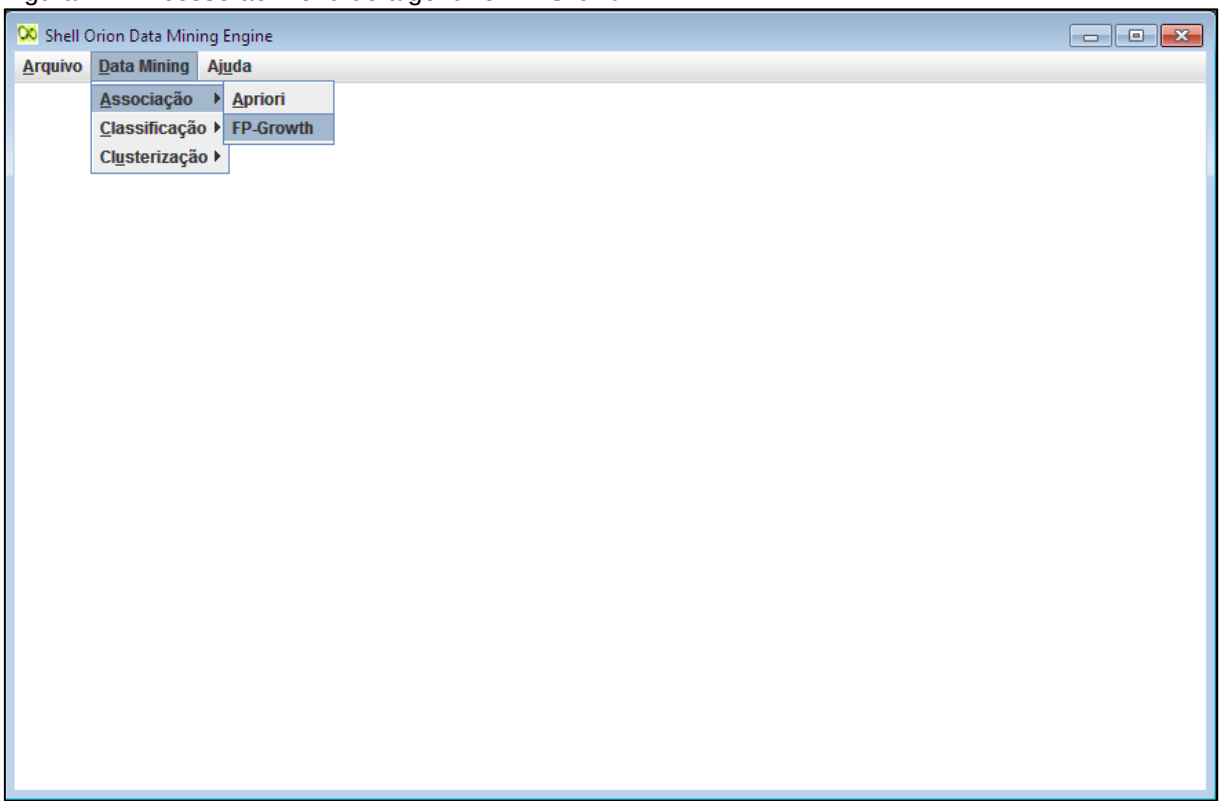
Tendo-se os dados devidamente pré-processados e normalizados, os mesmos devem ser inseridos no HSQLDB por meio de comandos SQL, o próximo passo é realizar a conexão da *Shell Orion* com o SGBD escolhido no menu *Arquivo*, submenu *Conectar*.

---

[10] Disponível para download gratuitamente no seguinte endereço eletrônico: http://netbeans.org.
[11] O HSQLDB é um SGBD relacional totalmente escrito na linguagem de programação Java e está disponível para download gratuitamente em: http://hsqldb.org.

O algoritmo *FP-Growth* só estará acessível a partir do momento que é estabelecida a conexão com o SGBD escolhido, por meio do menu *Data Mining,* submenu *Associação*, selecionando-se o algoritmo *FP-Growth* (Figura 22).

Figura 22 – Acesso ao menu do algoritmo *FP-Growth*.



Fonte: Do Autor.

A tarefa de associação por meio do algoritmo *FP-Growth* exige que o usuário informe alguns parâmetros de entrada (figura 23). No quadrante superior esquerdo denominado *Parâmetros do Algoritmo* estão parâmetros relacionados à execução do algoritmo e no quadrante superior direito denominado *Medidas de Qualidade* estão parâmetros que são utilizados para se medir a qualidade das regras geradas.

Os parâmetros solicitados ao usuário são:

a) **suporte:** por meio deste parâmetro o usuário específica o valor mínimo para o suporte dos *N itemsets* frequentes que irão compor as regras de associação geradas;

b) **minerar atributos no formato *basket*:** parâmetro que quando selecionado permite analisar bases de dados onde a quantidade de

itens que pode constar em uma transação (TID) não é limitada ao número de atributos da relação;

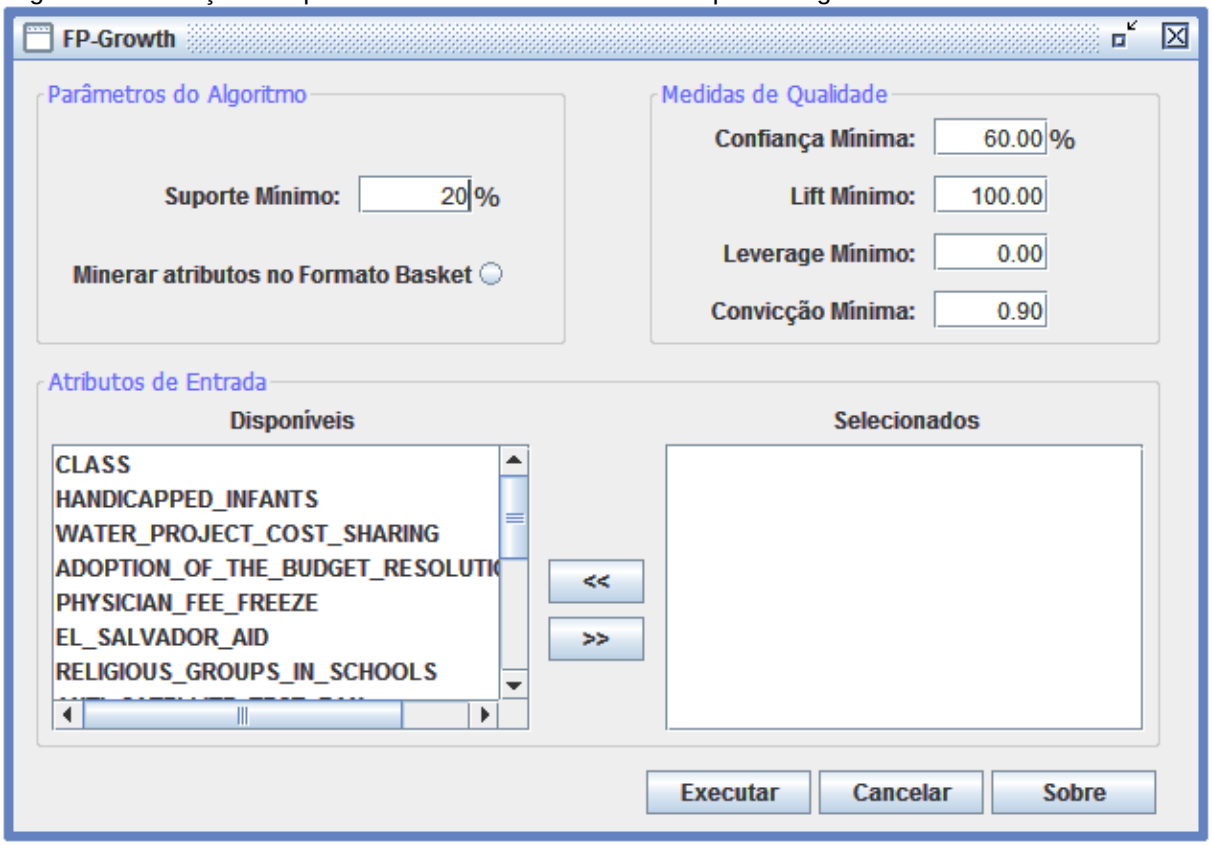
c) **confiança:** trata-se de uma medida de qualidade para a regra de associação, ou seja, é utilizada após o algoritmo ter descoberto todos os *N itemsets* frequentes. A confiança mede a força da regra, ou seja, a probabilidade da ocorrência do antecedente da regra, quando o consequente ocorre;

d) ***lift*:** o usuário irá utilizar este parâmetro para estipular o *Lift Mínimo*. *Lift* é utilizada para avaliar dependências, ou seja, esta medida de qualidade indica o quanto mais frequente torna-se o consequente da regra quando o antecedente ocorre. Pode-se dizer também que *lift* é a razão, em termos percentuais, entre confiança e confiança esperada;

e) ***leverage*:** esta medida é utilizada com o objetivo de se descobrir o valor da diferença entre o suporte real e o esperado de uma regra de associação;

f) ***convicção*:** tem por objetivo avaliar uma regra de associação como uma implicação (o sentido da seta "⇒"). Diferentemente do *lift* e do *leverage* que possuem como característica o fato de serem medidas simétricas, ou seja, *Lift*(A ⇒ B) = *Lift*(B ⇒ A) e *leverage*(A ⇒ B) = *leverage*(B ⇒ A), uma vez que estes índices possuem o objetivo de medir a dependência entre os itens.

Figura 23 – Seleção dos parâmetros e atributos de entrada para o algoritmo *FP-Growth*.

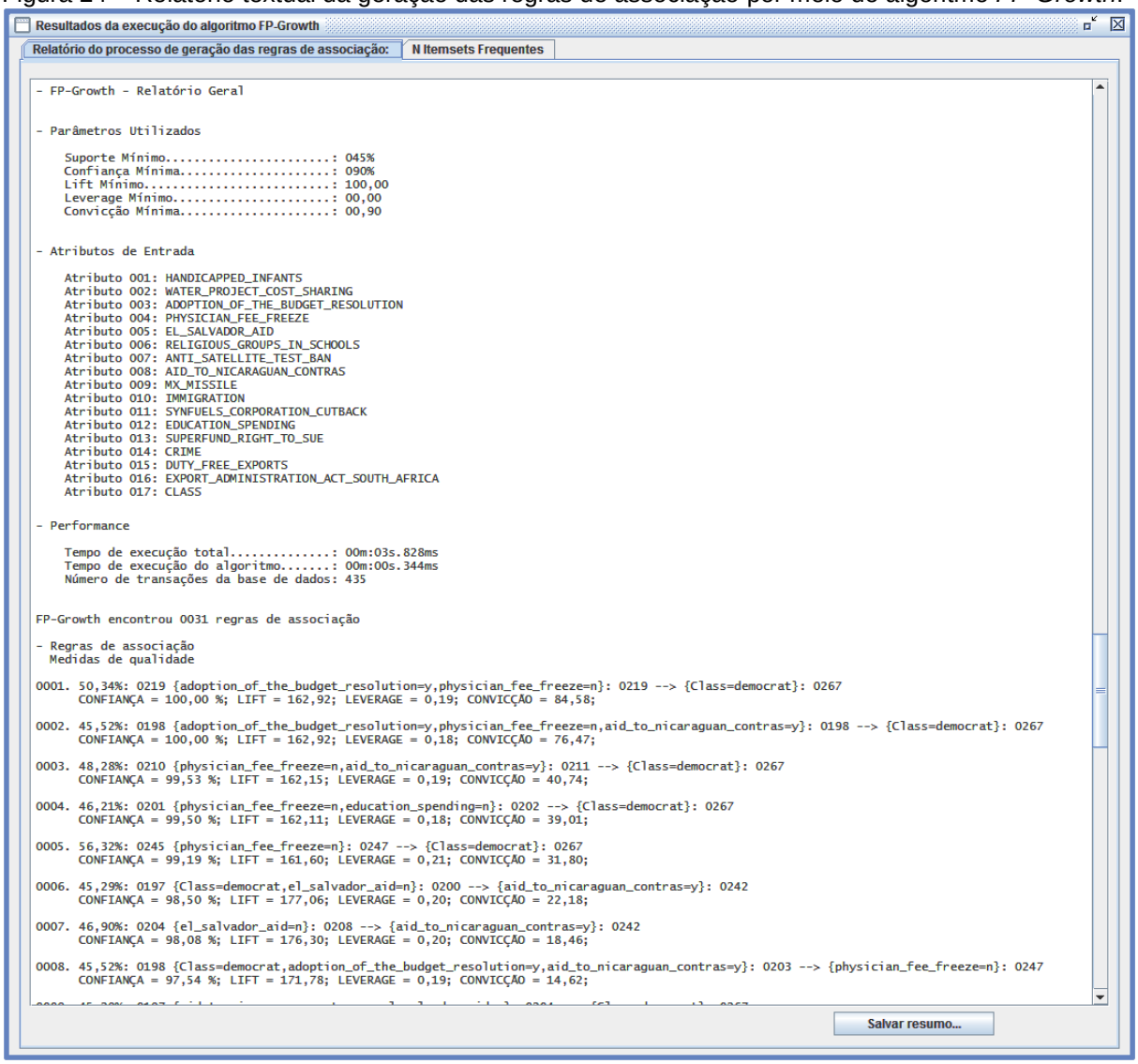


Fonte: Do Autor.

Os atributos de entrada selecionados e valores preenchidos para os parâmetros do algoritmo interferem no resultado apresentado ao usuário, o que possibilita obter diferentes resultados de acordo com os valores informados pelo usuário, em uma mesma base de dados. Portanto, devem ser cuidadosamente preenchidos e selecionados para se obter resultados adequados.

Os resultados obtidos pelo algoritmo *FP-Growth* na *Shell Orion*, podem ser analisados por meio de relatórios. A figura 24 mostra um relatório textual gerado pelo algoritmo contendo os valores informados para os parâmetros de entrada; os atributos de entrada selecionados; os tempos de execução do algoritmo e o tempo total (execução do algoritmo mais montagem do relatório); número de transações presentes na base de dados; as regras de associação geradas com o percentual e a quantidade de suporte; e os valores das medidas de qualidade. Também é possível exportar os resultados obtidos para um arquivo texto por meio do botão *Salvar resumo*.

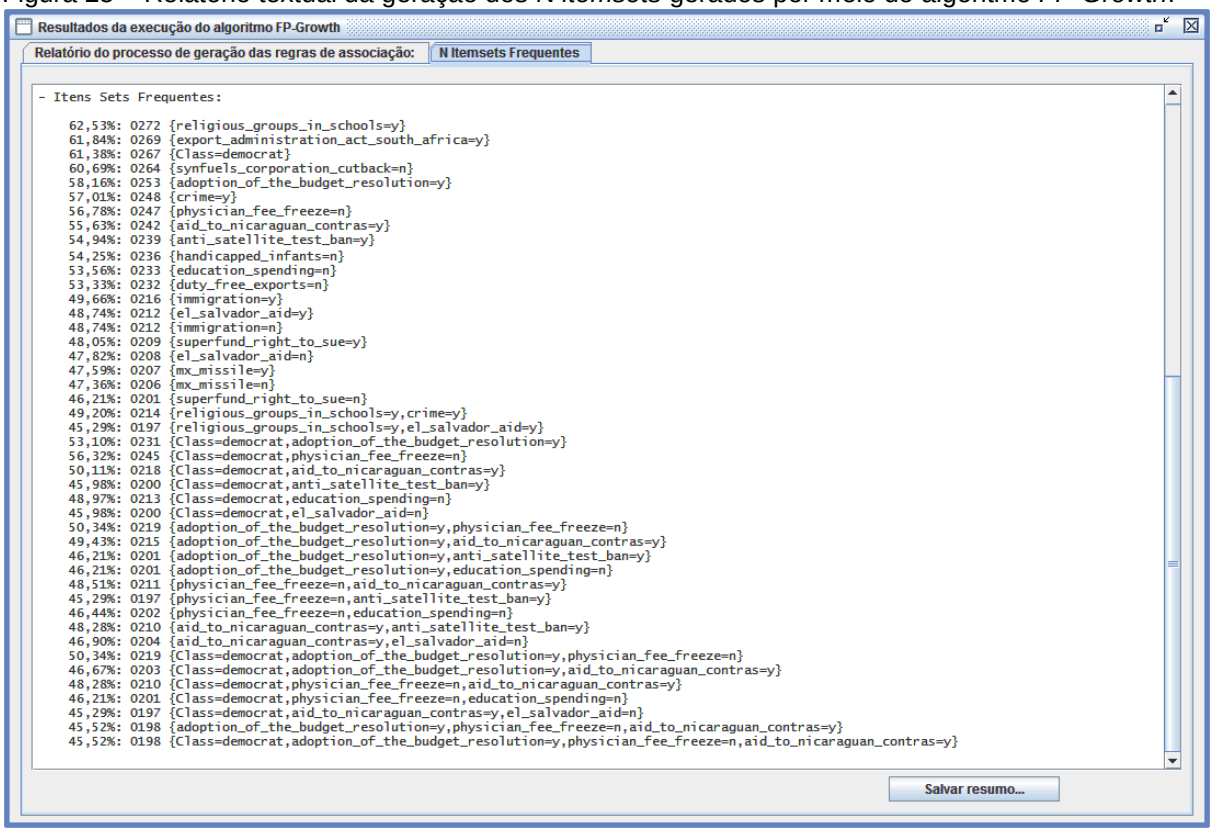
Figura 24 – Relatório textual da geração das regras de associação por meio do algoritmo *FP-Growth*.



Fonte: Do Autor.

Na figura 25 tem-se outro relatório gerado pela *Shell Orion* contendo informações dos resultados do algoritmo *FP-Growth*. Neste são listados os N *itemsets* frequentes encontrados com o percentual e a quantidade de suporte. Decidiu-se criar um relatório em separado para os N *itemsets* afim de facilitar a análise do usuário e não sobrecarregar o relatório principal que contém as regras de associação.

Figura 25 – Relatório textual da geração dos N *itemsets* gerados por meio do algoritmo *FP-Growth*.



Fonte: Do Autor.

Na tarefa de associação as regras não costumam ser apresentadas em forma de gráficos ou árvores, uma vez que até mesmo não especialistas em DM compreendem as regras geradas devido a sua compreensibilidade inerente. A pesar do algoritmo *FP-Growth* usar árvores, isso não é importante para análise dos resultados obtidos, pois a árvore FP é um meio utilizado pelo algoritmo *FP-Growth* para gerar regras de associação sem para isso gerar *itemsets* candidatos (MELANDA, 2004; SCHONHORST, 2010).

Tendo-se concluído a implementação, testes foram efetuados objetivando a comprovação dos resultados obtidos. Também verificou-se os tempos de execução do algoritmo.

6.3 RESULTADOS OBTIDOS

Finalizada a etapa da implementação, realizou-se a análise das regras de associação identificadas; a relevância das regras por meio de medidas de qualidade; a análise de desempenho por meio da avaliação dos resultados observados; o

tempo de execução do algoritmo; e a comparação do mesmo com o modelo implementado na ferramenta Weka.

Para a realização dos testes com o algoritmo *FP-Growth* utilizou-se um microcomputador com sistema operacional Windows7, 4GB de memória RAM e processador Intel Core 2 Duo 2.93 GHz.

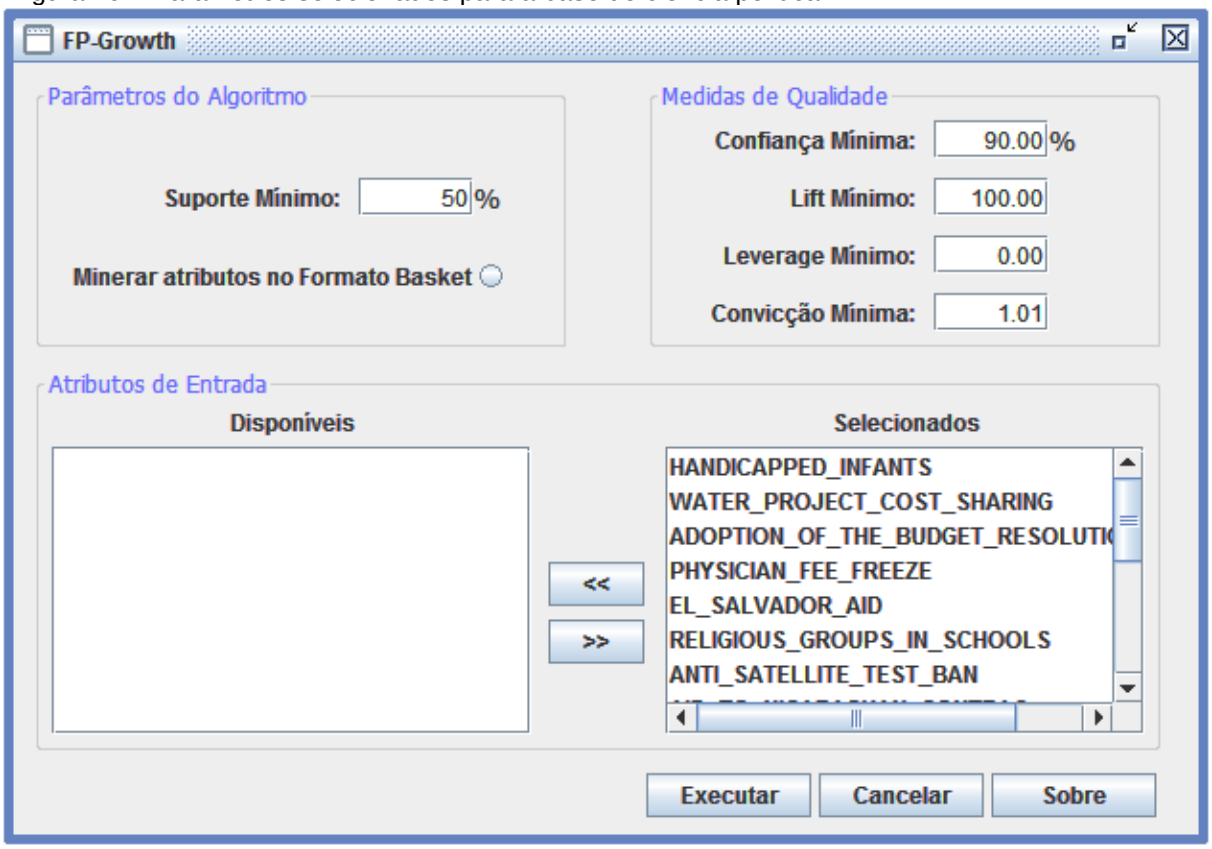
6.3.1 Regras de Associação Identificadas pelo Algoritmo *FP-Growth*

Utilizou-se uma base de dados na área da saúde e outra base de dados na área de ciência política para avaliar a capacidade do algoritmo *FP-Growth*, implementado na *Shell Orion*, em identificar corretamente regras de associação. Executou-se o algoritmo utilizando-se os seguintes parâmetros e atributos de entrada:

a) **suporte:** optou-se por utilizar um percentual de 50% com o intuito de analisar as regras mais fortes das bases de dados escolhidas;
b) **minerar atributos no formato basket:** este parâmetro ficou desmarcado para os testes realizados;
c) **confiança:** usou-se um percentual de 90% para também analisar as regras mais fortes;
d) ***lift*:** o valor escolhido para o *lift* foi 100 para eliminar as regras negativamente dependentes;
e) ***leverage*:** o valor informado para o *leverage* foi 0 (zero) também objetivando-se eliminar as regras negativamente dependentes;
f) ***convicção*:** informou-se o valor 1.01 para *convicção* por ser o valor mínimo ideal.

Na figura 26 verifica-se os parâmetros de entrada selecionados para a base de dados da área de ciência política e que foram usados pelo algoritmo *FP-Growth*.

Figura 26 – Parâmetros selecionados para a base de ciência política.



Fonte: Do Autor.

Na figura 27 é possível verificar as regras geradas pelo algoritmo *FP-Growth* usando os parâmetros e atributos mostrados na figura 26, e a base de dados na área de ciência política.

Figura 27 – Regras encontradas com os parâmetros informados.

```
FP-Growth encontrou 0006 regras de associação

- Regras de associação
  Medidas de qualidade

0001. 50,34%: 0219 {adoption_of_the_budget_resolution=y,physician_fee_freeze=n}: 0219 --> {Class=democrat}: 0267
      CONFIANÇA = 100,00 %; LIFT = 162,92; LEVERAGE = 0,19; CONVICÇÃO = 84,58;

0002. 56,32%: 0245 {physician_fee_freeze=n}: 0247 --> {Class=democrat}: 0267
      CONFIANÇA = 99,19 %; LIFT = 161,60; LEVERAGE = 0,21; CONVICÇÃO = 31,80;

0003. 50,34%: 0219 {Class=democrat,adoption_of_the_budget_resolution=y}: 0231 --> {physician_fee_freeze=n}: 0247
      CONFIANÇA = 94,81 %; LIFT = 166,96; LEVERAGE = 0,20; CONVICÇÃO = 7,68;

0004. 56,32%: 0245 {Class=democrat}: 0267 --> {physician_fee_freeze=n}: 0247
      CONFIANÇA = 91,76 %; LIFT = 161,60; LEVERAGE = 0,21; CONVICÇÃO = 5,02;

0005. 53,10%: 0231 {adoption_of_the_budget_resolution=y}: 0253 --> {Class=democrat}: 0267
      CONFIANÇA = 91,30 %; LIFT = 148,75; LEVERAGE = 0,17; CONVICÇÃO = 4,25;

0006. 50,11%: 0218 {aid_to_nicaraguan_contras=y}: 0242 --> {Class=democrat}: 0267
      CONFIANÇA = 90,08 %; LIFT = 146,76; LEVERAGE = 0,16; CONVICÇÃO = 3,74;
```

Fonte: Do Autor.

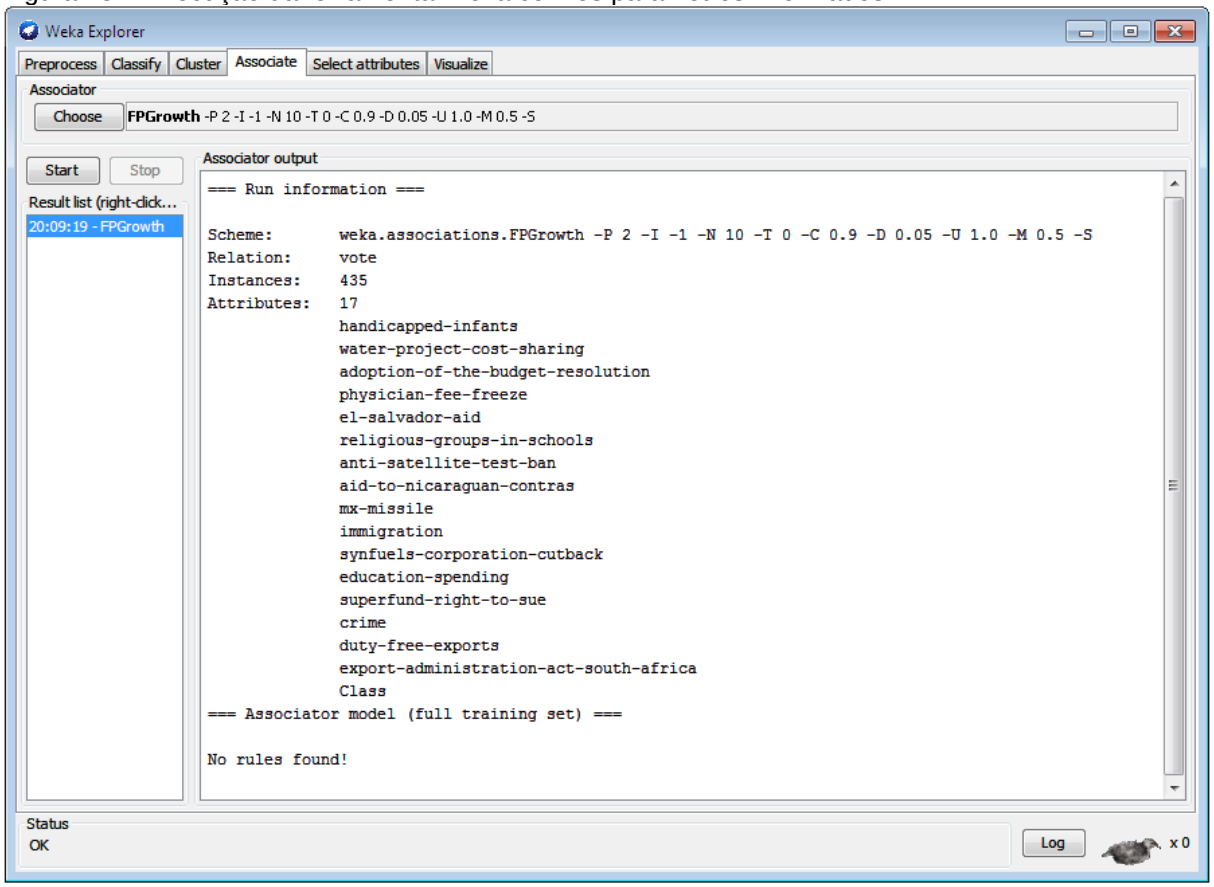
Os testes de mesa[12] necessários para se testar a implementação do algoritmo, foram feitos com o exemplo mostrado na modelagem matemática, por isso a primeira análise dos resultados feita nessa fase da pesquisa foi uma comparação com o resultado gerado pelo *FP-Growth* na ferramenta Weka[13].

A *Waikato Environment for Knowledge Analysis* (Weka) é uma ferramenta desenvolvida na Universidade de Waikato, na Nova Zelândia, nela são implementados diversos algoritmos e técnicas de *data mining* por meio da linguagem Java, o que permite a sua execução em diversas plataformas. Frequentemente utilizada em pesquisas na área de *data mining*, por se tratar de um *software* livre distribuído sob os termos da GNU[14] (WITTEN; FRANK; HALL, 2011, tradução nossa).

Estudou-se a ferramenta Weka em fóruns, tutoriais disponíveis na *internet* e no livro publicado por Witten, Frank e Hall no ano de 2011. Configurou-se a mesma para obedecer os mesmos parâmetros usados na *Shell Orion*. No primeiro teste não gerou nenhuma regra de associação (figura 28).

[12] Procedimento que simula a execução de um algoritmo utilizando apenas papel e caneta, por meio do teste de mesa é possível verificar as mudanças no conteúdo das variáveis de um algoritmo instrução a instrução, facilitando ao desenvolvedor identificar e corrigir comportamentos falhos (DEITEL; DEITEL, 2003).
[13] Disponibilizada gratuitamente em (http://www.cs.waikato.ac.nz/~ml/weka/).
[14] General Public License. Informações em (http://www.gnu.org).

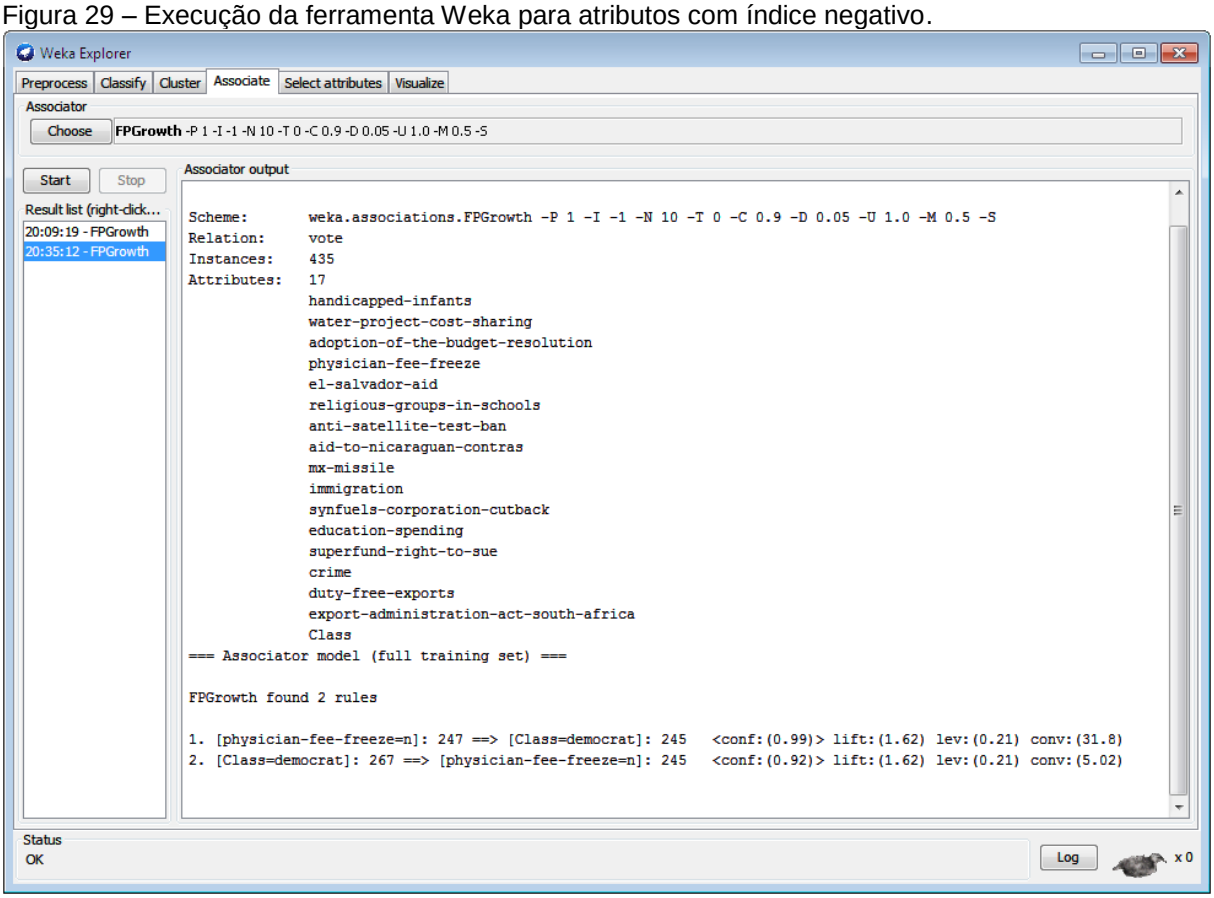
Figura 28 – Execução da ferramenta Weka com os parâmetros informados.



Fonte: Do Autor.

Após uma nova série de testes, estudos e exploração na ferramenta Weka, constatou-se que o algoritmo *FP-Growth* implementado na Weka trabalha somente com índices binários, isto é, com atributos que possuem dois valores (SIM = + / NÃO = -). E gera as regras considerando somente os valores positivos ou negativos dos atributos na base de dados. Descobriu-se que por *default* a ferramenta vem selecionada para gerar regras com os atributos que possuem índice positivo. Alterou-se para gerar com os atributos que possuem índice negativo. O resultado é apresentado na figura 29.

Figura 29 – Execução da ferramenta Weka para atributos com índice negativo.



Fonte: Do Autor.

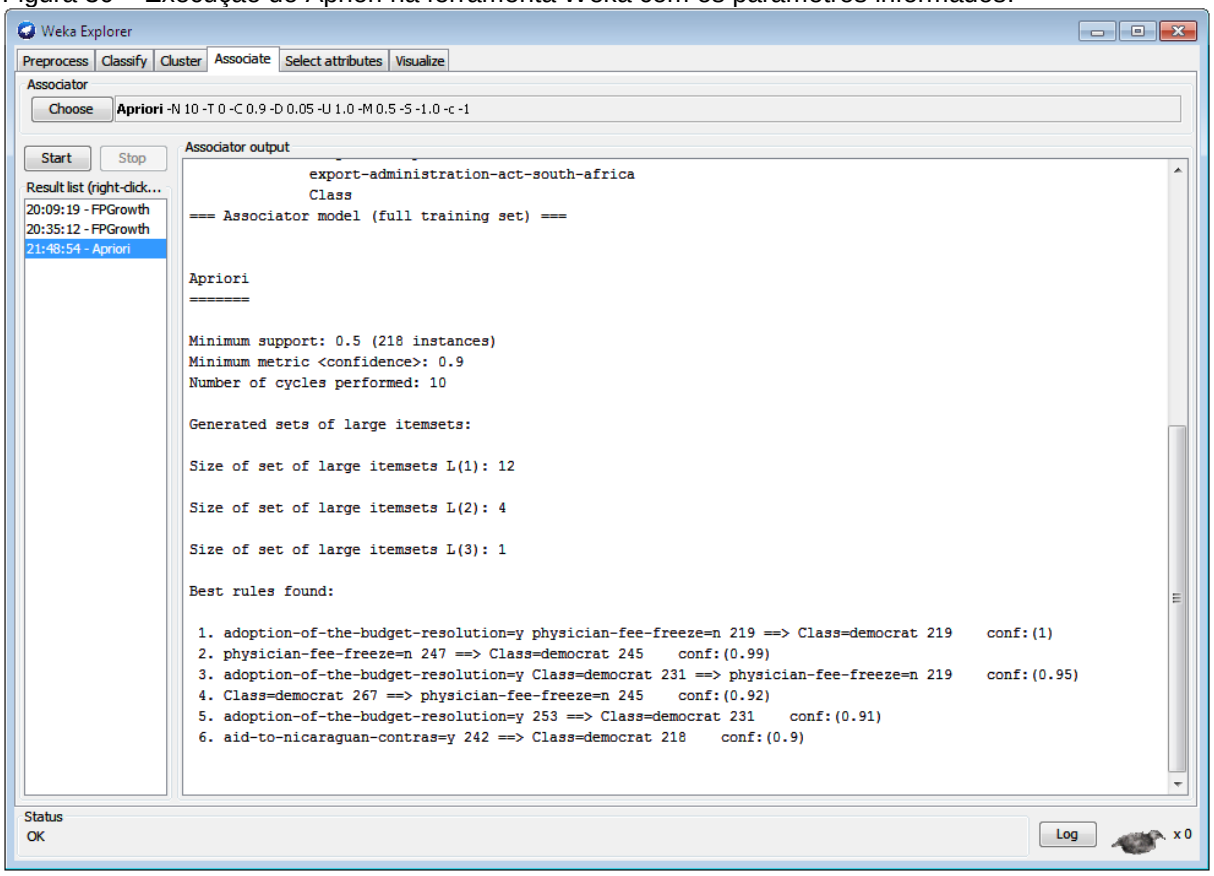
Desta vez a ferramenta Weka gerou duas regras de associação. Comparando-se estas duas regras com as regras geradas pela *Shell Orion* verifica-se que a regra um na Weka é igual a regra dois gerada na *Shell Orion* e a regra dois na Weka igual a quatro na *Shell Orion*. Inclusive os valores para as medidas de qualidade também são os mesmos diferenciando-se somente na forma como ambas mostram o resultado.

As demais regras geradas na *Shell Orion* possuem índice negativo e positivo enquanto que na implementação do *FP-Growth* na Weka as regras são somente com índice negativo ou somente com índice positivo. Verifica-se isso na regra um (figura 27), gerada na *Shell Orion*. Onde o antecedente da regra possui um *itemset* com índice positivo e outro com índice negativo (*{adoption_of_the_budget_resolution=y, physician_fee_freeze=n}: 0219*).

Uma vez que o algoritmo *FP-Growth* foi desenvolvido parar gerar regras de associação sem para isso gerar candidato, o que resulta em um melhor aproveitamento dos recostos computacionais disponíveis, resolveu-se realizar uma comparação com o resultado do algoritmo Apriori também implementado na Weka.

O resultado tem que ser o mesmo, ou no mínimo parecido, porque tanto o Apriori quanto o *FP-Growth* usam meios diferentes mas chegam a um mesmo fim, que é a descoberta de regras de associação úteis e fortes. O resultado da execução o algoritmo Apriori na Weka é mostrado na figura 30.

Figura 30 – Execução do Apriori na ferramenta Weka com os parâmetros informados.



Fonte: Do Autor.

Como resultado observou-se que o algoritmo Apriori implementado na ferramenta Weka gerou as mesmas regras com os mesmos parâmetros informados, bem como os mesmos percentuais de suporte e confiança. Conclui-se assim que as regras geradas pelo algoritmo *FP-Growth* implementado na *Shell Orion* estão corretas, pois *FP-Growth* foi desenvolvido para gerar regras de associação sem, para isso, gerar candidato (HAN et al, 2004, tradução nossa; HAN; PEI; YIN, 2000, tradução nossa). A figura 31 mostra um resumo das regras geradas na *Shell Orion* e na Weka.

Figura 31 – Regras geradas na *Shell Orion* e na Weka com base de dados da área de ciência política.

Shell Orion

```
FP-Growth encontrou 0006 regras de associação

- Regras de associação
  Medidas de qualidade

0001. 50,34%: 0219 {adoption_of_the_budget_resolution=y,physician_fee_freeze=n}: 0219 --> {Class=democrat}: 0267
      CONFIANÇA = 100,00 %; LIFT = 162,92; LEVERAGE = 0,19; CONVICÇÃO = 84,58;

0002. 56,32%: 0245 {physician_fee_freeze=n}: 0247 --> {Class=democrat}: 0267
      CONFIANÇA = 99,19 %; LIFT = 161,60; LEVERAGE = 0,21; CONVICÇÃO = 31,80;

0003. 50,34%: 0219 {Class=democrat,adoption_of_the_budget_resolution=y}: 0231 --> {physician_fee_freeze=n}: 0247
      CONFIANÇA = 94,81 %; LIFT = 166,96; LEVERAGE = 0,20; CONVICÇÃO = 7,68;

0004. 56,32%: 0245 {Class=democrat}: 0267 --> {physician_fee_freeze=n}: 0247
      CONFIANÇA = 91,76 %; LIFT = 161,60; LEVERAGE = 0,21; CONVICÇÃO = 5,02;

0005. 53,10%: 0231 {adoption_of_the_budget_resolution=y}: 0253 --> {Class=democrat}: 0267
      CONFIANÇA = 91,30 %; LIFT = 148,75; LEVERAGE = 0,17; CONVICÇÃO = 4,25;

0006. 50,11%: 0218 {aid_to_nicaraguan_contras=y}: 0242 --> {Class=democrat}: 0267
      CONFIANÇA = 90,08 %; LIFT = 146,76; LEVERAGE = 0,16; CONVICÇÃO = 3,74;
```

Weka

```
=== Associator model (full training set) ===

FPGrowth found 2 rules

1. [physician-fee-freeze=n]: 247 ==> [Class=democrat]: 245   <conf:(0.99)> lift:(1.62) lev:(0.21) conv:(31.8)
2. [Class=democrat]: 267 ==> [physician-fee-freeze=n]: 245   <conf:(0.92)> lift:(1.62) lev:(0.21) conv:(5.02)
```

Weka Apriori

```
Best rules found:

 1. adoption-of-the-budget-resolution=y physician-fee-freeze=n 219 ==> Class=democrat 219    conf:(1)
 2. physician-fee-freeze=n 247 ==> Class=democrat 245    conf:(0.99)
 3. adoption-of-the-budget-resolution=y Class=democrat 231 ==> physician-fee-freeze=n 219    conf:(0.95)
 4. Class=democrat 267 ==> physician-fee-freeze=n 245    conf:(0.92)
 5. adoption-of-the-budget-resolution=y 253 ==> Class=democrat 231    conf:(0.91)
 6. aid-to-nicaraguan-contras=y 242 ==> Class=democrat 218    conf:(0.9)
```

Fonte: Do Autor.

Para a análise da base de dados da área da saúde contendo informações de pacientes com hepatite optou-se por aumentar o percentual do suporte para 70% para analisar as regras mais fortes. A figura 32 mostra o resultado obtido na *Shell Orion* e na Weka.

Figura 32 – Regras geradas na *Shell Orion* e na Weka com base de dados da área da saúde.

Shell Orion

```
FP-Growth encontrou 0007 regras de associação

- Regras de associação
  Medidas de qualidade

0001. 77,42%: 0120 {ASCITES=2}: 0130 --> {VARICES=2}: 0132
      CONFIANÇA = 92,31 %; LIFT = 108,39; LEVERAGE = 0,06; CONVICÇÃO = 1,75;

0002. 72,90%: 0113 {Class=2}: 0123 --> {ASCITES=2}: 0130
      CONFIANÇA = 91,87 %; LIFT = 109,54; LEVERAGE = 0,06; CONVICÇÃO = 1,80;

0003. 70,97%: 0110 {SPLEEN_PALPABLE=2}: 0120 --> {VARICES=2}: 0132
      CONFIANÇA = 91,67 %; LIFT = 107,64; LEVERAGE = 0,05; CONVICÇÃO = 1,62;

0004. 72,26%: 0112 {Class=2}: 0123 --> {VARICES=2}: 0132
      CONFIANÇA = 91,06 %; LIFT = 106,92; LEVERAGE = 0,05; CONVICÇÃO = 1,52;

0005. 77,42%: 0120 {VARICES=2}: 0132 --> {ASCITES=2}: 0130
      CONFIANÇA = 90,91 %; LIFT = 108,39; LEVERAGE = 0,06; CONVICÇÃO = 1,64;

0006. 76,77%: 0119 {ANTIVIRALS=2}: 0131 --> {SEX=1}: 0139
      CONFIANÇA = 90,84 %; LIFT = 101,30; LEVERAGE = 0,01; CONVICÇÃO = 1,04;

0007. 70,32%: 0109 {SPLEEN_PALPABLE=2}: 0120 --> {SEX=1}: 0139
      CONFIANÇA = 90,83 %; LIFT = 101,29; LEVERAGE = 0,01; CONVICÇÃO = 1,03;
```

Weka

```
=== Associator model (full training set) ===

FPGrowth found 5 rules

1. [ASCITES=2]: 130 ==> [VARICES=2]: 120   <conf:(0.92)> lift:(1.08) lev:(0.06) conv:(1.75)
2. [Class=2]: 123 ==> [ASCITES=2]: 113   <conf:(0.92)> lift:(1.1) lev:(0.06) conv:(1.8)
3. [SPLEEN_PALPABLE=2]: 120 ==> [VARICES=2]: 110   <conf:(0.92)> lift:(1.08) lev:(0.05) conv:(1.62)
4. [Class=2]: 123 ==> [VARICES=2]: 112   <conf:(0.91)> lift:(1.07) lev:(0.05) conv:(1.52)
5. [VARICES=2]: 132 ==> [ASCITES=2]: 120   <conf:(0.91)> lift:(1.08) lev:(0.06) conv:(1.64)
```

Weka Apriori

```
Best rules found:

 1. ASCITES=2 130 ==> VARICES=2 120    conf:(0.92)
 2. Class=2 123 ==> ASCITES=2 113    conf:(0.92)
 3. SPLEEN_PALPABLE=2 120 ==> VARICES=2 110    conf:(0.92)
 4. Class=2 123 ==> VARICES=2 112    conf:(0.91)
 5. VARICES=2 132 ==> ASCITES=2 120    conf:(0.91)
 6. ANTIVIRALS=2 131 ==> SEX=1 119    conf:(0.91)
 7. SPLEEN_PALPABLE=2 120 ==> SEX=1 109    conf:(0.91)
```

Fonte: Do Autor.

Observa-se que a ferramenta *Shell Orion* gerou as mesmas regras de associação que o algoritmo Apriori implementado na Weka, com os mesmos percentuais de suporte e confiança. Comparando-se com o *FP-Growth* da Weka verifica-se que as cinco primeiras regras da *Shell Orion* são idênticas as cinco regras geradas na Weka. As regras seis e sete possuem índices positivos e negativos por isso são descartadas pelo algoritmo *FP-Growth* na Weka, mas não são descartadas pelo algoritmo Apriori na mesma ferramenta.

Verificado o correto funcionamento da implementação do algoritmo *FP-Growth* na ferramenta *Shell Orion* finalizou-se a etapa de análise dos resultados

obtidos. E com o intuito de se verificar o tempo de processamento do mesmo, realizaram-se testes, executando-o com diferentes parâmetros.

6.3.2 Tempos de Processamento do Algoritmo *FP-Growth*

Na etapa da avaliação do tempo de processamento do algoritmo *FP-Growth* foram analisadas a quantidade de registros presentes na base de dados e a quantidade de atributos de entrada selecionados. Também comparou-se os tempos de processamento da *Shell Orion* com os da ferramenta Weka na sua versão 3.6.10.

Para obter-se as bases de dados com diversos tamanhos a mesma foi replicada gradativamente. Adotou-se o padrão de crescimento geométrico[15] para a geração das diferentes cargas de dados a serem testadas. Definiu-se o padrão de crescimento utilizando-se a fórmula(8):

$$C_K = N * 2^{K-1} \quad (8)$$

Onde:

a) C_K: é o tamanho, em quantidade de transações, da base de dados a ser analisada;
b) *N*: número da quantidade de transações da base de dados original;
c) *K*: estabelece o número de iterações.

Para este teste utilizou-se a base de dados da área de ciência política que possui 435 registros. A tabela 20 mostra o resultado do crescimento geométrico a partir da base de dados original. Replicou-se a base de dados original 8 vezes (9 iterações) até atingir o número de 111360 transações, valor considerado suficiente para a análise dos tempos de processamento.

[15] Uma sequência de termos onde existe um termo inicial *A*, e cada termo subsequente é obtido pelo produto do anterior por um valor constante *R* chamada de razão é denominada progressão geométrica ou sequência geométrica (GERSTING, 1995).

Tabela 20. Definição dos tamanhos das cargas de dados.

Iteração	Quantidade de transações	Quantidade de atributos
1	435	17
2	870	17
3	1740	17
4	3480	17
5	6960	17
6	13920	17
7	27840	17
8	55680	17
9	111360	17

Fonte: Do Autor.

Devido a ter-se considerado somente a quantidade de transações (número de registros), os parâmetros de entrada não foram alterados, pois nesses testes não foram considerados os resultados obtidos, considerando-se apenas os tempos de processamento. Por meio da Tabela 21 é possível verificar os tempos de processamento obtidos pelo algoritmo *FP-Growth* na *Shell Orion* conforme o tamanho da base de dados.

Tabela 21. Impacto do número de transações na execução do algoritmo *FP-Growth*.

Quantidade de transações	*Shell Orion*
435	00m: 00s: 015ms
870	00m: 00s: 031ms
1740	00m: 00s: 061ms
3480	00m: 00s: 093ms
6960	00m: 00s: 171ms
13920	00m: 00s: 343ms
27840	00m: 00s: 701ms
55680	00m: 01s: 604ms
111360	00m: 03s: 634ms

Fonte: Do Autor.

O custo computacional também é afetado pela quantidade de atributos selecionados (BOTELHO, 2011). Na Tabela 22 são apresentados os tempos de processamento do algoritmo *FP-Growth* com relação ao efeito da dimensionalidade. Para este teste utilizou-se o modelo de crescimento geométrico com 9 iterações, totalizando 111360 transações.

Tabela 22. Impacto do número de transações na execução do algoritmo *FP-Growth*.

Quantidade de atributos	*Shell Orion*
2	00m: 00s: 405ms
3	00m: 00s: 593ms
5	00m: 00s: 842ms
8	00m: 01s: 524ms
12	00m: 02s: 745ms
17	00m: 03s: 634ms

Fonte: Do Autor.

Verificou-se que a dimensionalidade exerce forte influencia no tempo de processamento do algoritmo, portando deve-se selecionar somente os atributos relevantes como entrada para o algoritmo encontrar regras de associação, pois atributos não relevantes além de não trazer conhecimento útil prejudicam de modo considerável o tempo de processamento do algoritmo.

Após a análise dos tempos de processamento do algoritmo *FP-Growth*, fez-se uma comparação entre a *Shell Orion* e a ferramenta Weka 3.6.10 que também possui implementado o algoritmo *FP-Growth*.

6.3.3 Tempos de processamento *Shell Orion versus* Weka 3.6.10

Nesta etapa realizou-se a coleta dos tempos de processamento do algoritmo *FP-Growth* na *Shell Orion* e dos tempos de processamento usados pela ferramenta Weka para executar o mesmo algoritmo nas mesmas condições. Os resultados foram analisados com o auxílio do pacote estatístico *Statistical Package for the Social Sciences*[16] (SPSS). A tabela 23 mostra os tempos de processamento.

Tabela 23. Tempos de processamento da *Shell Orion* e da Weka.

Quantidade de transações	*Shell Orion*	Weka
435	00m: 00s: 015ms	00m: 00s: 008ms
870	00m: 00s: 031ms	00m: 00s: 011ms
1740	00m: 00s: 061ms	00m: 00s: 012ms
3480	00m: 00s: 093ms	00m: 00s: 025ms
6960	00m: 00s: 171ms	00m: 00s: 045ms
13920	00m: 00s: 343ms	00m: 00s: 081ms
27840	00m: 00s: 701ms	00m: 00s: 163ms
55680	00m: 01s: 604ms	00m: 00s: 335ms
111360	00m: 03s: 634ms	00m: 00s: 675ms

Fonte: Do Autor.

[16] Ferramenta que oferece diversas possibilidades de análise exploratória de dados e de cálculos estatísticos. Pode se obter uma versão para testes em: http://www-01.ibm.com/software/analytics/spss/.

Verifica-se uma diferença entre o tempo de execução da *Shell Orion* e da Weka (tabela 23). A pesar de a ferramenta *Shell Orion* levar um tempo maior para executar o algoritmo *FP-Growth*, ambas as ferramentas aparentam distribuir normalmente os tempos de acordo com o aumento da carga de dados.

Para confirmação desta hipótese, utilizou-se o teste de *Shapiro-Wilk*. Esse teste foi proposto em 1965 por Samuel Shapiro e Martin Wilk, por meio dele calcula-se uma estatística *W* que testa se uma amostra aleatória de tamanho *n* provém de uma distribuição normal. Valores pequenos de *W* dão evidência de desvios da normalidade. O critério para se avaliar o resultado retornado por meio do teste é o valor do *p-value* preestabelecido em 0,05. Portanto a distribuição gaussiana é confirmada se p>0,05. Caso contrário o resultado indica uma distribuição significativamente diferente de uma distribuição normal (FIELD, 2009).

De acordo com o resultado da na tabela 24, isso não se confirmou para ambos os grupos por apresentar valores de significância menor que 0,05 (*p* = 0,002 e *p* = 0,001 respectivamente).

Tabela 24. Teste de normalidade *Shapiro-Wilk* nos tempos da Weka e da *Shell Orion*.

Ferramenta	Estatística	Grau de liberdade	Significância
Shell Orion	0,677	9	0,001
Weka	0,710	9	0,002

Fonte: Do Autor.

Como alternativa utilizou-se a transformação logarítmica dos tempos calculando-se o log natural[17] para cada um em milissegundos (MS). Para o tempo de 15 ms, por exemplo, calculou-se ln(15) que resultou no valor 2,7080502011. Repetiu-se este cálculo para os demais tempos, com isso originou-se uma nova variável com base neperiana, aplicou-se novamente o teste de *Shapiro-Wilk*, desta vez obteve-se a distribuição gaussiana em ambos os grupos *p* = 0,556 para a Weka e *p* = 0,951 para a ferramenta *Shell Orion*, uma vez que o valor do *p* foi maior que 0,05 (tabela 25).

[17] O log natural é o logaritmo de base *e*, onde *e* é um número irracional aproximadamente igual a 2,718281828459045... chamado de número de Euler. O logaritmo natural é uma função que torna possível o estudo de fenômenos que evoluem de maneira exponencial uma vez que ele é a função inversa da função exponencial.

Tabela 25. Teste de normalidade *Shapiro-Wilk* após a transformação logarítmica.

Ferramenta	Estatística	Grau de liberdade	Significância
Shell Orion	0,978	9	0,951
Weka	0,937	9	0,556

Fonte: Do Autor.

Para comparar a homogeneidade das variâncias observadas em cada grupo (*Shell Orion* / Weka) utilizou-se o teste *F* de *Levene*. Esse teste, proposto por Levene em 1960, verifica a igualdade das variâncias, mais precisamente se a diferença entre elas é zero (BARBETTA; REIS; BORNIA, 2010).

O resultado comprovou que as variâncias são homogêneas, aceitando a hipótese nula (p = 0,673), isso acarretou na utilização do teste *t* de *Student* para amostras independentes objetivando a comprovação das médias dos tempos gastos para execução na *Shell Orion* e Weka (tabela 26). Esse teste é utilizado para se avaliar as diferenças entre as médias de dois grupos, se ocorre uma diferença significativa ou uma diferença ao acaso (FIELD, 2009).

Tabela 26. Teste t de *Student* na análise dos tempos da *Shell Orio*n e Weka.

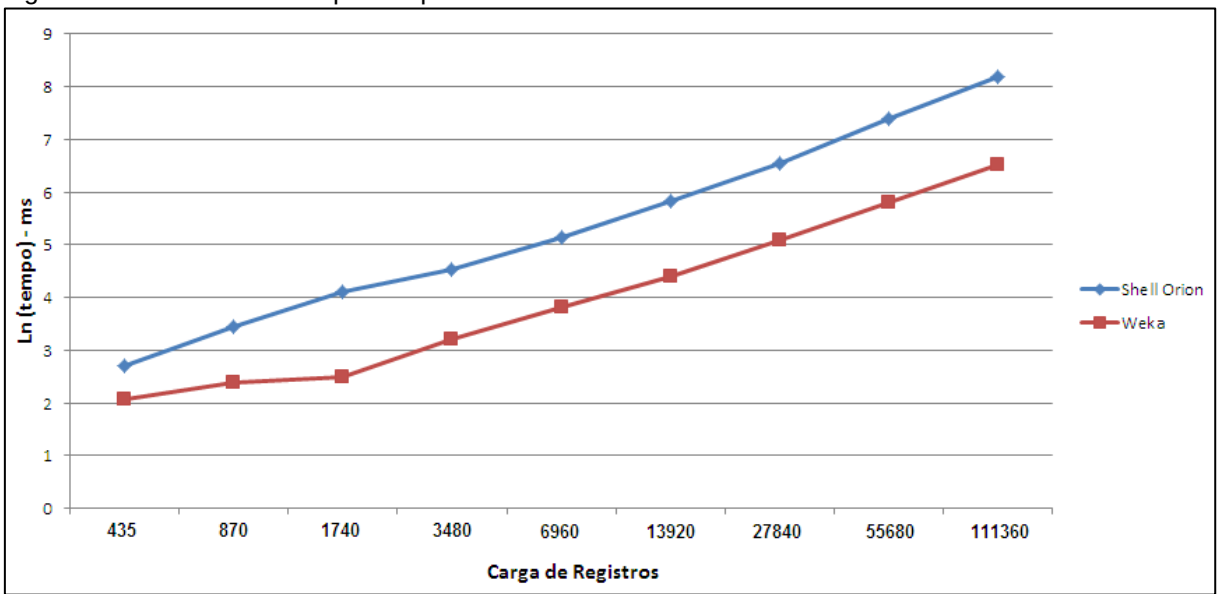
Ferramenta	N	Média	Desvio Padrão	Erro Padrão da Média	Significância
Shell Orion	9	5,321750854556	1,8305825586139	0,6101941862046	0,116
Weka	9	3,978313817454	1,5874831930347	0,5291610643449	

Fonte: Do Autor.

O valor-*p* 0,116 é maior que 0,05, isso comprova que a diferença entre o tempo gasto pela *Shell Orion* e o tempo gasto pela Weka não foi significativo, aceitando a hipótese nula, conclui-se, por meio deste teste estatístico, que a variação ocorre ao acaso.

A diferença nos tempos de processamento após a transformação logarítmica dos tempos entre a *Shell Orion* e a Weka é ilustrada de forma gráfica na figura 33, onde vemos que a Weka mostrou-se um pouco mais rápida que a Shell Orion.

Figura 33 – Gráfico dos tempos de processamento da *Shell Orion* e da Weka.



Fonte: Do Autor.

Analisando os resultados obtidos por meio da implementação do algoritmo *FP-Growth* na *Shell Orion Data Mining Engine*, verifica-se que ele gerou corretamente as regras de associação, pois apresentou resultados satisfatórios. Com relação aos tempos de processamento a Weka apresentou tempos um pouco melhores, porém, do ponto de vista estatístico, a variação apresentada não foi significativa. Portanto o algoritmo *FP-Growth* foi corretamente implementado no módulo de associação da *Shell Orion*.

7 CONCLUSÃO

Constantes avanços no armazenamento de dados, no conhecimento sobre as mais diversas áreas, e na disponibilidade de armazenamento com custo mais acessível, proporcionaram a coleta de grandes quantidades de dados. De fato a capacidade de reunir e armazenar os conjuntos de dados é superior a de analisar e entender os mesmos. Nesse contexto o *data mining* possui fundamental importância, pois disponibiliza técnicas e ferramentas computacionais necessárias para proporcionar agilidade na extração de novos conhecimentos e na confirmação dos já existentes, contribuindo nas tomadas de decisões.

O *data mining* é composto por diversas tarefas, sendo que nesta pesquisa acadêmica aprofundou-se no entendimento da geração de regras de associação e em sua aplicação por meio do algoritmo *FP-Growth*. Este algoritmo disponibiliza técnicas e métodos que geram o conjunto completo de regras sem para isso gerar candidato, isso apresenta grande potencial nesta tarefa uma vez que proporciona um melhor aproveitamento dos recursos computacionais disponíveis e resolve de modo satisfatório o problema de encontrar regras de associação relevantes e úteis.

Durante a realização da pesquisa encontrou-se dificuldades, dentre as quais se destacam a escolha de bases de dados adequadas para a aplicação da tarefa de associação por meio do algoritmo implementado; a seleção de quais medidas de qualidade devem ser implementadas para avaliação das regras geradas; a compreensão dos testes estatísticos utilizados na avaliação do tempo de processamento do algoritmo implementado na *Shell Orion* em comparação com a Weka; e os detalhes referentes ao fluxo de funcionamento do *FP-Growth*, superou-se essa dificuldade por meio da modelagem matemática do algoritmo.

Apesar das dificuldades encontradas no decorrer da pesquisa, conforme demonstraram os resultados obtidos por meio dos testes realizados, o algoritmo *FP-Growth* foi implementado com sucesso, apresentando o correto funcionamento, uma vez que se atingiram os objetivos propostos para a pesquisa.

Na comparação realizada entre as ferramentas *Shell Orion* e Weka, observou-se que os resultados obtidos foram satisfatórios tanto em relação ao tempo de processamento para a execução do algoritmo, quanto em relação as regras de associação geradas.

Por fim, são apresentadas algumas sugestões de trabalhos futuros, tendo por base o conhecimento adquirido por meio da pesquisa, objetivando dar continuidade ao desenvolvimento do projeto da *Shell Orion Data Mining Engine*:

a) pesquisar as variações do algoritmo *FP-Growth* a fim de comparar as vantagens e desvantagens de cada um, implementar as melhorias mais importantes disponíveis na literatura acadêmica e comparar os resultados;
b) implementar técnicas eficientes que exploram diretamente regras de associação ótimas;
c) estudar e implementar outras medidas de qualidade para as regras de associação geradas pelo algoritmo *FP-Growth*;
d) desenvolver um estudo de caso comparando os algoritmos Apriori e *FP-Growth* implementados na *Shell Orion*;
e) disponibilizar no módulo de associação da *Shell Orion* técnicas que permitam aplicar a etapa de *data mining* em tabelas relacionadas;
f) implementar opções no algoritmo *FP-Growth* que permita a exploração de conjuntos de itens que surgem com menor frequência, denominados de regras de associação raras ou infrequentes.

REFERÊNCIAS

AGARWAL, Ramesh C.; AGGARWAL, Charu C.; PRASAD, V.V.V. **A tree projection algorithm for generation of frequent itemsets.** Journal of Parallel and Distributed Computing, 2001, p. 350-371.

AGARWAL, Ramesh C.; AGGARWAL, Charu C.; PRASAD, V.V.V. **Depth Fist Generation of Long Patterns.** Proceedings of the ACM SIGKDD Conference, 2000.

AGRAWAL, Rakesh; IMIELINNSKI, Tomasz; SWAMI, Arun. **Mining association rules between sets of items in large databases.** Washington, DC: Int. Conf. Management of Data (SIGMOD'93), 1993, p. 207–216.

AGRAWAL, Rakesh et al. **Fast discovery of association rules.** In Advances in Knowledge Discovery and Data Mining, AAAI/MIT Press, 1996, p. 307–328.

AGRAWAL, Rakesh; SRIKANT, Ramakrishnan. **Fast algorithms for mining association rules.** Santiago, Chile: Int. Conf. Very Large Data Bases (VLDB'94), 1994, p. 487-499.

AGRAWAL, Rakesh; SRIKANT, Ramakrishnan. **Mining sequential patterns.** Taipei, Taiwan: Int. Conf. Data Engineering (ICDE'95), 1995, p. 3-14.

ALGHAMDI, Abdullah Saad Almalaise. **Efficient Implementation of FP Growth Algorithm-Data Mining on Medical Data**. International Journal of Computer Science and Network Security, 2011. Disponível em <http://paper.ijcsns.org/07_book/html/201112/201112002.html> Acesso em: 2013-05-25.

ALVES, Alexandre Soares**. Regras de Associação e Classificação em Ambiente de Computação Paralela Aplicadas a Sistemas Militares.** 2007. 149 f. Tese (Doutorado em Ciências em Engenharia Civil) – Coordenação dos Programas de Pós-Graduação de Engenharia, Universidade Federal do Rio de Janeiro. Rio de Janeiro, 2007. Disponível em: <http://www.coc.ufrj.br/index.php/component/docman/cat_view/28-doutorado/63-2007?Itemid=> Acesso em: 2013-04-09.

ARNOLD, Ken. **A linguagem de programação java**. 4. ed São Paulo: Bookman, 2007.

ASSUNÇÃO, Alinson Sousa de. **DESCOBERTA DIRETA E EFICIENTE DE REGRAS DE ASSOCIAÇÃO ÓTIMAS.** 2012. 178 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC-USP). São Carlos, 2012. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-29032012-090714/pt-br.php> Acesso em: 2013-06-16.

BARBETTA, Pedro Alberto; REIS, Marcelo Menezes; BORNIA, Antonio Cezar. **Estatística**: para cursos de engenharia e informática. 3. ed. São Paulo: Atlas, 2010.

BATISTA, Gustavo Enrique de Almeida Prado Alves. **Pré-processamento de dados em aprendizado de máquina supervisionado.** 2003. 231f. Tese (Doutorado em Ciências da Computação e Matemática Computacional)- Instituto de Ciências Matemáticas e de Computação. Universidade de São Paulo, São Carlos, 2003. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-06102003-160219/> Acesso em: 2013-03-28.

BERRY,Michael J.A.;LINOFF,Gordon. **Data mining techniques:** for marketing, sales, and customer relationship management. 2. ed. Indianapolis: Wiley Publishing, Inc., 2004.

BISQUERRA ALZINA, Rafael; CASTELLÃ SARRIERA, Jorge; MARTÍNEZ, Francesc. **Introdução à estatística**: enfoque informático com o pacote estatístico SPSS. Porto Alegre: Artmed, 2004.

BORGELT, Christian. **An Implementation of the FP-growth Algorithm**. Workshop Open Source Data Mining Software*,* Chicago, IL, USA: ACM Press, 2005. p.1-5.

BORTOLOTTO, Leandro Sehnem. **O Método de Redes Neurais pelo Algoritmo Kohonen para Clusterização na Shell Orion Data Mining Engine**. Trabalho de Conclusão de Curso - Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciuma, Santa Catarina, 2007.

BOTELHO, Glenda Michele. **Seleção de características apoiada por mineração visual de dados**. 2011. 84f. Dissertação (Mestrado em Ciências da Computação e Matemática Computacional)- Instituto de Ciências Matemáticas e de Computação. Universidade de São Paulo, São Carlos, 2011. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-29032011-145542/> Acesso em: 2013-09-23.

BRAGA, Antônio de Pádua; DE CARVALHO, André Carlos Ponce de Leon Ferreira;

CASAGRANDE, Diego Paz. **O Módulo da Técnica de Associação pelo Algoritmo Apriori no desenvolvimento da Shell de Data Mining Orion**. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, Santa Catarina, 2005.

BRIN, S et al. Dynamic Itemset **Counting and Implication Rules for Market Basket Data.** Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, Arizona, Estados Unidos, 1997, p. 255–264.

CASSETTARI JUNIOR, José Márcio. **O Método de Lógica Fuzzy pelo Algoritmo Gustafson-Kessel na Tarefa de Clusterização da Shell Orion Data Mining Engine.** Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, Santa Catarina. 2008.

CROTTI JUNIOR, Ademar. **O Método de Lógica Fuzzy pelos algoritmos Robust C-Prototypes e Unsupervised Robust C-Prototypes para a Tarefa de Clusterização na Shell Orion Data Mining Engine Engine.** Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, Santa Catarina. 2010.

DEITEL, H.M; DEITEL, P. J. **Java**: como programar. 4.ed Porto Alegre: Bookman, 2003.

DEVORE, Jay L. **Probabilidade e estatística**: para engenharia e ciências. São Paulo: Thomson, 2006.

FAYYAD, Usama M.; PIATETSKY-SHAPIRO, Gregory; SMYTH, Padhraic. **From Data Mining to Knowledge Discovery**: An overview. In: FAYYAD et al. Advances in Knowledge Discovery and Data Mining. G. Cambridge-Mass:AAAI/MIT Press, 1996a.

FAYYAD, Usama; PIATETSKY-SHAPIRO, Gegory; SMYTH, Padhraic. **From Data mining to Knowledge Discovery in Databases**. AI Maganize, v. 17, n. 3, p. 37-54, 1996b.

FAYYAD, Usama; PIATETSKY-SHAPIRO, Gegory; SMYTH, Padhraic. **The KDD Process for Extracting Useful Knowledge from Volumes of Data**. Communications of the ACM, v. 39, p. 27-34, 1996c.

FIELD, Andy. **Descobrindo a estatística usando o SPSS.** 2. ed. Porto Alegre: Artmed, 2009.

FURLAN, José Davi. **Modelagem de objetos através da UML**. São Paulo: Makron Books: 1998.

GAVA, Everton Marangoni. **O método de densidade pelo algoritmo Density-Based Spatial Clustering of Applications With Noise (DBSCAN) na tarefa de clusterização da shell orion data mining engine**. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, Santa Catarina, 2011.

GERSTING, Judith L. **Fundamentos matemáticos para a ciência da computação.** 3. ed. Rio de Janeiro: LTC, 1995.

GOLDSCHMIDT, Ronaldo; PASSOS, Emmanuel Lopes. **Data mining**: uma guia prático: conceitos, técnicas, ferramentas, orientações e aplicações. Rio de Janeiro: Elsevier, 2005.

GONÇALVES, Eduardo Corrêa. **Regras de Associação e suas Medidas de Interesse Objetivas e Subjetivas.** INFOCOMP Journal of Computer Science, 4(1), 2005, p. 26-35.

GUEDES, Gilleanes T. A.. **UML**: uma abordagem prática. 3. ed São Paulo: Novatec, 2008.

GUILLET, Fabrice; HAMILTON, Howard J. **Quality Measures in Data Mining.** Studies in Computational Intelligence 43, Springer, 2007.

HAN, Jiawei; KAMBER, Micheline. **Data mining**: concepts and techniques. 2. ed. San Francisco: Morgan Kaufmann, 2006.

HAN, Jiawei; PEI, Jian; YIN, Yiwen. **Mining Freuqent Patterns Without Candidate Generation.** Dallas, TX: Inf. Conf. on Management of Data (SIGMOD'00), Maio 2000, p. 1-12.

HAN, Jiawei et al. **Mining Freuqent Patterns Without Candidate Generation:** A Frequent-Pattern Tree Approach. Data Mining and Knowledge Discovery, v. 8, 2004, p. 53-87.

HAND, David; MANNILA, Heikki; SMYTH, Padhraic. **Principles of Data Mining.** Cambridge: Mit Press, 2001.

HUBBARD, J. R. **Teoria e problemas da programação com Java**. 2. ed Porto Alegre: Bookman, 2006.

KOUSALYA, R.; SUGUNA, K.; SARAVANAN, V. **Improving the Efficiency of Web Usage Mining Using K-Apriori and FP-Growth Algorithm**. International Journal of Scientific & Engineering Research Volume 4, 2013. Disponível em <http://www.ijser.org/researchpaper%5CImproving-the-Efficiency-of-Web-Usage-Mining-Using-K-Apriori-and-FP-Growth-Algorithm.pdf> Acesso em: 2013-05-17.

LARMAN, Graig. **Utilizando UML e padrões:** uma introdução à análise e ao projeto orientados a objetos. Porto Alegre: Bookman, 2000.

LAROSE, Daniel T.. **Discovering knowledge in data:** an introduction to data mining. Hoboken, New Jersey: John Wiley & Sons, Inc., 2005.

LUDERMIR, Teresa Bernarda. **Redes neurais artificiais**: teoria e aplicações. Rio de Janeiro: LTC, 2000.

LUGER, George F. **Inteligência artificial**: estruturas e estratégias para a resolução de problemas complexos. 4. ed Porto Alegre: Bookman, 2004.

MARTINS, Denis Piazza. **O Algoritmo de Particionamento K-means na Tarefa de Clusterização da Shell Orion Data Mining Engine**. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, Santa Catarina, 2007.

MELANDA, Edson Augusto**. Pós-processamento de Regras de Associação.** 2004. 225 f. Tese (Doutorado em Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade do Estado de São Paulo. São Carlos, 2004. Disponível em: < http://www.teses.usp.br/teses/disponiveis/55/55134/tde-13012006-171753/pt-br.php> Acesso em: 2013-04-08.

MONDARDO,Ricardo Lineburger.**O algoritmo C4.5 na tarefa de classificação da Shell Orion Data Mining Engine**. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, Santa Catarina. 2009.

MOTTA, Custódio Gouvêa Lopes da. **Metodologia para Mineração de Regras de Associação Multiníveis Incluindo Pré e Pós-Processamento.** 2010. 103f. Tese (Doutorado em Ciências em Engenharia Civil) – Coordenação dos Programas de Pós-Graduação de Engenharia, Universidade Federal do Rio de Janeiro. Rio de Janeiro, 2010. Disponível em: <http://fenix3.ufrj.br/60/teses/coppe_d/CustodioGouveaLopesDaMotta.pdf> Acesso em: 2013-04-11.

PAL, Sankar K.; MITRA, Pabitra. **Pattern recognition algorithms for data mining**: scalability, knowledge discovery and soft granular computing. Florida: Chapman & Hall, 2004.

PARK, Jong Soo; Chen Ming-Syan; YU, Philip S. **An Effective Hash-Based Algorithm for Mining Association Rules**. ACM SIGMOD Record archive, 1995.

PIATETSKY-SHAPIRO, G.; STEINGOLD, S. **Measuring Lift Quality in Database Marketing. SIGKDD Explorations** 2, 2000, p. 76–80.

PIATETSKY-SHAPIRO, G. **Discovery, Analysis and Presentation of Strong Rules, Knowledge Discovery in Databases.** AAAI/MIT Press, 1991, p. 229–248.

PELEGRIN, Diana Colombo. **A Tarefa de Classificação e o Algoritmo ID3 para Indução de Árvores de Decisão na Shell de Data Mining Orion**. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, Santa Catarina, 2005.

PEREGO, Daniel. **O Método de Lógica Fuzzy pelo algoritmo GATH-GEVA na tarefa de clusterização da Shell Orion Data Mining Engine**. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, Santa Catarina, 2009.

PRESSMAN, Roger S. **Engenharia de Software**. São Paulo: Makron Books, 1995.

PYLE, Dorian. **Data preparation for data mining**. California: Morgan Kaufmann Publishers, 1999.

RAIMUNDO, Lidiane Rosso. **O Algoritmo CART na Tarefa de Classificação da Shell Orion Data Mining Engine**. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, Santa Catarina, 2007.

REZENDE, Solange Oliveira. **Sistemas inteligentes:** fundamentos e aplicações. Barueri, SP: Manole, 2005.

RUSSELL, Stuart J.; NORVIG, Peter. **Inteligência artificial**. Rio de Janeiro: Elsevier, 2004.

SAVARESE, A.; OMIECINSKI, E.; NAVATHE, S. **An Efficient Algorithm for Mining Association Rules in Large Databases.** In: 21st Conference on Very Large Databases (vldb'95), 1995, Anais. 1995.

SASSI, Renato José**. Uma arquitetura híbrida para descoberta de conhecimento em bases de dados**: teoria dos rough sets e redes neurais artificiais mapas auto organizáveis. 2006. 169 f. Tese (Doutorado em Sistemas Eletrônicos) – Escola Politécnica, Universidade de São Paulo. São Paulo, 2006. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3142/tde-16032007-163930/> Acesso em: 2013-03-18.

SCHONHORST, Gustavo Bonnard**. Mineração de Regras de Associação Aplicada à Modelagem dos Dados Transacionais de um Supermercado.** 2010. 80 f. Dissertação (Mestrado em Engenharia de Produção) – Universidade Federal de Itajubá. Itajubá, 2010. Disponível em: <http://juno.unifei.edu.br/bim/0036319.pdf> Acesso em: 2013-04-12.

SCOTTI, Ana Paula. **O Método de Redes Neurais com Função de Ativação de Base Radial para a Tarefa de Classificação na Shell Orion Data Mining Engine**. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, Santa Catarina. 2010.

SHALABI, Luai Al, SHAABAN, Zyad, KASASBEH, Basel. **Data mining:** a preprocessing engine, Journal of Computer Science, Amman, p. 735-739. Jun. 2006.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. **Sistemas de banco de dados.** Rio de Janeiro: Elsevier, 2006.

SIVANANDAM, S. N.; SUMATHI, S. **Introduction to data mining and its applications.** Berlim: Springer, 2006.

SOMMERVILLE, Ian. **Engenharia de software**. São Paulo: Addison-Wesley, 2003.

SOUSA, Ricardo Miguel Oliveira Pires de. **Extração de Regras de Associação com Itens Raros e Frequentes.** 2009. 131 f. Dissertação (Mestrado em Engenharia Informática) – Instituto Superior de Engenharia do Porto. Porto, 2009. Disponível em: <https://dspace.isep.ipp.pt/jspui/bitstream/123456789/35/1/Master%20Thesis%201071020.pdf> Acesso em: 2013-05-10.

SRIKANT, R.; AGRAWAL, R. **Mining Sequential Patterns: Generalization and Performance Improvements.** EDBT '96. Avignon: França, 1996, p. 3-17.

TAN, Pan-Ning; STEINBACH, Michael; KUMAR, Vipin. **Introdução ao Datamining**: mineração de dados. Rio de Janeiro: Ciência Moderna, 2009.

WITTEN, Ian H.; FRANK, Eibe; HALL, Mark A. **Data mining practical machine learning tools and techniques.** 3. ed. Burlington: Morgan Kaufmann, 2011.

XAVIER, Sandro Luiz da Paixão. **SAMiRA** – Uma Proposta de Sistema de Apoio à Mineração de Regras de Associação. 2010. 97 f. Dissertação (Mestrado em Computação Aplicada) – Instituto Federal de Educação, Ciência e Tecnologia do Ceará. Fortaleza, 2010. Disponível em: < http://www.uece.br/mpcomp/index.php/dissertacoes/47-dissertacao/129-dissertacoes-2010> Acesso em: 2013-05-12.

ZAKI, Mohammed Javeed et al. **New Algorithms for Fast Discovery of Association Rules.** Conf. on Knowledge Discovery and Data Mining, 1997, p. 283-286.

ZAKI, Mohammed Javeed et al. **Parallel Algorithms for Discovery of Association Rules.** Data Mining and Knowledge Discovery, 1997, p. 343-373.

ZAKI, Mohammed Javeed. **Parallel and Distributed Data Mining:** An Introduction. Large-Scale Parallel Data Mining. Berlin: Springer-Verlag, 2000, p. 1-23.

APÊNDICE A – ARTIGO

A Técnica de Associação pelo Algoritmo *Frequent Pattern-Growth (FP-GROWTH)* na *Shell Orion Data Mining Engine*

Júlio César Borba Nandi[1], Merisandra Côrtes de Mattos Garcia[2]

[1]Acadêmico do Curso de Ciência da Computação – Unidade Acadêmica de Ciências, Engenharias e Tecnologias – Universidade do Extremo Sul Catarinense (UNESC) – Criciúma– SC

[2]Professora do Curso de Ciência da Computação – Unidade Acadêmica de Ciências, Engenharias e Tecnologias – Universidade do Extremo Sul Catarinense (UNESC) – Criciúma– SC

julio-bor@hotmail.com.br, mem@unec.ne

resumo. *O crescimento acelerado na coleta de dados nos mais variados campos dos negócios e da área científica tem despertado o interesse dos profissionais a atentar para a necessidade de se desenvolver tecnologias que permitam a análise e exploração de conhecimentos úteis implícitos nesses dados. Neste contexto, o data mining se destaca, pois automatiza o processo da extração de conhecimentos por meio de ferramentas e algoritmos computacionais. Este artigo demonstra a implementação do algoritmo FP-Growth, desenvolvido para superar as limitações presentes no algoritmo Apriori e melhorar a eficiência da descoberta de regras de associação por meio de uma abordagem diferente, sem a geração do conjunto de candidatos.*

Abstract. *The rapid growth in data collection in various fields of business and scientific area has piqued the interest of professionals to attend to the need to develop technologies that enable the analysis and exploitation of useful knowledge implicit in these data. In this context, data mining stands out because it automates the process of extracting knowledge through computational tools and algorithms. This article demonstrates the implementation of FP-Growth algorithm developed to overcome the limitations present in Apriori algorithm and improve the efficiency of discovery of association rules using a different approach without generating the set of candidate.*

1. Introdução

Constantes avanços no armazenamento de dados, no conhecimento sobre as mais diversas áreas, e na disponibilidade de armazenamento com custo mais acessível, proporcionaram a coleta de grandes quantidades de dados. De fato a capacidade de reunir e armazenar os conjuntos de dados é superior a de analisar e entender os mesmos. Portanto a criação de técnicas e ferramentas computacionais é necessária para proporcionar agilidade na extração de conhecimento útil dos volumes de dados em rápida expansão [Tan, Steinbach e Kumar 2009].

Para atender esta necessidade, surgiu a área denominada Descoberta de Conhecimento em Bases de Dados, *Knowledge Discovery in Databases* (KDD), esse processo inclui limpeza, integração, seleção e transformação de dados; data mining; avaliação de padrões e apresentação do conhecimento [Han e Kamber 2006]. Sendo o *data mining* (DM) a mais destacada etapa desse processo, pois disponibiliza técnicas e ferramentas computacionais necessárias para proporcionar agilidade na extração de novos conhecimentos e na

confirmação dos já existentes, contribuindo nas tomadas de decisões [Goldschmidt e Passos 2005].

O Grupo de Pesquisa em Inteligência Computacional Aplicada do Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense mantém em desenvolvimento a *Shell Orion Data Mining Engine*, que consiste em um projeto acadêmico que implementa métodos, técnicas e algoritmos de DM em um software gratuito e acessível a comunidade. Dentre as diversas tarefas existentes no processo de *data mining*, a *Shell Orio*n atualmente implementa as tarefas de classificação, clusterização e associação.

A descoberta de regras de associação tem recebido atenção por parte de pesquisadores tanto da área acadêmica como das organizações. Na área acadêmica, as pesquisas desenvolvidas estão gerando bons resultados. Esses resultados estão sendo utilizados pelas organizações em aplicações práticas [Han e Kamber 2006].

Neste artigo apresenta-se a implementação do algoritmo *FP-Growth*, no modulo de associação da *Shell Orion*.

2. O Algoritmo *FP-Growth*

Algoritmos tradicionais de associação adotam uma abordagem igual ou semelhante a do algoritmo Apriori, que se baseia na seguinte regra: se qualquer padrão de comprimento k não é frequente na base de dados, seu comprimento (k + 1) não será frequente. A ideia é, por meio de um processo iterativo, gerar o conjunto de padrões de candidatos de comprimento (k +1) a partir do conjunto de padrões de frequência de comprimento k (para $k \geq 1$), e verificar suas frequências de ocorrência na base de dados. No entanto, a geração de conjunto candidato é ainda dispendiosa especialmente quando há um grande número de padrões e/ou estes são longos, ou seja, padrões formados por um número expressivo de itens. Também é dispendioso percorrer repedidas vezes a base de dados para verificar e testar todos os conjuntos de candidatos e seus padrões correspondentes [Han et al 2004].

Com o objetivo de superar essas limitações e melhorar a eficiência da descoberta de regras de associação diversos métodos e algoritmos alternativos têm sido desenvolvidos. Como por exemplo o algoritmo *FP-Growth* que utiliza uma abordagem diferente do Apriori, sem a geração do conjunto de candidatos. Também não concorda com o paradigma de gerar e testar do Apriori, ao contrário disso, codifica o conjunto de dados em uma estrutura de dados compacta em forma de árvore chamada *Frequent Pattern tree (FP-tree)* e extrai os conjuntos de itens frequentes diretamente desta estrutura. Isso possibilita uma melhor eficiência na geração das regras de associação, pois evita constantes acessos na base de dados [Tan, Steinbach e Kumar 2009].

3. O Algoritmo *FP-Growth* na Tarefa de Associação da *Shell Orion Data Mining Engine*

A implementação do algoritmo *FP-Growth* teve inicio com a construção dos diagramas de caso de uso, atividades e sequência utilizando o padrão UML. Posteriormente foi desenvolvida a demonstração matemática do funcionamento do algoritmo com a finalidade de facilitar o entendimento do mesmo.

O algoritmo necessita de no mínimo dois parâmetros de entrada:

a) **suporte:** é a métrica utilizada pelo algoritmo para encontrar todos os *N itemsets* frequentes. O suporte de uma regra de associação X, $A \Rightarrow B$, é a porcentagem das transações que contêm $A \cup B$ em relação ao número total de transações analisadas;

b) **confiança:** calcula a força da regra. Assim, sendo C a confiança de uma regra de associação, $A \Rightarrow B$, C é na verdade a porcentagem das transações que contêm $A \cup B$ em relação a todas as transações que contêm A.

Definidos os parâmetros de entrada e a uma base de dados a ser analisada, o primeiro passo do *FP-Growth* consiste em montar a estrutura *FP-tree* que irá armazenar os *1 itemsets* frequentes. Para isso é necessário ler a base de dados duas vezes para que a *FP-tree* seja criada. Durante a primeira leitura é identificado o conjunto de *itemsets* frequentes *F*, de tamanho um, e seus respectivos suportes (tabela 1).

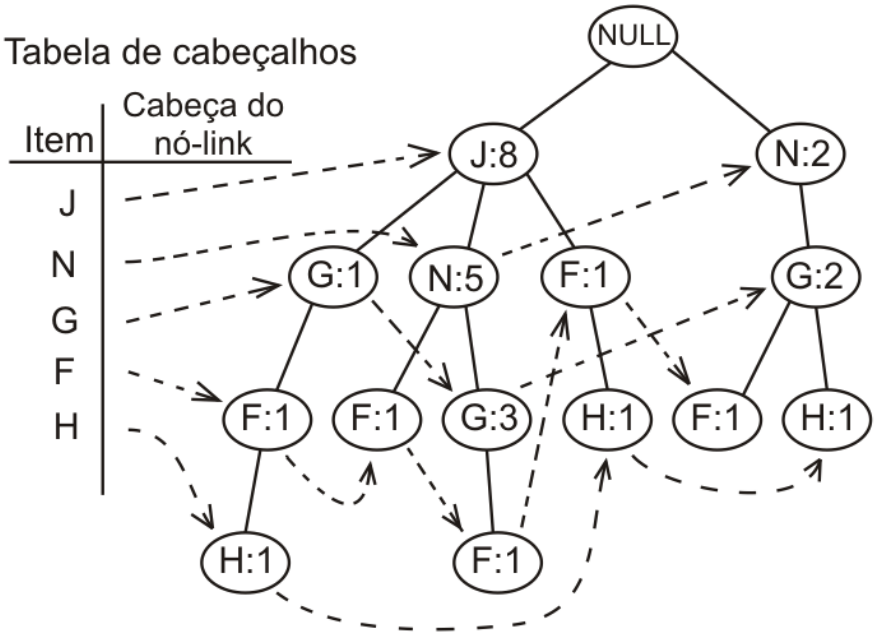
Tabela 1. Primeira leitura da base de dados.

Transações	Itens	1 *Itemsets*
1	N, J, M	N:1, J:1, M:1
2	N, F, G	N:2, J:1, M:1, F:1, G:1
3	J, G, F, H	N:2, J:2, M:1, F:2, G:2, H:1
4	J, F, H	N:2, J:3, M:1, F:3, G:2, H:2
5	J, N, G	N:3, J:4, M:1, F:3, G:3, H:2
6	J, N, G, F	N:4, J:5, M:1, F:4, G:4, H:2
7	J, P	N:4, J:6, M:1, F:4, G:4, H:2, P:1
8	J, N, G	N:5, J:7, M:1, F:4, G:5, H:2, P:1
9	J, N, F	N:6, J:8, M:1, F:5, G:5, H:2, P:1
10	N, G, H, O	N:7, J:8, M:1, F:5, G:6, H:3, P:1, O:1

Logo após a primeira leitura da base de dados os *1 itemsets* infrequentes, que possuem valor de suporte menor que o especificado são removidos, já os frequentes são ordenados em ordem decrescente e armazenados em uma lista *L*.

Começa a segunda leitura da base de dados onde cada transação é lida novamente, os *itemsets* infrequentes de tal transação são desconsiderados e os frequentes são armazenados em uma lista *T*. A lista *T* é ordenada de acordo com o valor de suporte dos *itemsets* armazenados na lista *L*. Em seguida a lista *T* é passada como parâmetro para o método que insere a transação na *FP-tre.* A figura 1 mostra uma *FP-tree* criada considerando-se que o suporte mínimo é igual a duas transações (20%).

Figura 1. Segunda leitura da base de dados.



Na construção da estrutura *FP-tree*, o algoritmo *FP-Growth* cria uma tendência. Isso ocorre devido à forma como as informações, que estão contidas nas transações que estão sendo lidas, são organizadas. Essa tendência, padrão frequente de crescimento, dá nome ao algoritmo; contribui para um melhor aproveitamento dos ramos inseridos na estrutura e a torna compacta; também evita a dispendiosa etapa da geração e teste de candidatos, pois não necessita que um *itemset* descoberto seja armazenado para consultas em iterações posteriores.

Ao término da segunda leitura da base de dados, todas as transações estarão processadas e inseridas na estrutura *FP-tree* que estará corretamente criada. Ela irá representar a base de dados original e será passada como parâmetro para outro método do algoritmo *FP-Growth* responsável pela descoberta dos *N itemsets* frequentes.

O método consiste no desenvolvimento de uma estratégia baseada na técnica de dividir para conquistar, onde o problema é fracionado em subproblemas. O algoritmo faz isso considerando cada item da tabela de cabeçalhos. O primeiro item a ser processado é o último, o segundo o penúltimo item e assim sucessivamente até o término do processamento de todos os itens. Para cada *itemset* presente na tabela de cabeçalhos, bases de padrões condicionais são geradas, as quais são utilizadas para a construção das *FP-tree* condicionais, que relacionam os caminhos frequentes que se conectam aos nós correspondentes ao *itemset* em questão. Uma vez criadas, as *FP-tree* condicionais são utilizadas para encontrar os padrões frequentes que apresentam o *itemset* como sufixo [Han, Pei e Yin 2000].

Tendo como base o conjunto de *N itemsets* frequentes encontrados por meio da *FP-tree*, o algoritmo *FP-Growth* gera as regras de associação. A partir deste momento começa a fase de pós-processamento das regras geradas, na *Shell Orion*, objetivando a seleção das regras úteis e fortes. O algoritmo *FP-Growth* irá primeiro calcular a confiança (fórmula 1):

$$conf(A \Rightarrow B) = prob(A\,|\mathrm{B}) = \frac{\sup(A \cup B)}{\sup(A)} = \frac{\mathrm{n}(A \cup B)}{\mathrm{n}(A)} * 100 \quad (1)$$

Depois irá calcular respectivamente:

a) ***lift*:** é utilizada para avaliar dependências, ou seja, esta medida de qualidade indica o quanto mais frequente torna-se o consequente da regra quando o antecedente ocorre (fórmula 2);

$$Lift(A \Rightarrow B) = \frac{Conf(A \Rightarrow B)}{Sup(B)} \quad (2)$$

b) ***leverage*:** esta medida é utilizada com o objetivo de se descobrir o valor da diferença entre o suporte real e o esperado de uma regra de associação (formula 3);

$$Leverage(A \Rightarrow B) = Sup(A \Rightarrow B) - (Sup(A) \times Sup(B)) \quad (3)$$

c) **convicção:** tem por objetivo avaliar uma regra de associação como uma implicação, ou seja, o sentido da seta "⇒"(fórmula 4).

$$Conv(A \Rightarrow B) = \frac{Sup(A) \times Sup(\neg B)}{Sup(A\ U\ \neg B)} \quad (4)$$

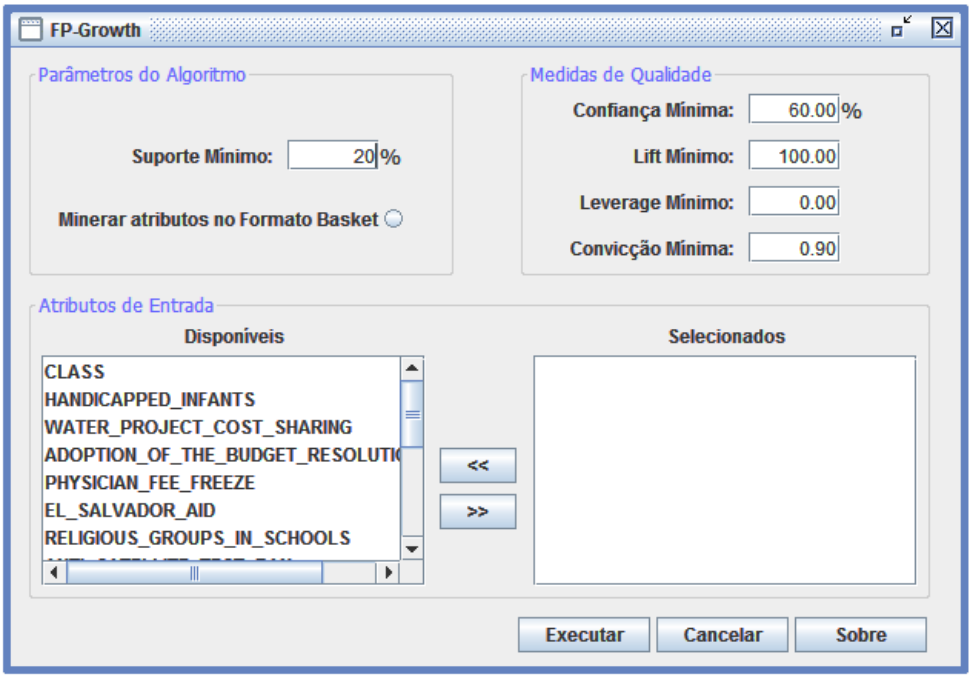
Somente as regras de associação que satisfazerem todas as medidas de qualidade serão consideradas relevantes e úteis.

3.1 Implementação

O algoritmo *FP-Growth* foi implementado no módulo de associação da *Shell Orion Data Mining Engine*, por meio da linguagem de programação Java e do ambiente de programação integrado *NetBeans* 7.3.1.

Para a execução do algoritmo é necessário selecionar os atributos da tabela que serão analisados e informar os valores mínimos para o suporte, a confiança, o *lift*; o *leverage* e a convicção (figura 2).

Figura 2. Seleção dos parâmetros e atributos de entrada.

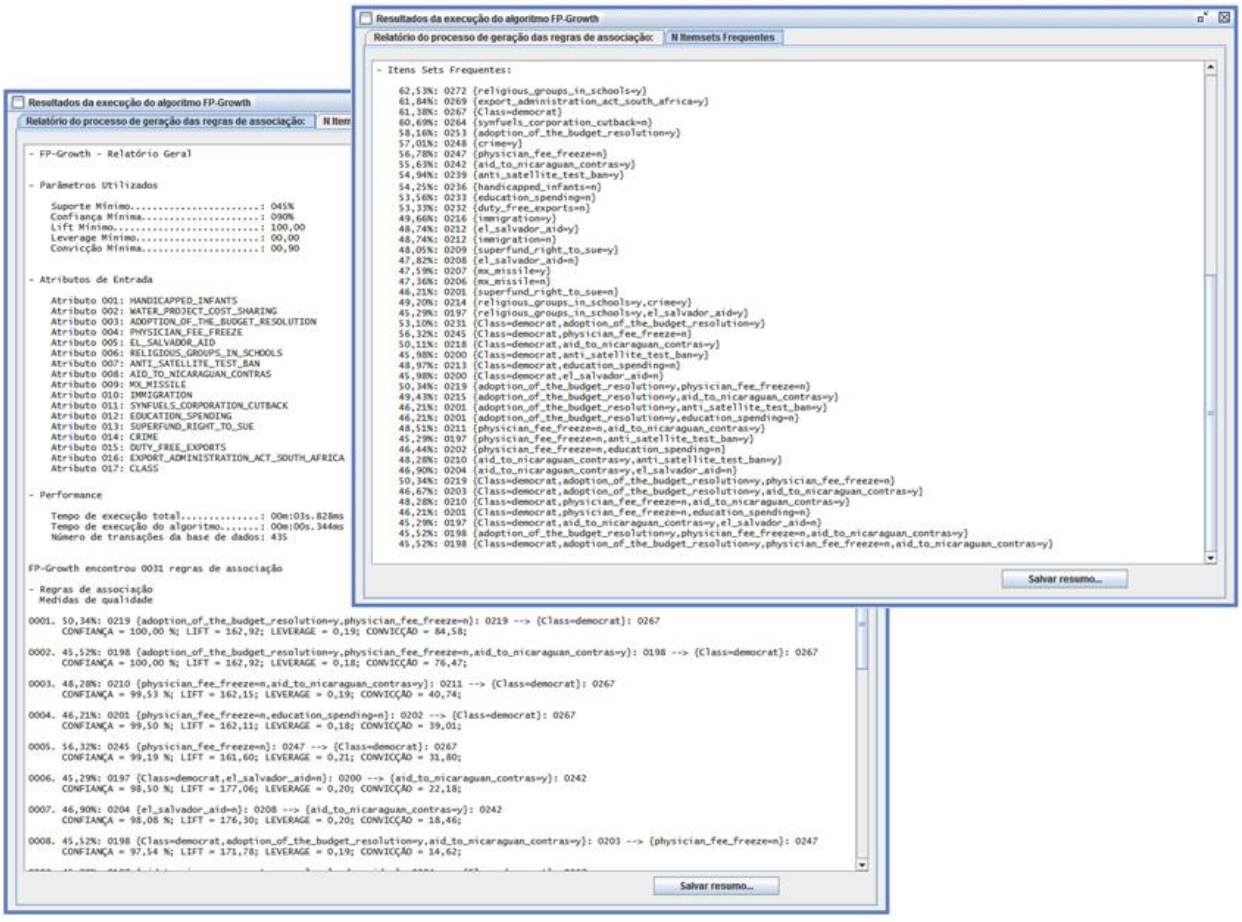


Os resultados obtidos pelo algoritmo *FP-Growth* na *Shell Orion*, podem ser analisados por meio de relatórios. A figura 3 mostra um relatório textual gerado pelo algoritmo contendo os valores informados para os parâmetros de entrada; os atributos de entrada selecionados; os tempos de execução do algoritmo e o tempo total (execução do algoritmo mais montagem do relatório); número de transações presentes na base de dados; as regras de associação geradas

com o percentual e a quantidade de suporte; e os valores das medidas de qualidade. Também é possível exportar os resultados obtidos para um arquivo texto por meio do botão Salvar resumo.

Na figura 3 tem-se outro relatório gerado pela *Shell Orion* contendo informações dos resultados do algoritmo *FP-Growth*. Neste são listados os *N itemsets* frequentes encontrados com o percentual e a quantidade de suporte. Decidiu-se criar um relatório em separado para os *N itemsets* afim de facilitar a análise do usuário e não sobrecarregar o relatório principal que contém as regras de associação.

Figura 3. Segunda leitura da base de dados.



3.2 Resultados Obtidos

A base de dados utilizada na avaliação do algoritmo *FP-Growth* na *Shell Orion* contém dados clínicos de pacientes com hepatite. Esta base de dados possui 155 transações distribuídas em duas classes, a dos pacientes que morreram composta por 32 registros e a outra com os que viveram com 123 casos e outra que identifica o sexo do paciente, 12 atributos são binários, enquanto 6 atributos apresentam valores contínuos, totalizando 20 registros.

Os testes realizados abrangem a análise das regras de associação identificadas, a análise dos tempos de processamento e a comparação dos tempos de processamento entre a *Shell Orion* e a ferramenta Weka 3.6.10.

3.2.1 Regras de Associação Identificadas pelo Algoritmo FP-Growth

Os testes de mesa[18] necessários para se testar a implementação do algoritmo, foram feitos com o exemplo mostrado na modelagem matemática, por isso a primeira análise dos resultados feita nessa fase da pesquisa foi uma comparação com o resultado gerado pelo *FP-Growth* na ferramenta Weka.

Executou-se o algoritmo utilizando-se os seguintes parâmetros e atributos de entrada: suporte 70%, confiança 90%, *lift* 100, *leverage* 0 (zero) e convicção 1.01. Na figura 4 é possível verificar as regras geradas pelo algoritmo *FP-Growth*.

Figura 4. Regras encontradas com os parâmetros informados.

```
FP-Growth encontrou 0007 regras de associação

- Regras de associação
  Medidas de qualidade

0001. 77,42%: 0120 {ASCITES=2}: 0130 --> {VARICES=2}: 0132
      CONFIANÇA = 92,31 %; LIFT = 108,39; LEVERAGE = 0,06; CONVICÇÃO = 1,75;

0002. 72,90%: 0113 {Class=2}: 0123 --> {ASCITES=2}: 0130
      CONFIANÇA = 91,87 %; LIFT = 109,54; LEVERAGE = 0,06; CONVICÇÃO = 1,80;

0003. 70,97%: 0110 {SPLEEN_PALPABLE=2}: 0120 --> {VARICES=2}: 0132
      CONFIANÇA = 91,67 %; LIFT = 107,64; LEVERAGE = 0,05; CONVICÇÃO = 1,62;

0004. 72,26%: 0112 {Class=2}: 0123 --> {VARICES=2}: 0132
      CONFIANÇA = 91,06 %; LIFT = 106,92; LEVERAGE = 0,05; CONVICÇÃO = 1,52;

0005. 77,42%: 0120 {VARICES=2}: 0132 --> {ASCITES=2}: 0130
      CONFIANÇA = 90,91 %; LIFT = 108,39; LEVERAGE = 0,06; CONVICÇÃO = 1,64;

0006. 76,77%: 0119 {ANTIVIRALS=2}: 0131 --> {SEX=1}: 0139
      CONFIANÇA = 90,84 %; LIFT = 101,30; LEVERAGE = 0,01; CONVICÇÃO = 1,04;

0007. 70,32%: 0109 {SPLEEN_PALPABLE=2}: 0120 --> {SEX=1}: 0139
      CONFIANÇA = 90,83 %; LIFT = 101,29; LEVERAGE = 0,01; CONVICÇÃO = 1,03;
```

Estudou-se a ferramenta Weka em fóruns, tutoriais disponíveis na internet e no livro publicado por Witten, Frank e Hall no ano de 2011. Configurou-se a mesma para obedecer os mesmos parâmetros usados na *Shell Orion*. No primeiro teste não gerou nenhuma regra de associação.

Após uma nova série de testes, estudos e exploração na ferramenta Weka, constatou-se que o algoritmo *FP-Growth* implementado na Weka trabalha somente com índices binários, isto é, com atributos que possuem dois valores (SIM = + / NÃO = -). E gera as regras considerando somente os valores positivos ou negativos dos atributos na base de dados. Descobriu-se que por default a ferramenta vem selecionada para gerar regras com os atributos que possuem índice positivo. Alterou-se para gerar com os atributos que possuem índice negativo.

Desta vez a ferramenta Weka gerou cinco regras de associação. Comparando-se estas regras (figura 5) com as regras geradas pela *Shell Orion* verifica-se que as cinco regras geradas na Weka são iguais as cinco primeiras regras geradas na *Shell Orion*. Inclusive os valores para as medidas de qualidade também são os mesmos diferenciando-se somente na forma como ambas mostram o resultado.

[18] Procedimento que simula a execução de um algoritmo utilizando apenas papel e caneta, por meio do teste de mesa é possível verificar as mudanças no conteúdo das variáveis de um algoritmo instrução a instrução, facilitando ao desenvolvedor identificar e corrigir comportamentos falhos (DEITEL; DEITEL, 2003).

Figura 5. Execução da ferramenta Weka para atributos com índice negativo.

Weka

```
=== Associator model (full training set) ===

FPGrowth found 5 rules

1. [ASCITES=2]: 130 ==> [VARICES=2]: 120   <conf:(0.92)> lift:(1.08) lev:(0.06) conv:(1.75)
2. [Class=2]: 123 ==> [ASCITES=2]: 113   <conf:(0.92)> lift:(1.1) lev:(0.06) conv:(1.8)
3. [SPLEEN_PALPABLE=2]: 120 ==> [VARICES=2]: 110   <conf:(0.92)> lift:(1.08) lev:(0.05) conv:(1.62)
4. [Class=2]: 123 ==> [VARICES=2]: 112   <conf:(0.91)> lift:(1.07) lev:(0.05) conv:(1.52)
5. [VARICES=2]: 132 ==> [ASCITES=2]: 120   <conf:(0.91)> lift:(1.08) lev:(0.06) conv:(1.64)
```

Então resolveu-se realizar uma comparação com o resultado do algoritmo Apriori também implementado na Weka. A figura 6 mostra um resumo dos resultados apresentados.

Figura 6. Regras geradas na Shell Orion e na Weka.

Shell Orion

```
FP-Growth encontrou 0007 regras de associação

- Regras de associação
  Medidas de qualidade

0001. 77,42%: 0120 {ASCITES=2}: 0130 --> {VARICES=2}: 0132
      CONFIANÇA = 92,31 %; LIFT = 108,39; LEVERAGE = 0,06; CONVICÇÃO = 1,75;

0002. 72,90%: 0113 {Class=2}: 0123 --> {ASCITES=2}: 0130
      CONFIANÇA = 91,87 %; LIFT = 109,54; LEVERAGE = 0,06; CONVICÇÃO = 1,80;

0003. 70,97%: 0110 {SPLEEN_PALPABLE=2}: 0120 --> {VARICES=2}: 0132
      CONFIANÇA = 91,67 %; LIFT = 107,64; LEVERAGE = 0,05; CONVICÇÃO = 1,62;

0004. 72,26%: 0112 {Class=2}: 0123 --> {VARICES=2}: 0132
      CONFIANÇA = 91,06 %; LIFT = 106,92; LEVERAGE = 0,05; CONVICÇÃO = 1,52;

0005. 77,42%: 0120 {VARICES=2}: 0132 --> {ASCITES=2}: 0130
      CONFIANÇA = 90,91 %; LIFT = 108,39; LEVERAGE = 0,06; CONVICÇÃO = 1,64;

0006. 76,77%: 0119 {ANTIVIRALS=2}: 0131 --> {SEX=1}: 0139
      CONFIANÇA = 90,84 %; LIFT = 101,30; LEVERAGE = 0,01; CONVICÇÃO = 1,04;

0007. 70,32%: 0109 {SPLEEN_PALPABLE=2}: 0120 --> {SEX=1}: 0139
      CONFIANÇA = 90,83 %; LIFT = 101,29; LEVERAGE = 0,01; CONVICÇÃO = 1,03;
```

Weka

```
=== Associator model (full training set) ===

FPGrowth found 5 rules

1. [ASCITES=2]: 130 ==> [VARICES=2]: 120   <conf:(0.92)> lift:(1.08) lev:(0.06) conv:(1.75)
2. [Class=2]: 123 ==> [ASCITES=2]: 113   <conf:(0.92)> lift:(1.1) lev:(0.06) conv:(1.8)
3. [SPLEEN_PALPABLE=2]: 120 ==> [VARICES=2]: 110   <conf:(0.92)> lift:(1.08) lev:(0.05) conv:(1.62)
4. [Class=2]: 123 ==> [VARICES=2]: 112   <conf:(0.91)> lift:(1.07) lev:(0.05) conv:(1.52)
5. [VARICES=2]: 132 ==> [ASCITES=2]: 120   <conf:(0.91)> lift:(1.08) lev:(0.06) conv:(1.64)
```

Weka Apriori

```
Best rules found:

 1. ASCITES=2 130 ==> VARICES=2 120    conf:(0.92)
 2. Class=2 123 ==> ASCITES=2 113    conf:(0.92)
 3. SPLEEN_PALPABLE=2 120 ==> VARICES=2 110    conf:(0.92)
 4. Class=2 123 ==> VARICES=2 112    conf:(0.91)
 5. VARICES=2 132 ==> ASCITES=2 120    conf:(0.91)
 6. ANTIVIRALS=2 131 ==> SEX=1 119    conf:(0.91)
 7. SPLEEN_PALPABLE=2 120 ==> SEX=1 109    conf:(0.91)
```

Observa-se que a *Shell Orion* gerou as mesmas regras de associação que o algoritmo Apriori implementado na Weka, com os mesmos percentuais de suporte e confiança. Comparando-se com o *FP-Growth* da Weka verifica-se que as cinco primeiras regras da Shell Orion são idênticas as cinco regras geradas na Weka. As regras seis e sete possuem índices positivos e negativos por isso são descartadas pelo algoritmo *FP-Growth* na Weka, mas não são descartadas pelo algoritmo Apriori na mesma ferramenta.

3.2.2 Tempos de processamento do algoritmo *FP-Growth*

A base de dados original foi replicada gradativamente para se obter conjuntos de diversos tamanhos, fez-se isso por meio do crescimento geométrico que torna possível a execução do algoritmo em diferentes tamanhos de cargas (centenas a milhares) dentro de um pequeno intervalo de iterações:

$$C_k = N \times 2^{k-1}$$

Na tabela 2 são verificados os tempos de processamento obtidos pelo algoritmo em relação ao tamanho da base de dados.

Tabela 2. Impacto do número de transações.

Quantidade de transações	*Shell Orion*
435	00m: 00s: 015ms
870	00m: 00s: 031ms
1740	00m: 00s: 061ms
3480	00m: 00s: 093ms
6960	00m: 00s: 171ms
13920	00m: 00s: 343ms
27840	00m: 00s: 701ms
55680	00m: 01s: 604ms
111360	00m: 03s: 634ms

3.2.3 Tempos de processamento *Shell Orion* versus Weka 3.6.10

A fim de comparar o desempenho em relação ao tempo de processamento entre as duas ferramentas, realizou-se também a coleta dos tempos gastos pela Weka para executar as mesmas tarefas nas mesmas condições. A tabela 3 mostra o resultado:

Tabela 3. Tempos de processamento da *Shell Orion* e da Weka.

Quantidade de transações	*Shell Orion*	Weka
435	00m: 00s: 015ms	00m: 00s: 008ms
870	00m: 00s: 031ms	00m: 00s: 011ms
1740	00m: 00s: 061ms	00m: 00s: 012ms
3480	00m: 00s: 093ms	00m: 00s: 025ms
6960	00m: 00s: 171ms	00m: 00s: 045ms
13920	00m: 00s: 343ms	00m: 00s: 081ms
27840	00m: 00s: 701ms	00m: 00s: 163ms
55680	00m: 01s: 604ms	00m: 00s: 335ms
111360	00m: 03s: 634ms	00m: 00s: 675ms

Para as análises dos tempos coletados utilizou-se o pacote estatístico *Statistical Package for the Social Sciences* (SPSS). Aplicou-se o teste de normalidade *Shapiro-Wilk*, objetivando verificar se a distribuição dos valores da amostra era normal. O teste resultou em um *p-value* menor que o nível de significância preestabelecido (0,05) constatando que a distribuição não foi normal.

Optou-se pela transformação logarítmica dos tempos. Após o cálculo do *log* natural dos tempos, aplicou-se mais uma vez o teste *Shapiro-Wilk* e desta vez obteve-se uma distribuição gaussiana em ambos os grupos.

Para comparar a homogeneidade das variâncias observadas, utilizou-se o teste *F* de *Levene*, que aceitou a hipótese nula (p = 0,673), o resultado comprovou que as variâncias são homogêneas, isso acarretou na utilização do teste *t* de *Student* para amostras independentes objetivando a comprovação das médias dos tempos gastos para execução na *Shell Orion* e Weka.

A realização do teste *t* de *Student* revelou que a diferença do tempo de processamento entre a *Shell Orion* e a Weka não foi significativa (p = 0,116), aceitando a hipótese nula (H0), portanto, conclui-se que a variação ocorre ao acaso.

4. Considerações Finais

Este artigo apresentou o algoritmo *FP-Growth*, que utiliza uma abordagem diferente para resolver o problema da descoberta de regras de associação, sem para isso gerar conjunto de candidatos, implementado na tarefa de associação da *Shell Orion Data Mining Engine*, contribuindo com a ampliação das suas funcionalidades.

Analisando os resultados obtidos, verifica-se que a *Shell Orion* gerou corretamente as regras de associação, pois apresentou resultados satisfatórios. Com relação aos tempos de processamento a Weka apresentou tempos um pouco melhores, porém, do ponto de vista estatístico, a variação apresentada não foi significativa. Portanto o algoritmo *FP-Growth* foi corretamente implementado no módulo de associação da *Shell Orion*.

Referências

AGRAWAL, Rakesh et al. **Fast discovery of association rules**. In Advances in Knowledge Discovery and Data Mining, AAAI/MIT Press, 1996, p. 307–328.

AGRAWAL, Rakesh; SRIKANT, Ramakrishnan. **Fast algorithms for mining association rules**. Santiago, Chile: Int. Conf. Very Large Data Bases (VLDB'94), 1994, p. 487-499.

GOLDSCHMIDT, Ronaldo; PASSOS, Emmanuel Lopes. **Data mining:** uma guia prático. Rio de Janeiro: Elsevier, 2005.

HAN, Jiawei; PEI, Jian; YIN, Yiwen. **Mining Freuqent Patterns Without Candidate Generation**. Dallas, TX: Inf. Conf. on Management of Data (SIGMOD'00), Maio 2000, p. 1-12.

HAN, Jiawei; KAMBER, Micheline. **Data mining**: concepts and techniques. San Francisco: Morgan Kaufmann, 2006.

TAN, Pan-Ning; STEINBACH, Michael; KUMAR, Vipin. **Introdução ao Datamining**: mineração de dados. Rio de Janeiro: Ciência Moderna, 2009.

WITTEN, Ian H.; FRANK, Eibe; HALL, Mark A. **Data mining practical machine learning tools and techniques.** 3. ed. Burlington: Morgan Kaufmann, 2011.