DISSERTAÇÃO DE MESTRADO

Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Mineração de Dados Em Múltiplas
Tabelas Fato de um *Data Warehouse*

Orientadora: Profa. Dra. Marina Teresa Pires Vieira
Aluna: Marcela Xavier Ribeiro

São Carlos
Abril/2004

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

R484md

Ribeiro, Marcela Xavier.
Mineração de dados em múltiplas tabelas fato de um *data warehouse* / Marcela Xavier Ribeiro. -- São Carlos : UFSCar, 2004.
121 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2004.

1. Banco de Dados. 2. *Data mining* (mineração de dados). 3. Descoberta de conhecimento em *data warehouses* - KDD. I. Título.

CDD: 005.75 (20ª)

Aos meus pais,
pelo exemplo de amor,
trabalho e dedicação.

Agradecimentos

A Deus, aquele que esteve sempre ao meu lado;
À minha orientadora Marina, não só pela valiosa orientação, mas também pela amizade e compreensão;
Ao Sérgio, pelo companheirismo, carinho e dedicação;
Aos meus familiares: Pedro, Isabel, Sale, Rê, Jorge, Nestor e Élide, pela infinita bondade de vocês;
Aos amigos do PPG-CC, especialmente ao Grupo de Banco de Dados, pelo convívio alegre, pelos momentos de descontração e pela troca de conhecimento;
A CAPES pelo apoio financeiro;
E a todos que, de alguma forma, contribuíram para essa minha vitória,

Meu muitíssimo obrigada!

Resumo

O progresso da tecnologia de informação permitiu que quantidades cada vez maiores de dados fossem armazenadas. Esses dados, no formato original de armazenamento, não trazem conhecimento, porém, quando tratados e passados por um processo de extração de conhecimento, podem revelar conhecimentos interessantes.

Os *data warehouses* são repositórios de dados com alta qualidade. Um procedimento que vem sendo amplamente adotado nas grandes empresas é a utilização conjunta das tecnologias de *data warehouse* e da mineração de dados, para que o processo de descoberta de conhecimento pela alta qualidade dos dados do *data warehouse*. *Data warehouses* que envolvem mais de um assunto também envolvem mais de uma tabela fato (tabelas que representam o assunto do *data warehouse*). A análise em conjunto de múltiplos assuntos de um *data warehouse* pode revelar padrões interessantes como o relacionamento entre as compras e as vendas de determinada organização.

Este projeto de pesquisa está direcionado ao desenvolvimento de técnicas de mineração de dados envolvendo múltiplas tabelas fato de um *data warehouse*, que é um novo tipo de mineração de regras de associação.

Abstract

The progress of the information technology has allowed huge amount of data to be stored. Those data, when submitted to a process of knowledge discovery, can bring interesting results. Data warehouses are repositories of high quality data. A procedure that has been adopted in big companies is the joint use of data warehouse and data mining technologies, where the process of knowledge discovery takes advantage over the high quality of the warehouse's data. When the data warehouse has information about more than one subject, it also has more than one fact table. The joint analysis of multiple fact tables can bring interesting knowledge as, for instance, the relationship between purchases and sales in a company.

This research presents a technique to mine data from multiple fact tables of a data warehouse, which is a new kind of association rule mining.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

1.1 Motivação

O progresso da tecnologia de informação permitiu que quantidades cada vez maiores de dados fossem armazenadas. Esses dados, no formato original de armazenamento não trazem conhecimento, porém, quando tratados e passados por um processo de mineração, podem revelar conhecimentos interessantes, que são úteis para vários tipos de aplicações como, por exemplo, a determinação de padrões de tempestades em determinadas regiões do globo terrestre.

Uma tendência no processo de Descoberta de Conhecimento em Bases de Dados, também conhecido como KDD (*Knowledge Discovery in Databases*), é o uso conjunto das tecnologias de mineração de dados (*data mining*) e *data warehouse*, de maneira que o processo de mineração seja beneficiado pelo pré-processamento que os dados passaram ao popular o *data warehouse*.

Data warehouses são repositórios de dados históricos que possuem uma tabela fato e várias tabelas dimensão para armazenar dados de um determinado assunto. Assim, se o *data warehouse* possuir dados sobre mais de um assunto, então ele terá mais de uma tabela fato. A análise conjunta de múltiplas tabelas fato pode revelar relacionamentos interessantes envolvendo dados de diversos assuntos de um *data warehouse*.

A mineração de dados envolve um conjunto de tarefas que permitem extrair conhecimento de bases de dados. As principais tarefas de mineração são classificação, agrupamento e associação. Na classificação, um conjunto de treinamento com dados pré-classificados é usado para a criação de regras de classificação que permite atribuir os registros de dados analisados a classes. No agrupamento, uma coleção de registros de dados é agrupada de acordo com o seu conteúdo, sem a existência de um conhecimento prévio. Já, a tarefa de associação, que é alvo desta pesquisa, permite encontrar padrões que correlacionam a ocorrência de itens na base de dados. Um exemplo clássico de regra de

associação é "*Em um supermercado, os compradores de pão e manteiga tendem também a comprar leite*" [Agrawal_1993].

A proposta deste projeto de pesquisa é desenvolver técnicas que permitam relacionar múltiplas tabelas fato de um *data warehouse* através da mineração de regras de associação, possibilitando a análise conjunta de múltiplos assuntos. Essas técnicas podem ser úteis para uma grande variedade de aplicações como, por exemplo, definições de estratégias de *marketing* e influência de diferentes aspectos na determinação do clima de uma região.

Tradicionalmente, as técnicas de mineração requerem que os dados sejam previamente transferidos para uma única tabela para serem submetidos ao processo de mineração. Entretanto, ao se trabalhar com múltiplas tabelas fato de um *data warehouse*, o processo de transferência de dados implica em perda de informação e envolve custos altos. As tabelas fato de um *data warehouse* sempre tem uma grande quantidade de dados e esses dados são replicados quando transferidos para uma única relação através de uma operação de junção. Assim, a abordagem que usamos durante o projeto de pesquisa é analisar cada tabela fato separadamente.

As técnicas que foram desenvolvidas durante o projeto de pesquisa, para a mineração de regras que relacionam múltiplas tabelas fato de um *data warehouse*, também podem ser usadas para a mineração de regras de associação envolvendo múltiplas relações de uma base de dados relacional, permitindo uma gama ainda maior de aplicações deste trabalho de pesquisa.

1.2 Objetivos

O objetivo deste projeto é desenvolver técnicas de mineração que permitam relacionar dados provenientes de múltiplas tabelas fato de um *data warehouse*. A mineração de dados é uma área muito extensa e, embora exista uma carência de desenvolvimento de técnicas para minerar vários tipos de conhecimento envolvendo múltiplas tabelas fato de um *data warehouse*, este projeto foca a mineração de regras de associação. As regras tratadas neste trabalho recebem o nome de regras de associação *multifatos*.

1.3 Breve Histórico

A tarefa de mineração de regras de associação tem sido amplamente pesquisada na literatura, onde técnicas foram desenvolvidas para obter padrões analisando uma relação [Agrawal et al, 1993, Agrawal and Srikant, 1994, Agrawal and Srikant, 1995, Cheung et al, 1996b, Houtsma and Swami, 1993, Klemettinen et al, 1994, Srikant and Agrawal, 1996, Zaki et al, 1997], múltiplas bases de dados em conjunto [Zhang et al, 2003, Zhong et al, 2001] e dados provenientes de *sites* distribuídos [Cheung et al, 1996b, Kantarcioglu and Clifton, 2002, Otey et al, 2003]. Inclusive aspectos como preservação de privacidade dos dados foram tratados ao analisar transações provenientes de diferentes *sites* [Agrawal and Aggarwal, 2001, Agrawal and Ramakrishnan, 2000, Clifton et al, 2002, Evfimievski et al, 2002, Evfimievski et al, 2003, Kantarcioglu and Clifton, 2002, Thuraisingham, 2002, Vaidya and Clifton, 2002, Veloso et al, 2003]. Alguns trabalhos na literatura foram propostos para minerar múltiplas relações de uma base de dados dedutiva, em conjunto [Deshape and Raedt, 1997, Nijssen and Kok, 2001]. Em [Deshape and Raedt, 1997] uma base de conhecimento, contendo múltiplas relações, foi explorada a partir de um conjunto de predicados pré-definidos. Uma introdução bastante interessante sobre a mineração envolvendo múltiplas relações de bases de dados dedutivas foi feita por Džeroski [Džeroski, 2003] que em [Džeroski and Raedt, 2002], enfatizou a necessidade de desenvolvimento de técnicas de mineração envolvendo múltiplas relações, para solucionar os problemas de extração de padrões a partir de dados estruturados.

1.3.1 Publicações resultantes deste trabalho de pesquisa

Em [Ribeiro and Vieira, 2004] foi proposta uma abordagem para a mineração relacionando múltiplas tabelas fato em conjunto, que também pode ser aplicada para a mineração envolvendo múltiplas relações em conjunto. Essa abordagem soluciona o problema de distorção que pode ocorrer se as abordagens tradicionais forem empregadas para analisar múltiplas tabelas distintas e, além disso, usa uma nova medida de interesse que permite minerar regras cujos itens sejam importantes no relacionamento das diferentes tabelas.

1.4 Organização da Monografia

Esta monografia está organizada como segue. No Capítulo 2 são apresentados os conceitos de *data warehouse*. No Capítulo 3 são apresentados o processo de descoberta de conhecimento, o processo de mineração de dados e os principais tipos de tarefa de mineração. No Capítulo 4 a tarefa de associação é explorada mais a fundo, onde os tipos de regras de associação e os tipos de processo de mineração são discutidos. No Capítulo 5 são detalhados os principais algoritmos de mineração de regras de associação. No Capítulo 6, a técnica de mineração de regras de associação *multifatos*, desenvolvida durante este trabalho de pesquisa, é apresentada. No Capítulo 7 são feitas as conclusões, onde são discutidas as principais contribuições deste projeto de pesquisa e os possíveis trabalhos futuros.

2 Data warehouse

2.1 Introdução

Com a evolução da tecnologia de informação e dos meios de armazenamento e o crescimento do uso de computadores, muitas organizações estão utilizando sistemas informatizados para armazenar os dados sobre seus processos mais importantes. Essa tendência acabou por gerar uma enorme quantidade de dados sobre os fluxos dos processos operacionais de cada organização. Para a integração e gerenciamento dessa grande quantidade de dados históricos surgiu a tecnologia de *data warehouse*.

Um *data warehouse*, ou em português, armazém de dados, pode ser definido como um banco de dados especializado, o qual integra e gerencia o fluxo de informações a partir dos bancos de dados corporativos e fontes de dados externas à uma determinada empresa. A função do *data warehouse* é integrar e disponibilizar as informações corporativas, sem interferir nas bases de dados operacionais. Assim como as bases de dados tradicionais, os *data warehouses* são dispositivos para armazenamento de informações, porém são estruturados e otimizados para a análise de dados.

O *data warehouse*, em uma primeira instância, serve como *um sistema de apoio a decisão* (SAD) fornecendo respostas a questões *simples* através de consultas sobre dados históricos armazenados. No entanto, em se tratando de questões mais complexas, como: *"determine os produtos que tendem a ser vendidos juntos"*, técnicas de mineração de dados devem ser aplicadas sobre os dados do *data warehouse* para permitir que esse tipo de análise seja efetuada. Assim, a aplicação de *data mining* sobre *data warehouses* permite análises mais abrangentes sobre os dados.

Este Capítulo visa discutir os principais conceitos envolvendo a tecnologia de *data warehouse* - características, população, modelos de dados usados para projetar um *data warehouse* e consultas sobre o *data warehouse* - fornecendo o embasamento teórico relativo a *data warehouse* utilizado neste projeto de pesquisa.

O tema deste projeto de pesquisa surgiu diante da limitação da mineração de dados para relacionar dados sobre múltiplos assuntos de um *data warehouse*. Essa limitação foi constatada quando a mineração de dados aplicada sobre múltiplas tabelas fato de um *data warehouse* de um ambiente de ensino a distância, desenvolvido em [Silva, 2002], levava a resultados distorcidos.

Este Capítulo está organizado como segue. Na seção 2.2 são discutidas as principais características dos *data warehouses*. A seção 2.3 apresenta o modelo multidimensional usado para projetar *data warehouses*. Na seção 2.4 são apresentadas as operações OLAP, que permitem consultar os dados de um *data warehouse*, e os servidores OLAP. Na seção 2.5 é apresentado o processo de *data warehousing*, que consiste em um conjunto de tecnologias usadas para popular e analisar os dados de um *data warehouse*. Finalmente na seção 2.6 são feitas as considerações finais deste capítulo.

2.2 Características dos data warehouses

Existem várias definições do termo *data warehouse.* O *data warehouse* é um repositório que provê acesso para análise de dados, descoberta de conhecimento e tomadas de decisões [Elmasri and Navathe, 2000]. Segundo Kimball, em [Kimball, 1996b], o *data warehouse* é uma cópia dos dados transacionais especificamente estruturada para o processamento de consulta e análise de dados.

Os sistemas de banco de dados podem ser classificados em dois tipos: OLTP (*On-Line Transaction Processing*) e OLAP (*On-Line Analytical Processing*). Enquanto o primeiro executa *on-line* a maior parte das operações diárias em uma organização, como por exemplo, a operação de retirar dinheiro de uma determinada conta, o segundo executa, basicamente, operações *off-line* que servem para tomadas de decisões e análise de dados históricos.

As bases de dados operacionais executam operações OLTP para a consulta de seus dados, já os *data warehouses* executam operações OLAP. As operações OLAP permitem organizar e exibir os dados de um *data warehouse* em vários formatos.. As principais diferenças, citadas em [Han and Kamber, 2001], entre OLTP e OLAP são exibidas na **Figura 2.1**.

Característica	OLAP	OLTP
Direcionamento	Orientado para análises gerenciais. Permite consultas a dados históricos para a elaboração de relatórios.	Orientado para o cliente. Permite que transações, como retirada de dinheiro de uma conta, sejam efetuadas.
Tipo de Operação	Apenas permitem operações de leitura.	Permite operações atômicas de atualização de dados.
Tipo de dados Envolvidos	Dados históricos.	Dados correntes.
Modelo de Dados Usado	Tipicamente usa um modelo de dados multidimensional.	Tipicamente usa o Modelo Entidade e Relacionamento.

Figura 2.1: *Principais diferenças entre sistemas OLAP e OLTP*

O *data warehouse* é mantido separado da base de dados operacional, pois esses dois sistemas têm diferentes funcionalidades. Um argumento usado para manter a separação entre *data warehouse* e base de dados operacional é a garantia de uma alta performance em ambos, no *data warehouse* e na base de dados operacional.

2.3 O Modelo Multidimensional

O primeiro passo para efetivamente usar um *data warehouse* é projetá-lo adequadamente e para isso deve ser usado um modelo de dados adequado. O modelo adotado para ser a base dos projetos de um *data warehouses* é o modelo multidimensional. No modelo multidimensional uma matriz é usada para armazenar os dados. Cada dimensão da matriz diz respeito a um atributo, que pode ser o objeto de análise. O valor de cada célula da matriz corresponde a uma medida ocorrida. É uma técnica particularmente útil para a sumarização e arranjo dos dados para facilitar sua análise. Em [Kimball, 1997] são apresentadas as principais diferenças entre a modelagem segundo o modelo entidade-relacionamento e a modelagem segundo o modelo multidimensional. Esforços têm sido direcionados para uma modelagem direta e natural de bases de dados multidimensionais [Vassiliadis and Sellis, 1999].

O modelo multidimensional é mais apropriado que o modelo relacional para operações do *data warehouse*, como, por exemplo, sumarização e agregação, isso porque a

estrutura multidimensional facilita a manipulação dos dados. Assim, *data warehouses* e ferramentas OLAP são geralmente baseadas no modelo multidimensional, pois esse modelo tem maior performance na execução de consultas e na análise dos dados do que o modelo relacional. No modelo multidimensional, os dados são visualizados em cubos de dados. Um cubo de dados permite que os dados sejam visualizados e modelados em múltiplas dimensões. A **Figura 2.2** mostra um exemplo do modelo multidimensional, onde o cubo de dados ilustrado armazena em cada célula o número de alunos relativos aos valores de suas três dimensões: *Disciplinas*, *Notas* e *Ano*.

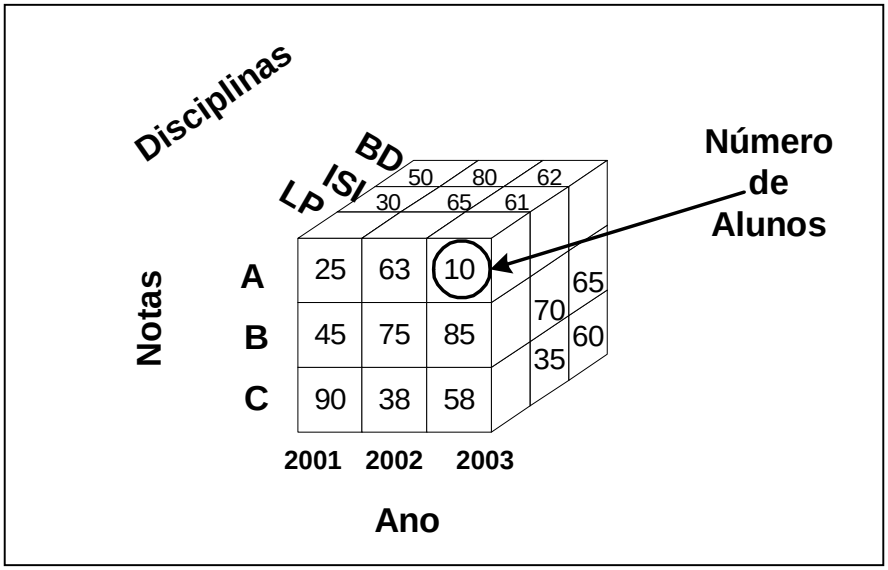


Figura 2.2:Exemplo de um cubo de dados

Os dados da matriz multidimensional podem ser representados no modelo relacional através de uma relação, porém essa representação implica em uma grande duplicação de dados, conforme mostrado na **Tabela 2.1**.

O modelo multidimensional é composto de uma tabela fato, com múltiplas chaves e um conjunto de tabelas menores, chamadas de tabelas dimensões [Kimball, 1996a]. Uma tabela fato representa um assunto do *data warehouse* e ela contém um conjunto de medidas identificadas por ponteiros para as tabelas dimensões. As dimensões são perspectivas ou entidades das quais se deseja armazenar valores [Han and Kamber, 2001]. Tempo, itens e localização são exemplos de dados armazenados nas tabelas dimensões.

Disciplina	Nota	Ano	Número de Alunos
LP	A	1998	25
LP	A	1999	63
LP	A	2000	10
LP	B	1998	45
LP	B	1999	75
LP	B	2000	85
LP	C	1998	90
LP	C	1999	38
LP	C	2000	58
ISI	...	...	...
...	...	...	...

Tabela 2.1: *Representação do modelo multidimensional através de uma relação*

O modelo multidimensional pode existir na forma de *esquema estrela*, *esquema flocos-de-neve* ou *constelação de fatos* [Han et al, 2000]:

- *Esquema Estrela:* é o esquema mais comum. Nesse esquema o *data warehouse* contém uma tabela central chamada de tabela fato e um conjunto de tabelas dimensão. Um exemplo de esquema estrela, usando um *data warehouse* referente ao domínio de educação a distância, é ilustrado na **Figura 2.3**.
- *Esquema Flocos-de-Neve:* esse esquema é uma variação do esquema estrela, onde algumas dimensões são normalizadas, isto é, são compostas de mais de uma tabela. Embora as redundâncias sejam diminuídas e o espaço de armazenamento seja poupado, esse esquema influencia negativamente a performance de consultas, pois a varredura é mais lenta devido à necessidade de mais junções entre tabelas, conforme citado em [Silva, 2002]. Um exemplo de esquema flocos-de-neve é mostrado na **Figura 2.4**.
- *Esquema Constelação de Fatos ou Multifatos*: são estruturas mais complexas do que o esquema estrela, pois permitem que múltiplas tabelas fato compartilhem tabelas dimensões. Um exemplo de esquema constelação de fatos é mostrado na **Figura 2.5**.

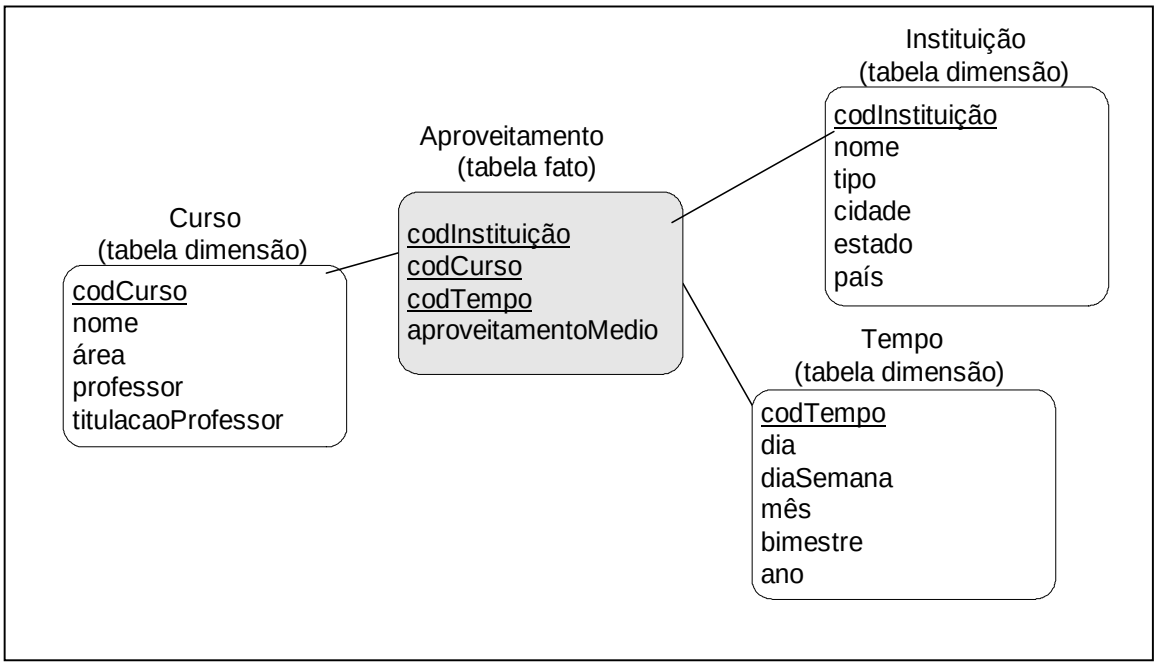


Figura 2.3: *Exemplo de Esquema Estrela* [Silva, 2002]

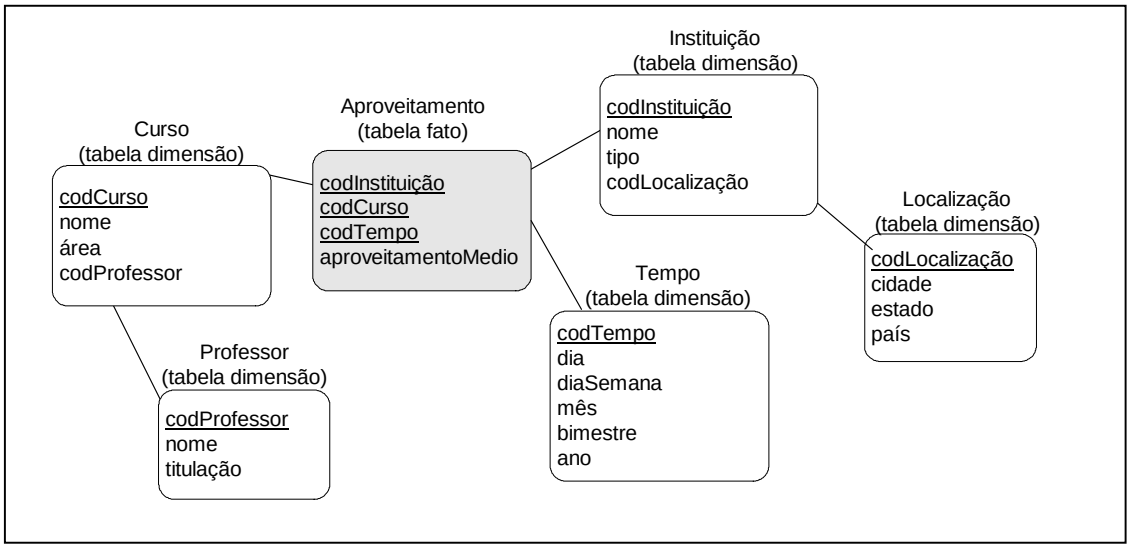


Figura 2.4: *Exemplo de Esquema Flocos de Neve* [Silva, 2002]

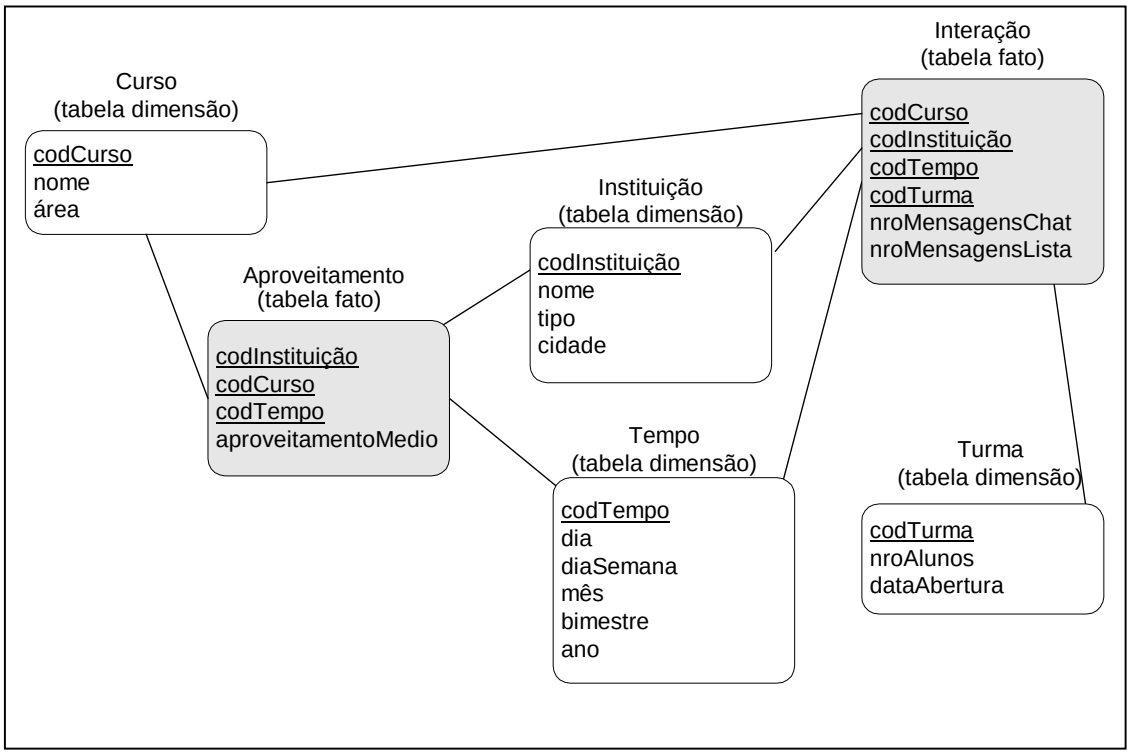


Figura 2.5: *Esquema Constelação de Fatos ou Multifatos* [Silva, 2002]

2.4 OLAP

No modelo multidimensional os dados são organizados em múltiplas dimensões, as quais podem ser visualizadas em diferentes níveis de abstração usando as operações OLAP. Essas operações são aplicadas sobre o *data warehouse* providenciando um ambiente interativo de análise de dados. Existem inúmeras operações OLAP como, por exemplo, as seguintes operações, discutidas em [Kimball, 1996a]:

- ***Drill Down:***: é uma das operações mais antigas do OLAP; consiste em aumentar o nível de detalhe do que é visualizado. Permite ao usuário navegar a partir de dados menos detalhados para dados mais detalhados. Essa operação deixa a granularidade dos dados mais fina. Um exemplo de como se pode caminhar para menores níveis de abstração de dados através da operação *Drill Down* é apresentado na **Figura 2.6**.
- ***Drill Up*** ou ***Roll Up:*** a operação de *drill up*, também chamada de *roll up*, consiste em reduzir o nível de detalhamento dos dados visualizados no cubo. Essa operação produz

agregações no cubo de dados, deixando a granularidade dos dados mais grossa. Um exemplo de como se pode caminhar para níveis mais alto de abstração de dados através da operação *Roll Up* é apresentado na **Figura 2.6**.

- ***Drill Across:*** consiste em fazer uma junção de duas ou mais tabelas fato que estão relacionadas entre si através de dimensões comuns. Muito cuidado deve ser tomado ao usar a operação de *drill across* uma vez que a junção pode acarretar em duplicações nos dados e distorções no resultado da consulta. Para a execução da operação *drill across*, as operações *OLAP (drill down e roll up)* são aplicadas separadamente em cada tabela fato, de modo a obter relatórios separados para cada tabela fato. Os resultados dessas operações são juntados a partir de uma operação de *outer join* sobre as colunas de agrupamento provenientes das dimensões comuns. A operação de *drill across* aplicada sobre o esquema de constelação de fatos mostrado na **Figura 2.5**, permite visualizar, em conjunto, dados provenientes da junção das tabelas fato *Aproveitamento* e *Interação*.

- ***Drill Around:*** nesta operação múltiplas tabelas fato agrupam-se em torno de uma dimensão comum, e a junção entre as múltiplas tabelas fato é feita através dessas dimensões comuns. A operação *drill around* é um caso especial de aplicação da operação *drill across* para a situação em que múltiplas tabelas fato se agrupam em torno de apenas uma dimensão comum, sendo executada do mesmo modo que a operação *drill across*. Assim, a operação de *Drill Around* também pode ser aplicada sobre o esquema de constelação de fatos mostrado na **Figura 2.5**.

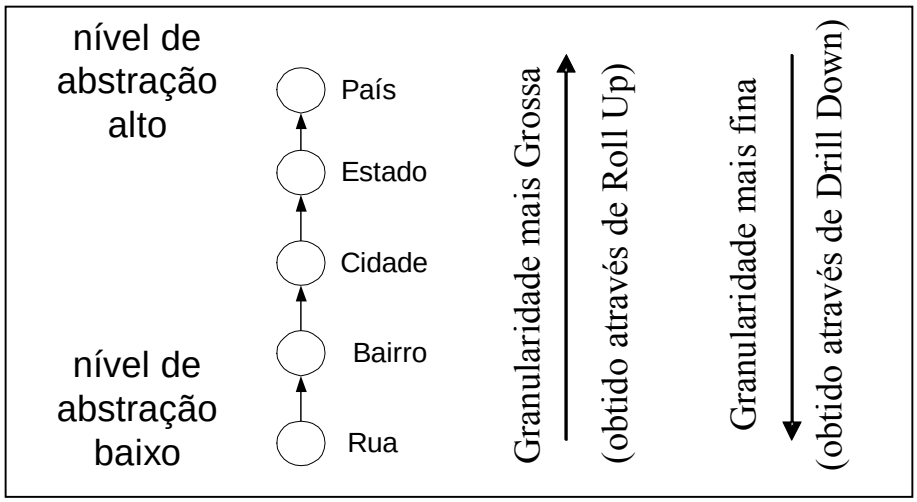


Figura 2.6: *Níveis da abstrações do domínio localização obtidos através de operações OLAP*

2.4.1 Tipos de servidores OLAP: ROLAP, MOLAP E HOLAP

O Servidor OLAP é uma ferramenta responsável pela manipulação dos dados de um *data warehouse* com a finalidade de permitir que o usuário consulte e examine os dados. A arquitetura física e a implementação dos servidores OLAP precisam considerar a forma em que os dados estão armazenados. Os tipos de servidores OLAP mais comuns encontrados na literatura são:

- *ROLAP – Relational On-line Analytical Processing:* esses servidores manipulam *data warehouses* gerenciados por SGBD's relacionais, constituindo uma camada de interface entre o modelo relacional e o modelo multidimensional, pois transformam as requisições multidimensionais do usuário em rotinas de acesso às tabelas que armazenam os dados. Sua vantagem é a eficiência no armazenamento de dados esparsos.

- *MOLAP – Multidimensional On-line Analytical Processing:* servidores MOLAP executam operações diretamente sobre uma matriz de dados. Se os dados não forem esparsos, esses servidores são mais eficientes em armazenamento e recuperação do que os servidores ROLAP.

- *HOLAP – Hydrid On-line Analytical Processing:* os servidores HOLAP adotam uma forma de armazenamento em dois níveis, um para dados densos, que são colocados em matrizes e outro para dados esparsos, que são alocados em tabelas.

2.5 Data Warehousing

Data Warehousing é o processo de popular e usar *data warehouses*. Segundo Chaudhuri e Dayal em [Chaudhuri and Dayal, 1997] o processo de *data warehousing* constitui-se de uma seqüência de passos:

- *Extração dos dados das fontes heterogêneas*: os dados são extraídos das bases de dados operacionais para popular os *data warehouses*.

- *Limpeza e integração dos dados*: os dados são limpos de maneira que as redundâncias e as diferenças de formatos sejam eliminadas.

- *Armazenamento no data warehouse*: os dados são efetivamente armazenados no *data warehouse* após passarem pelo processo de limpeza e integração.

- *Gerenciamento e acesso aos dados do data warehouse através de servidores OLAP*: freqüentemente as consultas aos dados do *data warehouse* são feitas através da execução de operações *OLAP*.

- *Análise dos dados através da mineração de dados*: os dados de um *data warehouse* podem ser usados em um processo de mineração de dados, para que sejam efetuadas análises mais complexas e abrangentes sobre os dados do que aquelas que as operações *OLAP* permitem.

A **Figura 2.7** ilustra o processo de *data warehousing*. Observe que os dados de análise também podem ser usados para realimentar o *data warehouse*, servindo como base para futuras análises.

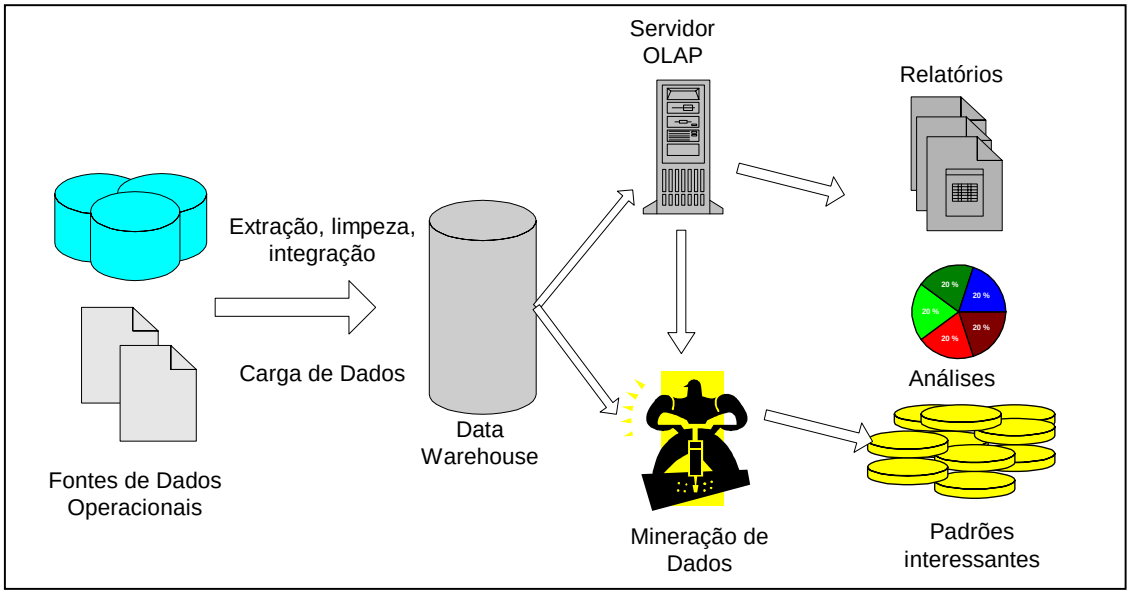


Figura 2.7: O processo de *data warehousing* [Chaudhuri and Dayal, 1997]

Quanto maior a qualidade dos dados de um *data warehouse* melhor serão os resultados obtidos pelas análises dos seus dados. A qualidade dos dados de um *data warehouse* é determinada pelo estado de completude, consistência e precisão, que tornam

seus dados mais apropriados para a análise do que os dados de outros repositórios [Hufford, 1996].

2.6 Considerações Finais

O *data warehouse* é um repositório de dados muito usado para armazenamento de informações históricas nas organizações. Os dados antes de serem armazenados no *data warehouse* passam por um processo de limpeza e integração, fazendo com que a qualidade dos mesmos seja melhorada. Os dados são armazenados em uma estrutura otimizada para promover análises e são acessados através de consultas OLAP. As consultas OLAP permitem apenas uma análise superficial sobre os dados. Quando análises mais abrangentes são necessárias, devem-se usar as tecnologias de mineração de dados em conjunto com o *data warehouse*. O uso do *data warehouse* como repositório de dados para a mineração, faz com que a mesma se beneficie do pré-processamento (processo de limpeza e integração) pelo qual os dados passaram ao popular o *data warehouse*.

Neste capítulo foram apresentados os conceitos básicos envolvidos na tecnologia de *data warehouse*: o modelo multidimensional, as operações OLAP e o processo de *data warehousing*. O esquema constelação de fatos será constantemente referenciado no decorrer desta monografia, pois os *data warehouses* que envolvem múltiplos assuntos são modelados segundo esse tipo de esquema, sendo a análise conjunta desses múltiplos assuntos, através da mineração de dados, o enfoque desta pesquisa.

3 Descoberta de Conhecimento e Mineração de Dados

3.1 Introdução

A facilidade com que os sistemas computacionais geram uma imensa quantidade de dados fez com que se tornasse necessário o desenvolvimento de técnicas para extrair padrões sobre os dados armazenados. Atualmente, análises complexas e sofisticadas sobre o comportamento dos dados são requeridas para que decisões estratégicas sejam tomadas. Um exemplo desse tipo de análise seria determinar os efeitos das políticas de uma companhia de cartão de crédito no consumo de equipamentos eletrônicos de uma determinada rede de lojas.

O processo de Descoberta de Conhecimento em Base de Dados (KDD - Knowledge Discovery in Databases) é usado para que padrões sejam extraídos das bases de dados. Esse processo possui várias etapas envolvendo preparação dos dados e mineração dos mesmos. A preparação dos dados envolve limpeza, agregação e consolidação dos mesmos.

A mineração de dados (*data mining*), que é a parte principal do processo de KDD, tem sido intensamente discutida na literatura nos últimos anos. Ela constitui-sena etapa do processo de KDD onde efetivamente é feita a análise dos dados para a descoberta de padrões. Nessa fase, algoritmos são executados sobre os dados previamente preparados, para extração de conhecimento. Muitas vezes, devido à sua importância no processo, o termo mineração de dados é usado erroneamente para se referir a todo o processo de KDD, no entanto, o processo de KDD engloba as fases de limpeza, integração e apresentação dos dados, que não são consideradas no processo de mineração [Han and Kamber, 2001]. A realização de *data warehousing*, discutida no Capítulo 2, pode ser usada como a fase de preparação dos dados para a análise através da mineração de dados.

Este Capítulo está organizado como segue. Na seção 3.2 é apresentado o processo de descoberta de conhecimento. Na seção 3.3 é apresentado o processo de mineração de dados e nas seções seguintes são apresentadas as principais tarefas de mineração usadas na literatura: a tarefa de associação (seção 3.4); a tarefa de classificação (seção 3.5); a tarefa

de agrupamento (seção 3.6), e a tarefa de mineração de padrões seqüenciais (seção 3.7). Finalmente, na seção 3.7 são apresentadas as considerações finais deste capítulo.

3.2 Processo de Descoberta de Conhecimento

A definição mais usada de processo de Descoberta de Conhecimento em Base de Dados -KDD- foi a fornecida por Fayyad [Fayyad et al, 1996]:

"A descoberta de conhecimento é um processo não trivial de identificação de padrões embutidos nos dados que sejam válidos, novos, potencialmente úteis e compreensíveis"

O processo de KDD compreende um conjunto de etapas [Elmasri and Navathe, 2000]: 1) seleção dos dados; 2) limpeza dos dados; 3) integração dos dados; 4) transformação dos dados; 5) mineração dos dados; 6) apresentação do conhecimento. Os objetivos de cada etapa do processo de KDD são apresentados na **Tabela 3.1**. Na **Figura 3.1** é ilustrado o processo de descoberta de conhecimento.

Etapa	Objetivo
1) Seleção dos dados	Buscar na base os dados relevantes para a tarefa de análise.
2) Limpeza dos dados	Remover dados inconsistentes.
3) Integração dos dados	Compatibilizar os dados, no caso de serem provenientes de múltiplas fontes.
4) Transformação dos dados	Transformar os dados para o formato de entrada dos algoritmos de mineração.
5) Mineração dos dados	Aplicar algoritmos para extração de conhecimento dos dados.
6) Apresentação do conhecimento	Apresentar o conhecimento minerado usando técnicas adequadas de representação.

Tabela 3.1: *Objetivos das etapas do processo de KDD*

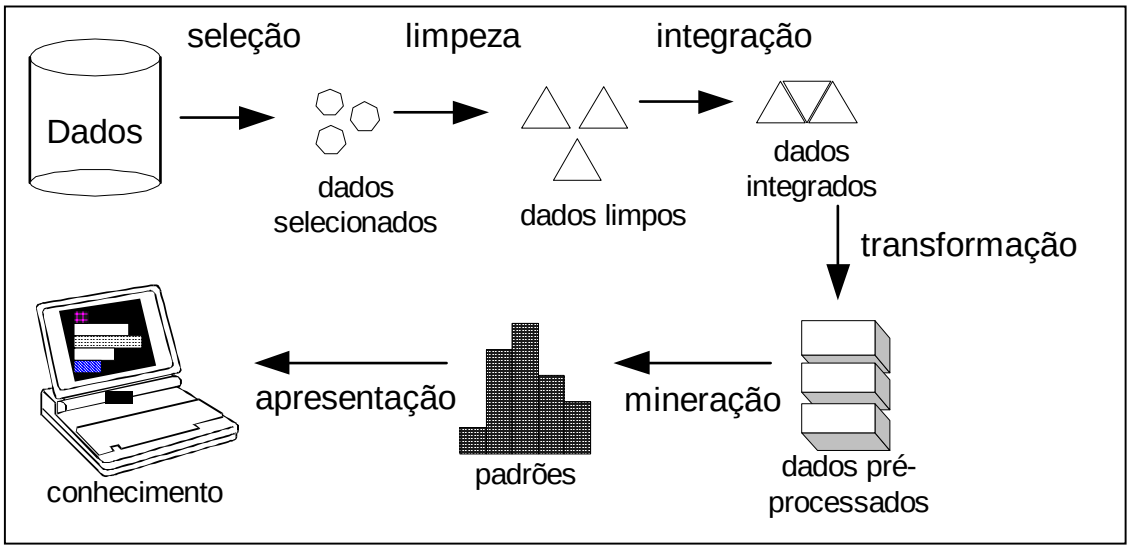


Figura 3.1: *Processo de descoberta do conhecimento.*

3.3 Mineração de Dados

A mineração de dados é um campo multidisciplinar incluindo áreas de inteligência artificial, estatística, processamento de imagens (reconhecimento de padrões) e banco de dados [Roden et al, 1999]. A informação extraída através da mineração de dados pode ser usada em uma grande variedade de aplicações tais como na análise de mercado, no controle de produção e nas análises de dados biomédicos.

O objetivo de aplicar mineração de dados é a verificação de uma hipótese ou obtenção de novos padrões sobre os dados. Assim, as tarefas de mineração podem ser de *predição* ou *descrição*. Enquanto as tarefas de predição constroem modelos para predizer o comportamento de dados futuros, as tarefas de descrição revelam padrões e propriedades presentes nos dados analisados.

Para a aplicação dos algoritmos de mineração sobre os dados é necessária a definição dos seguintes fatores:

- *Atributos relevantes para análise*: freqüentemente, a análise de todos os atributos de uma base de dados resulta em uma grande quantidade de padrões desinteressantes e sem significado. Assim, recomenda-se que somente os atributos de interesse sejam previamente selecionados para serem submetidos ao processo de mineração, reduzindo o número de padrões desinteressantes gerados.

- *Tarefa de Mineração Adequada*: existe um grande número de padrões diferentes que pode ser obtido através da mineração de dados: classificação, associação, regressão linear, agrupamento, etc. Assim, é importante saber qual o tipo de padrão desejado para que seja usada a técnica de mineração adequada.
- *Medidas de interesse usadas:* as medidas de interesse impõem restrições para a mineração de regras, eliminando padrões encontrados que não a satisfazem. A função da medida de interesse é evitar a mineração de padrões que não são interessantes.

Uma *tarefa de mineração* é um conjunto de técnicas usadas para extrair determinado tipo de padrão sobre os dados. A seguir, neste Capítulo, são apresentadas as principais tarefas de mineração de dados

3.4 Regras de Associação

A tarefa de associação permite relacionar a ocorrência de um determinado conjunto de itens com a ocorrência de outro conjunto de itens. Uma regra de associação é uma regra da forma $X \rightarrow Y$, onde X e Y são conjuntos de itens, significando que se X ocorre em uma transação da base de dados Y também tende a ocorrer.

A análise da associação em um banco de dados pode gerar uma grande quantidade de regras de associação. Algumas dessas regras podem não ser interessantes, pois ocorrem com baixa freqüência nos dados. Para contornar esse problema foram definidos parâmetros (medidas de interesse) que determinam quais associações são interessantes ou não para o usuário. Assim, para uma regra de associação ser considerada satisfatória, ela deve satisfazer algumas medidas de interesse. As medidas de interesse mais comuns são o suporte e a confiança, que são discutidas no Capítulo 4.

Para exemplificar a tarefa de associação, considere os dados de compra de produtos apresentados na **Tabela 3.2**.

Transação	Itens
1	cerveja, fralda, leite
2	arroz, cerveja, fralda
3	cerveja, fralda
4	Fralda, manteiga, pão
5	manteiga, farinha

Tabela 3.2: *Exemplo de transações de um supermercado*

A partir dos dados da **Tabela 3.2**, um exemplo de regra que pode ser minerada é apresentado a seguir:

$$fralda \rightarrow cerveja$$

É possível observar que os itens fralda e cerveja aparecem juntos em 60% das transações da base de dados e, de todas as transações que possuem fralda, 75% também contém cerveja. Conforme veremos mais adiante, os valores 60% e 75%, obtidos no exemplo anterior, representam os valores de suporte e a confiança da regra *fralda* → *cerveja*.

A tarefa de associação é o foco desta pesquisa, e por isso, o Capítulo 4 desta monografia é dedicado a ela.

3.5 Classificação

A tarefa de classificação permite agrupar dados em uma hierarquia de classes, de acordo com os valores dos seus atributos. Os registros agrupados em classes são formados por um atributo alvo (ou atributo de classe) que determina a classe do registro, e um conjunto de atributos de predição. O objetivo é descobrir as relações existentes entre os atributos de predição e o atributo alvo, utilizando registros cuja classificação é conhecida. A tarefa de classificação é, portanto, uma tarefa de predição, uma vez que ela prediz os valores dos atributos alvo.

Na **Figura 3.2** é ilustrado o processo de classificação. A mineração de regras de classificação é feita em duas etapas. Na primeira etapa um algoritmo de classificação é aplicado sobre uma amostra do banco de dados, que é chamada de conjunto de treinamento. O conjunto de treinamento contém registros do banco cuja classificação é conhecida, ou seja, apresenta dados com o valor do atributo alvo preenchido.O resultado da execução da

primeira etapa é um conjunto de regras de classificação. Na segunda etapa, as regras de classificação mineradas anteriormente são utilizadas para classificar a base de dados.

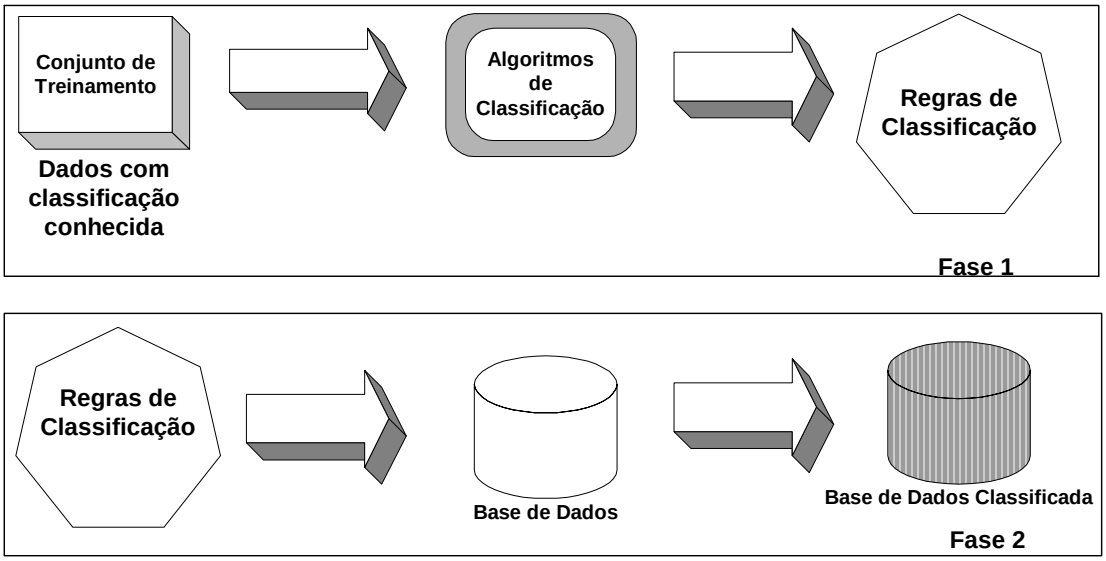


Figura 3.2: *As duas fases do Processo de Classificação*

Um exemplo do processo de mineração de regras de classificação é fornecido a seguir. Considere os dados da **Tabela 3.3** como sendo um conjunto de treinamento com dados relativos ao sucesso de lançamentos de novos aparelhos de barbear.

Número De Lâminas	**Publico Alvo**	**Nível Preço**	**Sucesso**
1	F	Caro	Não
2	F	Barato	Sim
3	F	Caro	Sim
3	M	Barato	Não
2	M	Caro	Não
2	M	Barato	Sim
1	F	Barato	Não
2	F	Caro	Sim
3	F	Barato	Sim

Tabela 3.3: *Dados relativos ao sucesso de lançamentos de novos aparelhos de barbear*

A partir dos dados da **Tabela 3.3**, é possível gerar as seguintes regras de classificação:

Se (NROLÂMINAS = 2) e (PREÇO = "barato") ENTÃO SUCESSO = "sim"

Se (NROLÂMINAS = 1) ENTÃO SUCESSO = "não"

Se (NROLÂMINAS = 3) e (PUB_ALVO = "FEMININO") ENTÃO SUCESSO = "sim".

Existe um conjunto de técnicas usadas para a obtenção de regras de classificação, sendo as mais usadas:

- **Classificação *Bayesiana*:** É uma classificação estatística baseada no teorema de *Bayes*. Na classificação *Bayesiana* é assumido que o efeito do valor de um atributo não tem influência nos demais atributos da mesma classe. As redes *Bayesianas*, comumente usadas para classificação, também podem ser usadas para criar agrupamentos.
- **Indução usando redes neurais:** O uso de redes neurais simula o método de funcionamento do cérebro através de uma estrutura de dados computacional. Neste tipo de classificador existe um conhecimento adquirido através da comparação da predição com cada amostra do conjunto de dados de treinamento. A vantagem desse método é que ele apresenta as propriedades do cérebro de tolerância a falhas, lógica imprecisa e paralelismo.
- **Indução usando árvore de decisão**: Amplamente utilizadas em algoritmos de classificação, as árvores de decisão são representações simples e eficientes do conhecimento. Maiores detalhes sobre o uso de árvores de decisão são fornecidos na próxima subseção.

3.5.1 Gerando regras de classificação através de árvores de decisão

Uma árvore de decisão tem a função de particionar recursivamente um conjunto de treinamento, até que cada subconjunto obtido deste particionamento contenha casos de uma única classe [Han and Kamber, 2001]. Para atingir essa meta, a técnica de árvores de decisão examina e compara a distribuição de classes durante a construção da árvore. Como resultado da construção de uma árvore de decisão, tem-se os dados organizados de uma maneira compacta, que são utilizados para classificar novos casos.

A **Figura 3.3** mostra uma árvore de decisão construída a partir da amostra de treinamento da **Tabela 3.3**, onde os nós folhas indicam o valor do atributo sucesso para o lançamento de um novo aparelho de barbear.

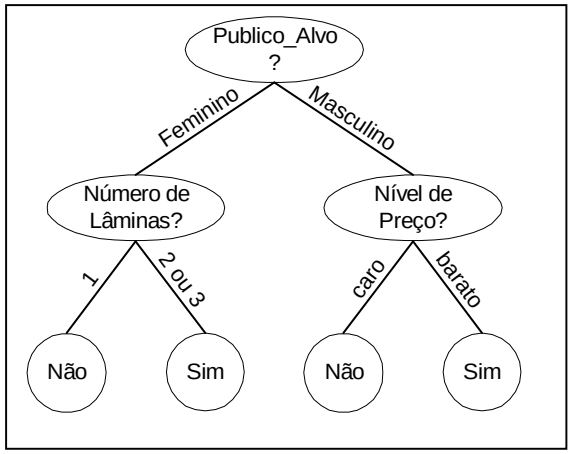


Figura 3.3: *Exemplo de Árvore de decisão*

As árvores de decisão são estruturas onde cada nó interno representa um teste sobre algum atributo, cada ramo representa o resultado de um teste e os nós folha representam a distribuição de classe, ou seja, o valor determinado do atributo alvo. Para classificar uma amostra desconhecida, os valores dos atributos dessa amostra são testados seguindo o fluxo da árvore de decisão. Para a construção da árvore de decisão é medida a influência de cada atributo para a definição da classe (isto é, para a determinação do valor do atributo alvo) através do cálculo de uma função de ganho. O atributo com maior influência é adicionado a um novo nó de teste.

A principal vantagem da árvore de decisão é que ela pode ser mapeada para regras de fácil entendimento.

3.6 Agrupamento (Clustering)

Agrupamento ou *Clustering* é uma tarefa na qual pesquisam-se os dados com a finalidade de identificar grupos de registros similares baseado em valores próximos de seus atributos. Considerando que os dados podem ser representados através de pontos, o agrupamento permite agrupar esses pontos em partições de acordo com a distância entre eles. A distância entre dois pontos indica o quanto eles são similares, com base em valores de determinados atributos. A distância entre os pontos de um mesmo cluster deve ser sempre menor do que a distância entre pontos de clusters distintos.

A tarefa de agrupamento tem várias aplicações, como, por exemplo, a determinação de grupos de clientes baseada em características de suas compras. Além disso, a tarefa de agrupamento pode ser usada como um pré-processamento dos dados para outras tarefas de mineração.

Os métodos mais comuns de agrupamento usados na literatura, segundo [Han and Kamber, 2001] são:

- **Métodos Hierárquicos:** Criam grupos de objetos de maneira hierárquica. Os métodos hierárquicos podem usar a estratégia *bottom-up* ou *top-down* para formar a hierarquia. Os métodos hierárquicos *bottom-up* iniciam com cada objeto formando um grupo e, posteriormente, junções sucessivas dos grupos são feitas. Na estratégia *top-down* ocorre o processo inverso: os objetos iniciam-se no mesmo grupo, e divisões sucessivas geram os demais grupos.

- **Métodos Baseados em *Grid*:** O espaço do objeto é dividido em infinitas células que formam uma estrutura de *Grid*. As operações de agrupamento são baseadas na posição dos objetos na *Grid*. O método baseado em *Grid* é um dos métodos de agrupamento mais rápidos.

- **Métodos Baseados em Modelos:** Nesse método, um modelo é criado hipoteticamente para cada partição e o conjunto de dados que melhor se adapta a cada modelo é encontrado.

- **Métodos de Particionamento:** Dada uma base de dados com n objetos, esses métodos constroem k partições dos dados, onde $k \leq n$. Cada objeto deve pertencer a somente uma partição e cada partição deve ter ao menos um objeto. O algoritmo *K-means*, apresentado na **Figura 3.4**, é um exemplo de algoritmo que usa o método do particionamento. Suponha que o algoritmo *k-means* é aplicado sobre o conjunto de dados da **Figura 3.5(a)**, considerando k = 3. Inicialmente os três objetos, representados por + na **Figura 3.5(b)**, são escolhidos arbitrariamente como *centro* da partição. Cada objeto é então adicionado à partição cujo centro é mais próximo (ver **Figura 3.5(b)**). Posteriormente, o centro de cada partição é recalculado, através do cálculo da média aritmética dos pontos que formam cada partição. Os objetos são alocados para novas partições de acordo com os novos centros calculados, conforme o mostrado na **Figura 3.5(c)**. Esse processo continua

até não haver mais redistribuição de objetos nas partições. A **Figura 3.5(d)** mostra o resultado final do processo de agrupamento.

Algoritmo: *k-means*
Entrada: Numero de *clusters k*, base de dados contendo *n* elementos
Saída: Um conjunto de *k clusters*
Inicio Algoritmo
1. Inicializar os k centros de clusters randomicamente.
2. Repita:
3. Distribuir os n elementos atribuindo cada um deles ao centro do cluster mais próximo.
4. Definir os novos centros dos clusters, calculando a média dos elementos em cada cluster gerado na linha 3.
5. Até que nenhum elemento troque de cluster.
Fim Algoritmo

Figura 3.4: Algoritmo *K-means [Han and Kamber, 2001]*

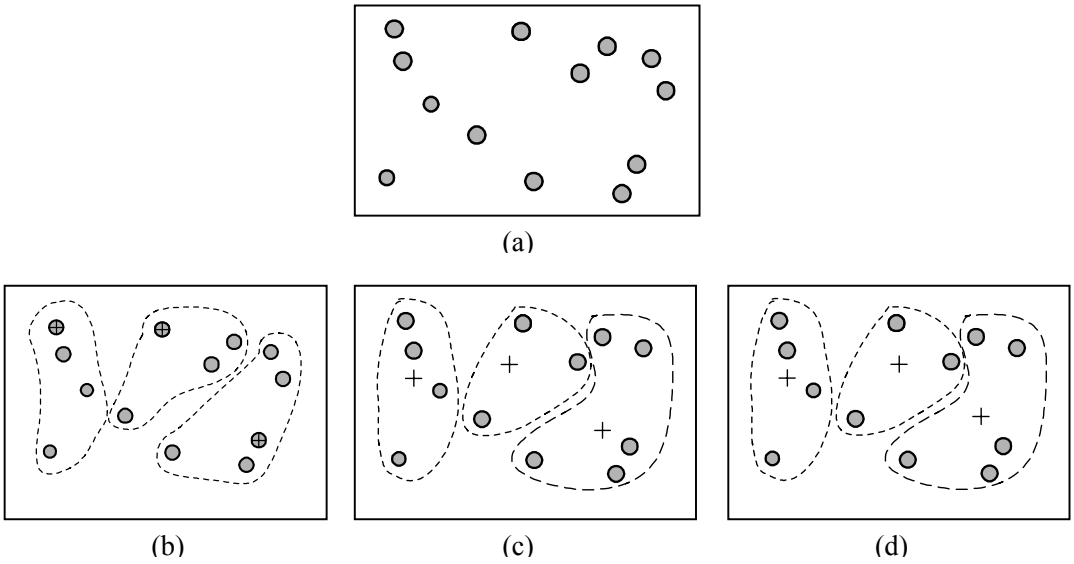


Figura 3.5: *Agrupamento de objetos baseado no método k-means.*

3.7 Padrões Seqüenciais

Consiste em encontrar as seqüências de itens que satisfazem a freqüência mínima de ocorrência definida pelo usuário [Agrawal and Srikant, 1995].

Um exemplo de padrão seqüencial sobre o comportamento de fregueses em uma locadora seria: *os fregueses de uma locadora tipicamente retiram os filmes "Matrix", posteriormente "Senhor dos Anéis" e por último retiram "Harry Potter*". Apesar dos filmes retirados respeitarem a ordem anterior, podem existir outros filmes retirados entre uma locação e outra.

Considere as transações de compras de clientes em um supermercado exibidas na **Figura 3.3(a)**. Para a mineração de padrões seqüenciais, inicialmente são geradas as seqüências de dados conforme ilustrado na **Figura 3.3(b)**, onde os símbolos < e > delimitam as seqüências, e cada elemento da seqüência é delimitado por parênteses. Posteriormente as seqüências são processadas para encontrar aquelas cujo número de ocorrências é maior do que um limite mínimo estabelecido. A **Figura 3.3(c)** mostra as seqüências geradas a partir dos dados da **Figura 3.3(a)**, para uma freqüência mínima de 25%.

ClienteID	Horário	CodItens
1	18/05/2002 14:00:30	30
1	20/06/2002 15:20:00	90
2	31/01/2002 16:30:00	10,20
2	06/02/2002 17:40:00	30
2	13/11/2002 18:50:11	40, 60, 70
3	24/08/2002 19:00:12	30, 50, 70
4	25/09/2002 01:05:13	30
4	15/12/2002 02:16:14	40,70
4	16/01/2003 03:15:15	90
5	14/03/2002 04:12:11	90

(a)

Cliente_ID	Seqüência de itens comprados por Cliente
1	< (30) (90) >
2	< (10 20) (30) (40 60 70) >
3	< (30 50 70) >
4	< (30) (40 70) (90)>
5	< (90)>

(b)

Seqüências com freqüência maior que 25%
< (30) (90)>
< (30) (40,70)>

(c)

Figura 3.6: *Exemplo de extração de padrões seqüências.*

3.8 Considerações Finais

Neste Capítulo foram apresentadas as etapas do processo de descoberta de conhecimento. A mineração de dados é uma etapa do processo de descoberta de

conhecimento que consiste em aplicar algoritmos sobre os dados preparados para a extração de padrões. As principais tarefas de mineração de dados (associação, classificação, agrupamento e padrões seqüenciais) foram apresentadas neste Capítulo. Com o avanço das pesquisas no campo de mineração de dados, novas tarefas mais complexas de mineração estão surgindo para atender a necessidade de análises mais abrangentes dos dados, incluindo a extensão das técnicas de mineração existentes atualmente para analisar tipos mais complexos de dados, como dados multimídia e dados estruturados no formato XML.

Este projeto de pesquisa enfoca a tarefa de associação, por isso o próximo Capítulo é dedicado a ela.

4 Tarefa de Associação

4.1 Introdução

A tarefa de associação é uma tarefa descritiva que busca por associações interessantes entre itens de uma base de dados. A descoberta de relacionamentos entre os dados é extremamente útil em uma grande variedade de aplicações, como por exemplo, na determinação de estratégias de negócios, nos processos de tomada de decisão e na determinação dos efeitos de alterações no clima.

O objetivo deste capítulo é fornecer uma visão geral dos parâmetros e tipos de problemas envolvidos na mineração de regras de associação. Diferentes tipos de problemas de mineração de regras de associação são definidos em função do formato dos dados e da distribuição do mesmo. Além disso, medidas de interesse podem ser usadas no processo de mineração para criar um ranking das regras mineradas, ou para selecionar as mais interessantes dentre as regras mineradas.

Este Capítulo está organizado como segue. Na Seção 4.2 é apresentada a definição geral do problema de regra de associação. Os diferentes tipos de regras de associação são classificados na seção 4.3. Na seção 4.4, os tipos de mineração de regras de associação são apresentados. As medidas de interesse são apresentadas na seção 4.5. Finalmente, na seção 4.6 são apresentadas as considerações finais deste Capítulo.

4.2 Definição de Regras de Associação

O problema de minerar regras de associação foi introduzido em [Agrawal et al, 1993] como um problema de encontrar relacionamentos entre a ocorrência de itens em transações de uma base de dados.

O problema de mineração de dados em uma relação pode ser definido como [Agrawal et al, 1993]: Seja $I = \{i_1, \ldots, i_n\}$ um conjunto de literais, denominados itens. Um conjunto $X \subseteq I$ é chamado de ***itemset***. Um *itemset* X com k elementos é chamado de ***itemset*-k**. Seja R uma relação com tuplas t que envolvem elementos que são subconjuntos de I. A tupla t **suporta** um *itemset* X se $X \subseteq t$. O **suporte** de um *itemset* X é a razão entre o número de tuplas em R

que suportam *X*, e o número total de tuplas de *R*. Um *itemset X* é chamado ***itemset* freqüente** se o suporte de *X* for maior ou igual ao suporte mínimo especificado pelo usuário. Uma **regra de associação** é uma expressão da forma $X \rightarrow Y$, onde *X,Y* são *itemsets*; o **suporte da regra** é a razão entre o número de tuplas em *R* que contém *X* e *Y*, e o número total de tuplas de *R*. A **confiança** é a razão entre o número de tuplas que contém *X* e *Y*, e o número de tuplas que contém *X*. Um exemplo bem conhecido de regra de associação envolvendo dados de uma cesta de compras é "70% das compras que contêm fralda também contêm cerveja e 4% de todas as compras contêm esses dois itens". Nesse exemplo, 70% é a confiança da regra e 4% é o suporte da regra. O **problema de minerar regras de associação** consiste em encontrar todas as regras de associação que satisfazem as restrições de suporte mínimo e de confiança mínima especificadas pelo usuário. Essas regras são chamadas **regras fortes**.

O problema apresentado acima é também conhecido como o problema de minerar regras de associação *booleanas*, assim denominado porque ele pode ser visto como um problema de encontrar associações entre valores "1" em uma tabela que tem um atributo *booleano* para cada item $i_i \in I$ e um registro r_j para cada tupla $t_j \in R$. Em um registro r_j, o valor "1" de um atributo indica que o item representado por esse atributo está presente na tupla t_j, e o valor "0" indica que o item está ausente.

4.3 Classificação das Regras de Associação

As regras de associação são classificadas de acordo com várias características, como por exemplo, tipo de atributo envolvido, número de atributos envolvidos e o nível de detalhamento dos dados envolvidos nas regras. A seguir são apresentados os principais tipos de regra de associação encontrados na literatura.

4.3.1 Quanto ao tipo de atributo envolvido

O domínio dos atributos de uma tabela submetida a um processo de mineração pode ser quantitativo ou categórico. Atributos quantitativos são atributos numéricos contínuos, onde existe uma ordem explícita entre seus valores, como por exemplo, idade e preço. Atributos categóricos são atributos que têm um domínio discreto e finito de elementos, não apresentando uma ordem explícita entre os valores dos elementos, como por exemplo, cor.

As tabelas com atributos categóricos podem ser mapeadas para tabelas formadas por atributos *booleanos.* Observe que os dados da **Tabela 4.1** podem ser representados pelos dados da **Tabela 4.2** formada por atributos *booleanos*. A **Tabela 4.2** tem um atributo correspondente para cada item e uma linha para cada transação. O atributo terá o valor 1 se o seu item correspondente ocorreu naquela transação e, 0 caso contrário.

TID	Itens
T1	Arroz, Cerveja, Fralda, Leite
T2	Fralda, Leite, Pão

Tabela 4.1: *Exemplo de transações realizadas em um supermercado*

TID	Arroz	Cerveja	Fralda	Leite	Pão
T1	1	1	1	1	0
T2	0	0	1	1	1

Tabela 4.2: *Representação booleana dos dados contidos na* ***Tabela 4.1***

Outro tipo de atributo usado nas regras de associação são os atributos nebulosos. Um atributo nebuloso é aquele cujo valor é um termo lingüístico impreciso ou incerto como, por exemplo, *velho*, *novo* e *bastante*.

As regras de associação podem ser classificadas em três categorias de acordo com o tipo de atributo envolvido:

- **Regras de Associação *Booleanas***: quando os atributos envolvidos são categóricos;
- **Regras de Associação Quantitativas**: quando os atributos envolvidos são atributos numéricos contínuos.
- **Regras de Associação Nebulosas**: são regras de associação que envolvem conceitos nebulosos.

Muitas vezes os atributos quantitativos são mapeados em intervalos, para serem tratados na mineração de regras de associação como atributos categóricos. Em outros procedimentos, como o apresentado em [Kuok et al, 1998], os atributos quantitativos são mapeados para valores nebulosos.

4.3.2 Quanto ao número de atributos envolvidos

Freqüentemente o termo *dimensão* é usado para referenciar um atributo envolvido na regra de associação. Esse termo originou-se do uso de *data warehouses* como repositórios de dados para a mineração de dados.

Quanto ao número de atributos envolvidos, as regras de associação podem ser classificadas em [Han and Kamber, 2001]:

- **Regras de Associação Unidimensionais:** onde os dados envolvidos são provenientes de um único atributo.
- **Regras de Associação Multidimensionais:** onde os dados envolvidos são provenientes de múltiplos atributos.

Para exemplificar, considere as regras apresentadas na **Figura 4.1**. A regra 1 envolve somente o atributo *item_comprado*, portanto ela pode ser classificada como unidimensional. Já a regra 2 envolve três atributos distintos: *idade*, *renda* e *item_comprado*, sendo classificada como regra de associação multidimensional.

Regra 1) *item_comprado= "fralda", item_comprado= "leite" => item_comprado= "cerveja"*

Regra 2) *idade= [20..30], renda= "1000..3000" => item_comprado= "farinha"*

Figura 4.1: *Representação de regras de associação*

4.3.3 Quanto ao nível de detalhamento envolvido

Os itens da base de dados podem ser agrupados em diferentes níveis de abstração. As regras de associação podem ser classificadas de acordo com o nível de abstração de seus itens em:

- **Regras de associação *multiníveis:*** são regras que envolvem itens em diferentes níveis de abstração.
- **Regras de associação *uninível*:** são regras que envolvem itens de apenas um nível de abstração.

Considere os dados de uma loja de equipamentos de informática, com computadores e impressoras de diversos tipos e marcas, conforme mostrado na **Figura 4.2**.

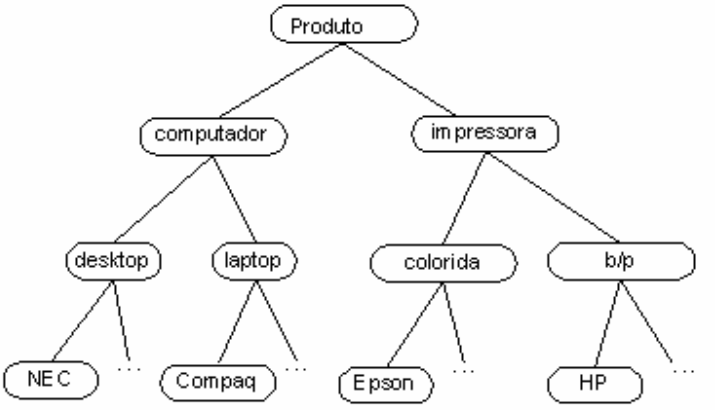


Figura 4.2: Níveis de abstração dos itens impressora e computador

Na hierarquia apresentada na **Figura 4.2**, cada nó carrega consigo as informações dos seus antecedentes. Assim, o nó folha *"NEC"* carrega a informação: "*computador desktop NEC"*. Suponha que, a partir desse esquema as seguintes regras de associação *multiníveis* sejam geradas:

computador → Impressora

computador desktop → impressora b/p

computador desktop NEC → impressora b/p HP

Como as regras de associação *uninível* consideram apenas o nível mais baixo de abstração dos dados, se forem consideradas apenas as regras de associação de único nível teríamos somente a regra: *computador desktop nec → impressora b/p HP*.

4.3.4 Quanto ao relacionamento entre os itens da regra

As regras de associação também podem ser classificadas quanto ao tipo de relacionamento existente entre seus itens. Assim, uma regra de associação $X \rightarrow Y$, onde X e Y são conjuntos de itens, pode ser classificada como:

- **Regra de Associação Direta**: a presença de X aumenta a possibilidade da presença de Y em uma transação.
- **Regra de Associação Inversa ou Negativa**: a presença de X diminui a possibilidade da presença de Y em uma transação.

É possível saber se uma regra de associação é direta ou inversa através do cálculo do coeficiente de correlação da mesma. A medida coeficiente de correlação é apresentada na seção 4.5 deste capítulo.

A **Figura 4.3**, mostra um esquema com os tipos de regras de associação apresentados nesta seção.

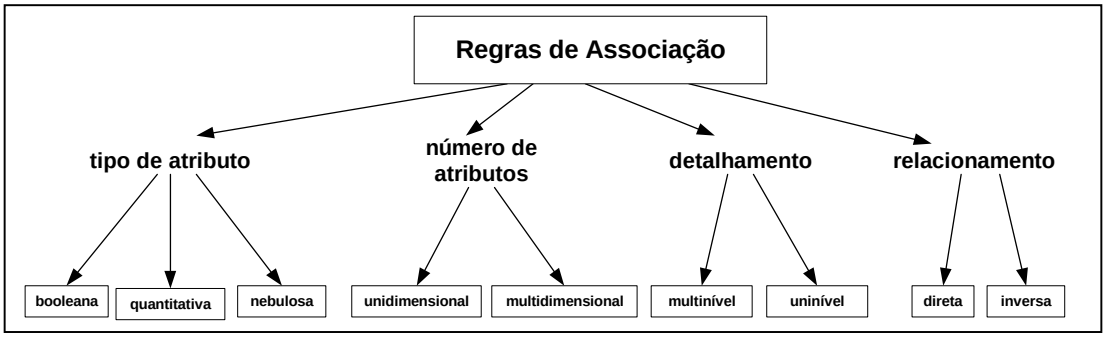


Figura 4.3: *Tipos de Regras de Associação*

Observe que uma regra de associação é classificada de acordo com os quatro parâmetros (tipo de atributo, número de atributos, detalhamento e relacionamento). Por exemplo, considere a seguinte regra:

Fralda→Cerveja

Essa regra é classificada como *booleana* (envolve atributos categóricos), unidimensional (envolve somente um tipo de atributo), *uninível* (envolve somente o nível mais baixo de abstração) e direta (a ocorrência do antecedente aumenta a possibilidade de ocorrência do conseqüente).

Apesar de existirem vários tipos diferentes de regras, a base da mineração desses diferentes tipos provém do problema de mineração de regras de associação *booleanas*.

4.4 Tipos de mineração de Regras de Associação

O processo de mineração pode ser classificado, quanto à origem dos dados e restrições de acesso aos mesmos, nas seguintes categorias: mineração dos dados em uma

única relação, mineração envolvendo múltiplas relações, mineração envolvendo múltiplas bases de dados, mineração distribuída, mineração com preservação de privacidade.

Conceitos gerais sobre essas categorias de mineração são discutidos nesta seção; maiores detalhes, assim como algoritmos empregados para os diferentes tipos de mineração são fornecidos no próximo Capítulo, que é destinado ao detalhamento das técnicas de mineração de dados.

4.4.1 Mineração de Regras de Associação em uma Relação

A mineração envolvendo dados categóricos em uma única relação de uma base de dados é comumente referenciada como mineração de regras de associação tradicional. Diversos trabalhos foram direcionados na literatura para aprimorar esse tipo de mineração [Agrawal et al, 1993, Agrawal and Srikant, 1994, Agrawal and Srikant, 1995, Cheung et al, 1996b, Houtsma and Swami, 1993, Klemettinen et al, 1994, Srikant and Agrawal, 1996, Zaki et al, 1997].

As técnicas de mineração tradicionais são usadas como base para os demais tipos de mineração de regras de associação.

4.4.2 Mineração Multi-relacional de Regras de Associação

A maioria dos métodos tradicionais de mineração de regras de associação requer que os dados sejam transferidos para uma única relação, no entanto, essa transferência implica em perda de informações (de relacionamento dos dados) e possui um custo alto, além de acarretar duplicações dos dados.

Alguns trabalhos na literatura foram propostos para minerar múltiplas relações de uma base de dados dedutiva em conjunto [Deshape and Raedt, 1997, Nijssen and Kok, 2001], onde uma base de conhecimento, contendo múltiplas relações, é explorada a partir de um conjunto de predicados pré-definidos. Uma introdução bastante interessante sobre a mineração envolvendo múltiplas relações de bases de dados dedutivas foi feita por Džeroski [Džeroski, 2003] que, em [Džeroski and Raedt, 2002], enfatizou a necessidade de desenvolvimento de técnicas de mineração envolvendo múltiplas relações para solucionar problemas de mineração envolvendo dados estruturados.

Embora muito tenha se discutido na literatura, na prática nenhum trabalho foi direcionado a minerar regras de associação em bases de dados relacionais, envolvendo transações provenientes de relações distintas.

4.4.3 Mineração Distribuída de Regras de Associação

Tradicionalmente os algoritmos de mineração de dados foram desenvolvidos para serem aplicados em uma base de dados centralizada, onde todos os dados, a serem usados no processo de mineração, se encontram em uma mesma fonte. Entretanto, em muitos casos esses dados podem estar distribuídos em múltiplas fontes.

A mineração de dados envolvendo bases de dados *particionadas*, onde os dados estão distribuídos em múltiplas fontes, tem sido objeto de pesquisa recente [Agrawal and Ramakrishnan, 2000, Agrawal et al, 2003, Evfimievski et al, 2002, Kantarcioglu and Clifton, 2002, Vaidya and Clifton, 2003, Vaidya and Clifton, 2002]. Nesse tipo de mineração, os dados podem estar verticalmente ou horizontalmente particionados. A **Figura 4.4** ilustra as diferenças entre o particionamento vertical e horizontal:

- **Dados Particionados Horizontalmente**: Os dados estão *particionados* horizontalmente quando os conjuntos de transações que formam a base de dados estão distribuídos em diferentes *sites*.
- **Dados Particionados Verticalmente**: Os dados estão *particionados* verticalmente quando cada *site* possui dados referentes a uma parte dos atributos de cada transação.

Os problemas de mineração de regras em bases de dados particionadas verticalmente e horizontalmente diferem entre si, embora ambos derivem do problema de mineração de regras de associação *booleanas*.

O problema de mineração de regras de associação em dados particionados horizontalmente é definido como segue: Seja D uma base de dados particionada em n *sites* $S_1, S_2, ..., S_n$. Para um determinado *itemset* X seja $X.sup$ o suporte de X global (considerando todas as partições) e $X.sup_j$ o suporte de X na partição j. Seja *minsup* o mínimo valor de suporte estabelecido e *minconf* o mínimo valor de confiança estabelecido. Um *itemset* X é localmente freqüente se $X.sup_j \geq minsup$. Um *itemset* X é globalmente freqüente se $X.sup \geq minsup$. A mineração de regras de associação consiste em encontrar

todos os *itemsets* globalmente freqüentes e gerar as regras de associação com confiança acima de *minconf.*

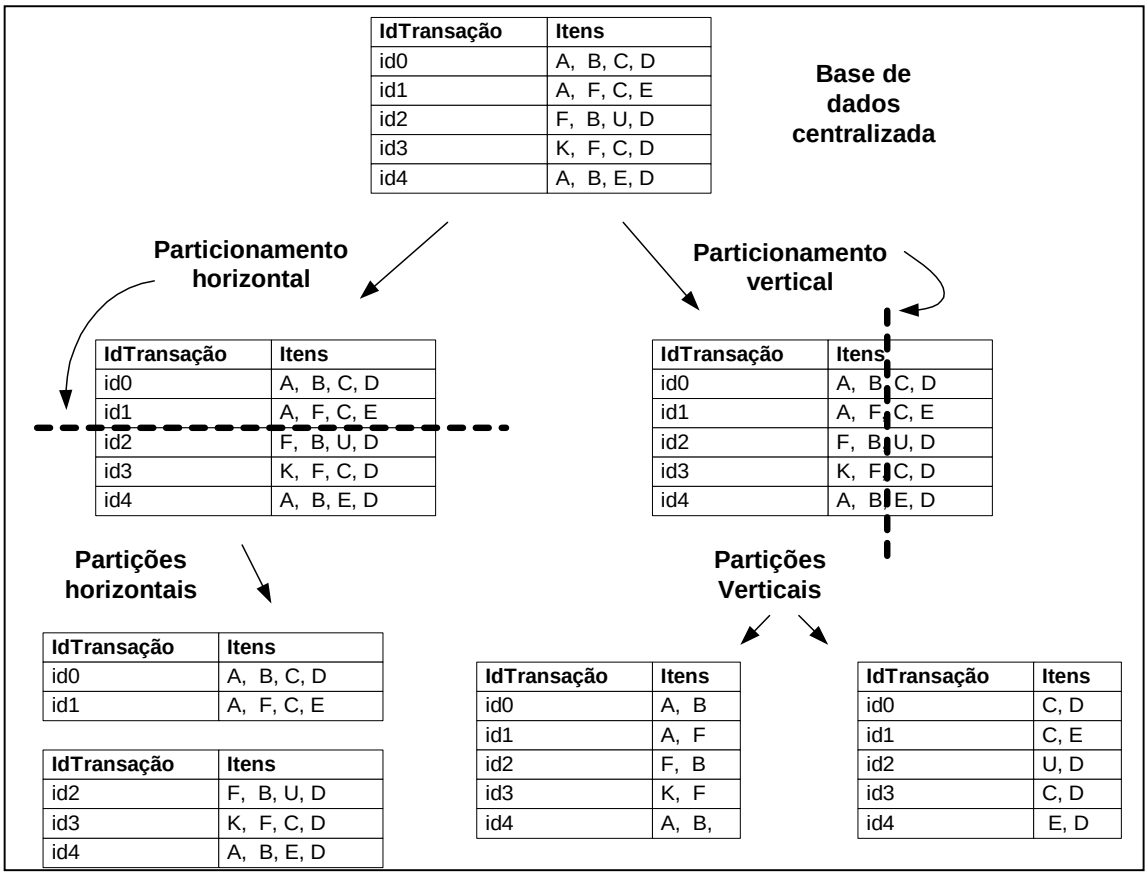


Figura 4.4: *Diferenças entre particionamento vertical e horizontal de uma base de dados.*

O problema de mineração de regras de associação em dados verticalmente *particionados* é semelhante ao problema de minerar regras de associação *Booleanas*, no entanto, o fato dos atributos envolvidos nas regras serem provenientes de diferentes partições faz com que o procedimento de determinação do suporte seja diferente.

Usualmente as fontes de dados envolvidas em base de dados particionadas pertencem a instituições diferentes e, embora essas instituições colaborem para obter o resultado da mineração, elas não desejam revelar seus dados para as demais instituições. Assim, a mineração envolvendo dados *particionados* requer que medidas de *preservação de*

privacidade sejam usadas durante o processo de mineração. O conceito de mineração com preservação de privacidade é apresentado a seguir.

4.4.4 Mineração com Preservação de Privacidade

A computação segura envolvendo múltiplas fontes (*Secure Multiparty Computation*) foi proposta em [Yao, 1986] e sua idéia básica consiste em determinar que o processamento somente é seguro se, no final do mesmo, nada, exceto os valores de entrada e saída, são conhecidos pelas demais entidades. Técnicas de preservação de privacidade permitem que o processamento seja realizado sem que os dados de transações de múltiplas fontes sejam divulgados. Em [Evfimievski et al, 2003] considera-se que a privacidade dos dados é mantida somente se a quantidade de informações conhecidas após o processamento não é maior do que a quantidade de informações conhecidas anteriormente.

As técnicas de preservação de privacidade envolvem métodos de inserção de perturbações nos dados [Agrawal and Ramakrishnan, 2000, Kantarcioglu and Clifton, 2002, Vaidya and Clifton, 2002] e criptografia [Agrawal et al, 2003, Veloso et al, 2003]. Métricas que quantificam a preservação de privacidade foram propostas em [Agrawal and Ramakrishnan, 2000].

4.4.5 Mineração de Regras de Associação Multi-bases (*Multi-Database Mining*)

A mineração de regras entre múltiplas bases de dados é uma importante área de pesquisa que está surgindo em virtude da necessidade de análise de dados provenientes de múltiplas bases. A diferença entre a mineração multi-bases e a mineração distribuída é que na mineração de dados distribuída é assumido que as fontes de dados não tenham conflitos, no entanto, a mineração entre múltiplas bases de dados envolve o problema da inconsistência. Os dados na mineração multi-bases podem ter formatos, nome e tamanho diferentes podendo apresentar conflitos uns com os outros. Assim, muitas vezes, o emprego do pré-processamento se faz necessário. Um dos grandes problemas na mineração multi-bases é que ainda se faz o emprego de técnicas de mineração mono-bases (envolvendo somente uma base) para relacionar dados de múltiplas bases, onde os dados das múltiplas fontes são colocados juntos para análise. No entanto essa migração para uma única fonte pode destruir padrões, informações sobre o relacionamento, estrutura e informações sobre

as proporções nos dados originais, impedindo que os padrões que considerem esses tipos de informações sejam descobertos, além de acarretar duplicação dos dados e padrões inconsistentes.

Em [Zhang et al, 2003] é proposto um processo para mineração de regras de associação em uma companhia com múltiplas filiais sendo que cada companhia tem sua própria base de dados. O procedimento proposto tem duas camadas de processamento:

- **Processamento local:** os padrões locais a cada base de dados são encontrados.
- **Processamento global:** baseados nos padrões locais os padrões globais são determinados.

Baseado na mínima freqüência e na distribuição dos padrões, os padrões encontrados na análise *multi-bases* apresentada em [Zhang et al, 2003] podem ser de três tipos:

- **Padrões de voto alto:** padrões existentes na maioria das bases de dados.
- **Padrões excepcionais:** padrões existentes em apenas uma pequena parcela das bases de dados.
- **Padrões sugeridos:** padrões com suporte menor, porém próximo do suporte mínimo.

A abordagem acima é voltada para um tipo de mineração multi-bases, que é aplicada à descoberta de padrões em múltiplas bases de estruturas semelhantes, pois pertencem à mesma companhia, onde existe a presença de uma entidade central, no caso a matriz, a qual centraliza os dados e decide quais padrões minerados são mais interessantes. No entanto, existem várias outras aplicações da mineração envolvendo múltiplas bases de dados, como por exemplo, uma análise, revelando as tendências de consumo, envolvendo bases de dados de empresas diferentes pode determinar acordos e estratégias de marketing entre as empresas envolvidas. Nesse tipo de análise, medidas que garantam a privacidade dos dados devem ser adotadas, pois na maioria dos casos, as políticas de segurança de cada empresa impedem que uma instituição revele informações de transações de seus clientes para outras instituições.

4.5 Medidas de Interesse

Tipicamente, o número de padrões gerados ao aplicar mineração de dados é grande, e somente alguns desses são realmente de interesse para o domínio esperado. Para aumentar a utilidade, relevância ou mesmo criar um ranking dos padrões descobertos, as *medidas de interesse* são utilizadas. Em [Hilderman and Hamilton, 1999] é apresentada uma coletânea das medidas de interesse mais usadas na literatura e em [Tan et al, 2002] é apresentada uma discussão de como escolher a medida de interesse certa para ser aplicada em um específico tipo de mineração.

As medidas de interesse quantificam o grau de interesse das regras mineradas, através de cálculos estatísticos. As principais medidas de interesse usadas no processo de mineração são as medidas de suporte e confiança já apresentados nesta monografia.

Segundo [Hilderman and Hamilton, 1999] as medidas de interesse diferem quanto ao:

- **Tipo de padrão em que é aplicável**: regras de associação; regras de classificação; hierarquias de classificação; etc.
- **Metodologia usada:** probabilística; sintática; distância; etc.
- **Escopo em que é aplicada:** uma única regra; múltiplas regras.
- **Classe:**
  - **Objetiva:** baseada na estrutura do conhecimento descoberto.
  - **Subjetiva:** é baseada na crença que o usuário (especialista de domínio) possui ou baseada em tendências aparentes do relacionamento entre os dados. Em outras palavras, a análise de interesse é conduzida pelo usuário quando medidas de interesse subjetivas são usadas .

Os fatores mais usados ao definir medidas de interesse em regras de associação $X \rightarrow Y$ são:

- **Cobertura** $|X|$: o número de tuplas que possui o antecedente da regra X.
- **Completude** $|X \cup Y|/|Y|$: a proporção de tuplas contendo Y que também contém X.
- **Confiança** $|X \cup Y|/|X|$: a proporção de tuplas contendo X que também contém Y.

As medidas de interesse podem ser aplicadas como um pré-processamento ou pós-processamento no processo de mineração de dados.

4.5.1 Peculiaridade (Peculiarity)

A medida peculiaridade [Zhong et al, 2001] quantifica o quanto um objeto difere de um outro objeto do mesmo tipo. Um objeto é peculiar se ele representa relativamente uma pequena parcela do conjunto dos dados analisados e se ele é bem diferente dos demais objetos do conjunto de dados. Assim, dados peculiares têm um baixo suporte.

Considere que existam n transações na base de dados e cada transação tenha m atributos A_j. O valor de A_j na transação i é x_{ij}. A peculiaridade de um valor x_{ij} é medida em função da variação dos valores do atributo A_j na base. Assim, o *fator de peculiaridade* é definido por:

$$PF(x_{ij}) = \sum_{j=1}^{n} \sqrt{N(x_{ij}, x_{kj})},$$

$N(x_{ij},x_{kj})$ é a distância conceitual entre os valores x_{ij} e x_{kj}. Essa distância pode ser obtida através de conhecimento prévio do domínio dos dados, ou, se os dados são contínuos, a seguinte fórmula de distância pode ser usada:

$N(x_{ij},x_{kj}) = |x_{ij} - x_{kj}|$

Para verificar se um objeto é peculiar, um teste é realizado na base após os fatores de peculiaridade terem sido calculados:

$T = média\ de\ PF(x_{ij}) + \beta \times desvio\ padrão\ de\ PF(x_{ij})$

Onde β é um parâmetro ajustado pelo usuário, cujo valor padrão é *1*. O dado analisado é peculiar se o seu valor de *PF* é muito maior do que a média dos valores de *PF* dos elementos da base de dados, ou seja, um dado é peculiar se o valor de *PF* for igual ou maior do que um determinado limite *T*. Assim, para um dado x_{ij} ser peculiar, a condição seguinte deve ser satisfeita:

$PF(x_{ij}) \geq T$

Distúrbios nos dados também podem aparecer como dados peculiares, no entanto, nesse tipo de problema considera-se que os dados tenham sido previamente tratados de maneira a terem as perturbações removidas.

4.5.2 Exceções Confiáveis

Qualquer exceção deve ter um baixo suporte, pois, caso contrário, é um padrão encontrado nos dados. Assim, exceções confiáveis [Liu et al, 1999] são regras que possuem relativamente baixo suporte e alta confiança.

Considere que $\neg X$ indica a não ocorrência do *itemset X*. Uma regra de associação $X \rightarrow Y$, minerada com alta confiança, pode ter a seguinte regra de exceção associada a ela:

$X, Z \rightarrow \neg Y$

Segundo a regra acima, *Z* pode ser considerado como uma exceção para a ocorrência da regra $X \rightarrow Y$, isto é, quando *X* ocorre em conjunto com *Z*, o *itemset Y* tende a não ocorrer.

O processo de obtenção de exceções confiáveis envolve uma série de fases, detalhadas em [Liu et al, 1999], sendo que inicialmente são encontradas as regras de interesse e, posteriormente, são verificados se existem exceções à mesma.

4.5.3 Grau de Interesse de Dong e Li

O grau de interesse de Dong e Li [Dong and Li, 1998] determina a importância de uma regra de associação, considerando o quanto ela é não esperada em termos de outras regras de associação em sua vizinhança.

A vizinhança-r de uma regra de associação constitui todas as regras de associação potenciais dentro de um determinado raio *r*. O conjunto de regras vizinhas de uma regra R_0 dentro de um raio *r* é:

$N(R_0,r) = \{R \mid D(R,R_0) \leq r, \text{ onde } R \text{ é uma regra em potencial}\}$

A distância métrica $D(R_1,R_2)$ entre duas regras de associação $R_1: X_1 \rightarrow Y_1$ e $R_2: X_2 \rightarrow Y_2$ é dada por:

$$D(R_1, R_2) = \delta_1 \times |(X_1 \cup Y_1) \Theta (X_2 \cup Y_2)| + \delta_2 \times |X_1 \Theta X_2| + \delta_3 \times |Y_1 \Theta Y_2|$$

Na função de distância, δ_1, δ_2 e δ_3 são parâmetros usados para balancear a importância relativa dos 3 termos usados no cálculo de distância e Θ é o operador de diferença simétrica. A diferença simétrica entre dois *itemsets X* e *Y* é dada por:

$$X \Theta Y = (X - Y) \cup (Y - X)$$

Segundo Dong e Li, existem dois tipos de interesse em uma regra: o interesse baseado na *confiança não-esperada* e o interesse baseado na *confiança isolada*.O seguinte teste permite verificar se uma regra R_0 tem ou não *confiança não-esperada*:

se $|c(R_0) - ac(R_0,r) - sc(R_0,r)| > t_1$ *então* R_0 ***tem*** *confiança não-esperada*
senão R_0 ***não*** *tem confiança não-esperada*

Onde $c(R_0)$, $ac(R_0,r)$, $sc(R_0,r)$ são, respectivamente, a confiança de R_0, a média e o desvio padrão das confianças das regras do conjunto $M \cap N(R_0,r) - \{R_0\}$. M é o conjunto de regras mineradas e t_1 é um parâmetro estabelecido pelo usuário.

A presença ou ausência da *confiança isolada* também é dada por um teste:

se $|N(R_0,r)| - |M \cap N(R_0,r)| > t_2$ então R_0 ***tem*** *confiança isolada*
senão R_0 ***não*** *tem confiança isolada.*

Onde $|N(R_0,r)|$ é o número de regras potenciais na vizinhança-r de R_0, e $|M \cap N(R_0,r)|$, é o número de regras geradas da vizinhança-r e t_2 é um parâmetro estabelecido pelo usuário.

4.5.4 Coeficiente de Correlação

O coeficiente de correlação permite descobrir, em uma regra $X \rightarrow Y$, se o relacionamento entre os conjuntos X e Y é direto, inverso ou não existente [Han and Kamber, 2001]. O coeficiente de correlação entre X e Y, na regra $X \rightarrow Y$, é:

$$corr_{x,y} = \frac{P(X \cup Y)}{P(X)P(Y)}$$

Onde $P(Z)$ é igual ao suporte do conjunto de itens Z, representando a probabilidade de Z ocorrer nas transações da base de dados.

Se corr $_{X,Y}$ é menor que 1, os itens X e Y são inversamente correlacionados. Se corr $_{X,Y}$ é maior do que 1, os itens X e Y estão diretamente correlacionados e se corr $_{X,Y}$ é igual a 1 os itens X e Y não estão correlacionados.

4.5.5 Grau de Interesse de Gray e Orlowska

A medida de interesse de Gray e Orlowska [Gray and Orlowska, 1998] indica a independência entre o antecedente e conseqüente em uma regra de associação. A medida de interesse I para uma regra $X \rightarrow Y$ é definida por:

$$I = \left(\left(\frac{P(X \cap Y)}{P(X) \times P(Y)} \right)^K - 1 \right) \times (P(X) \times P(Y))^n$$

Nesta medida $P(X \cap Y)$ é a confiança, $P(X) \times P(Y)$ é o suporte, $\frac{P(X \cap Y)}{P(X) \times P(Y)}$ é a discriminação e *k* e *m* são parâmetros para balancear os componentes de discriminação e de suporte respectivamente. Maiores valores de *I* conduzem a regras mais interessantes.

Nesta seção, foram apresentadas as principais medidas de interesse usadas na mineração de regras de associação. Freqüentemente, um grande número de regras é gerado ao aplicar a mineração de regras de associação, tornando extremamente árduo o trabalho de analisar e distinguir quais regras são realmente importantes. As medidas de interesse permitem ordenar as regras mineradas e selecioná-las de acordo com o seu grau de interesse, facilitando a análise das regras obtidas.

4.6 Considerações Finais

Este projeto de pesquisa está direcionado ao desenvolvimento de técnicas de mineração de dados envolvendo múltiplas tabelas fato de um *data warehouse*, que é um novo tipo de mineração de regras de associação. Para que seja possível o desenvolvimento desse novo tipo de mineração é preciso conhecer os tipos de mineração já existentes. Assim, neste Capítulo uma visão geral dos tipos de problemas existentes na mineração de regras de associação foi fornecida. Aqui foram apresentados os tipos de regras, os tipos de processo e as principais medidas de interesse envolvidas no processo de mineração de regras de associação. Os diferentes tipos de processo de mineração apresentados neste Capítulo ainda podem ser classificados quanto às estratégias de mineração adotadas. O próximo Capítulo apresenta os principais algoritmos e as principais estratégias usadas na mineração de regras de associação.

5 Algoritmos de Mineração de Regras de Associação

5.1 Introdução

Na mineração de regras de associação tradicionais, o problema de minerar regras consiste em duas etapas: primeiro, encontrar todos os *itemsets* freqüentes; segundo, gerar todas as regras fortes. A determinação dos *itemsets* freqüentes, por sua vez, também é feita em duas etapas: primeiro, o conjunto de *itemsets* candidatos é gerado; segundo, a base de dados é varrida, o suporte dos *itemsets* candidatos é determinado e o conjunto de *itemsets* freqüentes é encontrado.

Os primeiros algoritmos de mineração de dados foram o AIS [Agrawal et al, 1993] e o SETM [Houtsma and Swami, 1993]. Em [Agrawal and Srikant, 1994] foi apresentado o algoritmo *Apriori,* que hoje é o algoritmo mais conhecido de mineração de regras de associação. O algoritmo *Apriori* tem uma grande vantagem de eficiência em relação aos seus algoritmos precedentes. No entanto, esse algoritmo faz inúmeras varreduras na base de dados, o que limita o seu desempenho.

A fase da mineração que exige maior processamento é a determinação dos *itemsets* freqüentes. Novos algoritmos foram desenvolvidos para tornar essa fase mais eficiente, dentre eles destacam-se os algoritmos *Partition* [Savarese et al, 1995], *FP-Growth* [Han et al, 2000] e *Eclat* [Zaki et al, 1997]. Esses algoritmos, que são estudados neste Capítulo, usam diferentes abordagens para tornar a determinação dos *itemsets* freqüentes mais eficiente.

Os algoritmos *Apriori*, *Partition*, *Eclat* e *FP-Growth*, assim como vários outros encontrados na literatura, estão direcionados a resolver o problema tradicional de mineração de regras de associação. Quando outros tipos de dados estão envolvidos, como por exemplo, dados quantitativos, ou mesmo, quando a origem dos dados passa a não ser apenas uma tabela, mas um conjunto de tabelas, ou um conjunto de fontes geograficamente distribuídas, outros procedimentos de mineração devem ser usados para que seja possível extrair as regras de associação.

Técnicas de mineração envolvendo dados quantitativos foram apresentadas em [Hong et al, 1999, Srikant and Agrawal, 1996]. Em [Zhang et al, 2003, Zhong et al, 2001] foi tratado o problema de analisar múltiplas bases de dados em conjunto. Aspectos relativos à mineração de regras de associação provenientes de dados distribuídos foram tratados em [Cheung et al, 1996b, Kantarcioglu and Clifton, 2002, Otey et al, 2003]. Em [Datcu and Seidel, 2000, Hsu et al, 2002, Hsu et al, 2003, Ordonez and Omiecinski, 1999, Zaïane et al, 1998] foram tratados problemas relativos a mineração de regras de associação envolvendo dados multimídia.

Este Capítulo visa apresentar as principais técnicas e algoritmos usados para a mineração de regras de associação. Uma atenção especial é fornecida aos algoritmos usados para a mineração tradicional de regras de associação, uma vez que eles representam a base para o entendimento e o desenvolvimento de algoritmos para outros tipos de mineração de regras de associação. Com o objetivo de entender como as técnicas tradicionais de mineração podem ser estendidas para a mineração não tradicional, neste Capítulo também serão apresentados os algoritmos de mineração distribuída e de mineração envolvendo bases de dados dedutivas.

Este Capítulo está organizado como segue. Na seção 5.2 são apresentados os conceitos iniciais de mineração de regras de associação. Na seção 5.3 são apresentadas as estratégias de determinação de *itemsets* freqüentes usadas nos algoritmos de mineração de regras de associação tradicionais. Nas seções 5.4, 5.5, 5.6, 5.7 e 5.8 são apresentados, respectivamente, os algoritmos *Apriori*, *AprioriTID*, *Partition*, *FP-Growth* e *Eclat* usados para a mineração tradicional de regras de associação. Na seção 5.9 é apresentado o algoritmo *FDM* que é usado para a mineração de dados distribuída envolvendo bases de dados particionadas horizontalmente. Na seção 5.10 é apresentado o algoritmo de *Vaidya e Clifton*, também usado para a mineração de dados distribuída, porém envolvendo bases de dados particionadas verticalmente. Na seção 5.11 é apresentado o algoritmo *Warmr*, usado para minerar dados provenientes de múltiplas relações de uma base de dados dedutiva. Finalmente, na seção 5.12 são feitas as considerações finais deste Capítulo.

5.2 Conceitos Básicos

Conforme discutido no Capítulo anterior, uma grande variedade de algoritmos de mineração de regras de associação tem sido desenvolvida ao longo dos últimos anos. Os algoritmos de mineração se diferenciam quanto a:

- Distribuição dos dados envolvidos.
- Tipos dos dados envolvidos.
- Tipo de abordagem utilizada para encontrar os *itemsets* freqüentes.

De uma maneira geral, as técnicas de mineração não tradicionais derivam das técnicas de mineração tradicionais, que são as técnicas destinadas à mineração de dados categóricos processadas em uma única tabela.

O **problema de minerar regras de associação**, conforme discutido no Capítulo anterior, consiste em encontrar todas as regras de associação fortes (regras que satisfazem o suporte mínimo e a confiança mínima) em uma base de dados. A restrição de minerar regras que satisfazem os valores estabelecidos de suporte mínimo e confiança mínima permite dividir o problema de mineração em duas etapas [Agrawal and Srikant, 1994]:

1. Encontrar $L = \{ X \subseteq I \mid X$ é freqüente $\}$, onde I é o conjunto de todos os itens da base de dados analisada. L é o conjunto de todos os *itemsets freqüentes*, juntamente com seus respectivos valores de suporte.
2. Para todos os *itemsets* freqüentes $X \in L$, calcular a confiança de todas as regras $Y \rightarrow X\text{-}Y$, onde $Y \subset X$ e $Y \neq \phi$, e eliminar todas aquelas que não satisfazem a confiança mínima.

Vários algoritmos existentes, como por exemplo, os algoritmos *Apriori* [Agrawal and Srikant, 1994] e *Partition* [Savarese et al, 1995], determinam primeiramente o conjunto de *itemsets* candidatos[1] e, após isso, determinam os *itemsets* freqüentes.

A fase crítica da mineração é a fase de determinação dos *itemsets* freqüentes. Segundo Hipp; Güntzer e Nakhaeizadeh [Hipp et al, 2000], o problema de encontrar regras de associação pode ser reduzido a encontrar todos os *itemsets* freqüentes e seus respectivos

[1] ***Itemsets* candidados** são *itemsets* cujo suporte ainda não foi determinado. Um *itemset* candidato pode ou não se tornar um *itemset* freqüente após a determinação do seu suporte.

valores de suporte, uma vez que, tendo encontrado os *itemsets* freqüentes, para determinar as regras, basta gerar as combinações internas de cada *itemset* freqüente e calcular a confiança de cada combinação, descartando aquelas combinações que não satisfazem a confiança mínima estabelecida.

Em geral, a fase de geração das regras a partir do conjunto de *itemsets* freqüentes é comum para a maioria dos algoritmos. Agrawal e Srikant fornecem em [Agrawal and Srikant, 1994] um algoritmo simples, ilustrado na **Figura 5.1**, para a geração das regras de associação a partir de um conjunto *L* de *itemsets* freqüentes.

```
1.  Para cada itemset X ∈ L faça
2.     Para todos os subconjuntos Y ∈ X
3.        Se Y→(X-Y) satisfaz a confiança mínima
4.            Adicione Y→(X-Y) ao conjunto de saída C
5.        Fim se
6.     Fim para
7.  Fim Para
```

Figura 5.1: *Algoritmo para a geração de regras de associação* [Agrawal and Srikant, 1994]

A fase de determinação de *itemsets* freqüentes varia de acordo com a estratégia adotada. As principais estratégias usadas para a determinação de *itemsets* freqüentes são discutidas na próxima seção.

5.3 Métodos de Determinação dos Itemsets Freqüentes

Conforme já discutido, o problema de minerar regras de associação pode ser resumido em achar todos os *itemsets* freqüentes (aqueles que satisfazem o suporte mínimo). Uma alternativa, para achar todos os *itemsets* freqüentes, seria determinar a freqüência de ocorrência de todos os subconjuntos de itens que aparecem na base de dados. Na prática, determinar a freqüência de ocorrência de todos os subconjuntos de itens da base de dados é inviável devido ao grande espaço de busca a ser considerado.

Para delimitar o espaço de busca a ser considerado na determinação dos *itemsets* freqüentes, a seguinte propriedade é usada:

Propriedade 5.1: *Se um itemset é freqüente, então todas as combinações de seus elementos também são freqüentes.*

Para exemplificar o uso da propriedade acima, considere o espaço de busca delimitado pelo contorno da **Figura 5.2**. Os *itemsets* freqüentes, encontrados a partir da base de dados apresentada na **Tabela 5.1** (considerando o suporte mínimo igual a 40%), estão localizados na parte superior do contorno e os *itemsets* não freqüentes estão na parte inferior. A existência do contorno separando *itemsets* freqüentes dos não freqüentes é garantida pela *propriedade 5.1*. Essa propriedade implica que no mínimo dois *itemsets* freqüentes de um nível são necessários para formar um *itemset* freqüente de nível mais baixo. Assim, se em um nível não existem pelo menos dois *itemsets* distintos, então esse nível delimita a borda e, portanto, nenhum nível mais baixo do que ele pode conter um *itemset* freqüente.

idTransação	**Itens**
T1	1,2,4
T2	1,2,3
T3	1,3,4
T4	1,4
T5	1,2,4

Tabela 5.1: *Base de dados fictícia para I = {1,2,3,4}*

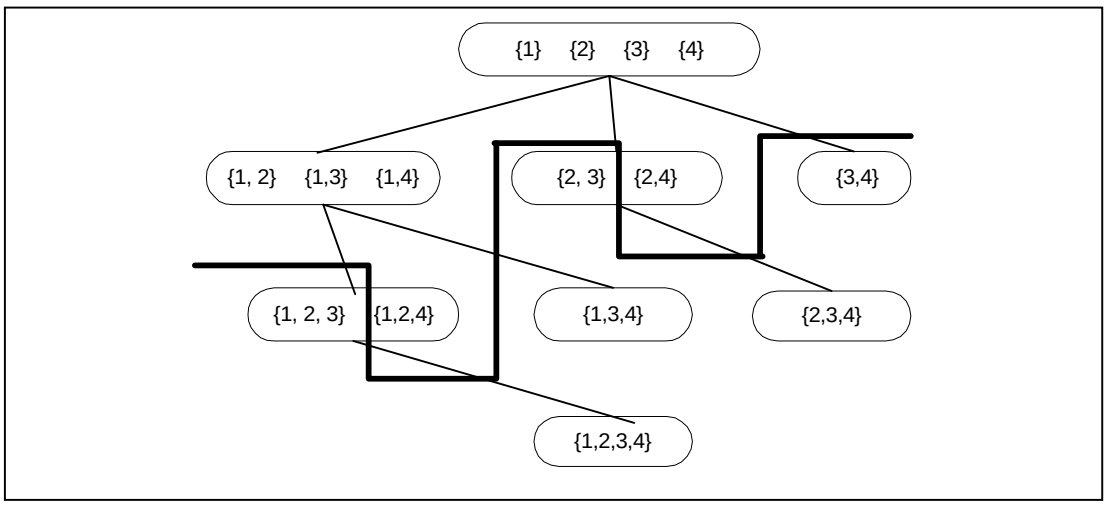


Figura 5.2: *Exemplo de espaço de busca*

O espaço de busca é percorrido usando uma das seguintes estratégias: **busca em profundidade** ou **busca em largura**:

- *Busca em profundidade*: consiste em primeiro determinar o suporte dos nós na profundidade. Um exemplo de busca em profundidade é apresentado na **Figura 5.3**, onde as setas indicam os primeiros *itemsets*, do espaço de busca da **Figura 5.2**, que terão seus valores de suporte determinados.
- Na *busca em largura* os valores de suporte de todos os *itemsets* de tamanho *k-1* são determinados antes do cálculo do suporte dos *itemsets* de tamanho *k.*

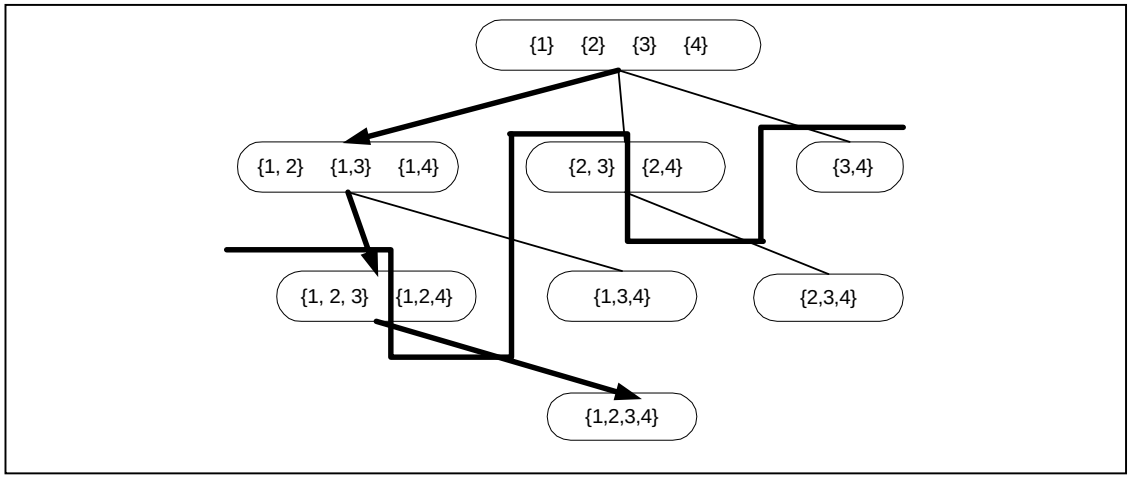


Figura 5.3: *Busca em profundidade no espaço de busca da* ***Figura 5.2***

Existem dois modos usados determinar o suporte de um *itemset*: o modo de **contagem direta do suporte** e o modo de **intersecção de *tidlist***:

- *Contagem direta do suporte*: consiste em diretamente contar as ocorrências dos *itemsets* na base de dados. Para isso, um contador é inicializado com 0 para cada *itemset* candidato e, então, todas as transações são percorridas. O contador relativo a um *itemset* é incrementado sempre que esse *itemset* for reconhecido como um subconjunto de uma transação da base de dados.
- *Intersecção de tidlist*: consiste em interseccionar as listas de identificadores das transações que contém subconjuntos de elementos do *itemset* cujo suporte será determinado. Uma *tidlist* de um *itemset* candidato *Y* (*Y.tidlist*) é um conjunto de

identificadores das transações que contém o *itemset Y*. A *tidlist* de um *itemset* $X=Y \cup Z$ é dada por: $X.tidlist = Y.tidlist \cap Z.tidlist$. O suporte do *itemset* candidato *X* é dado por $|X.tidlist(X)|$[1].

Os algoritmos encontrados na literatura usam uma combinação de estratégias para a delimitação do espaço de busca e determinação do suporte. A seguir são apresentados exemplos de algoritmos e as combinações de estratégias que os mesmos usam:

- *Busca em largura e contagem direta do suporte:* Pode-se citar, como exemplos de algoritmos que usam *busca em largura e contagem direta do suporte* os algoritmos: *AIS* [Agrawal et al, 1993], *SETM* [Houtsma and Swami, 1993], *Apriori*, *AprioriTID* e *AprioriHibrid* [Agrawal and Srikant, 1994]. A parte mais custosa desses algoritmos é a determinação do suporte dos *itemsets*, onde uma estrutura chamada *Hash-Tree* (apresentada na seção 5.3) é usada.
- *Busca em largura e intersecção de tidlist:* No caso do uso da *busca em largura e interseção de tidlist*, o grande volume de memória necessária para armazenar as *tidlist* de todos os *itemsets* do mesmo nível torna essa combinação de estratégias inviável. No entanto, técnicas que reduzem a quantidade de dados armazenados na memória têm sido propostas. Por exemplo, em [Savarese et al, 1995] foi apresentado o algoritmo *Partition* que divide a base de dados para o processo de mineração. O algoritmo *Partition* é detalhado na seção 5.6.
- *Busca em profundidade e contagem direta do suporte:* Em [Han et al, 2000] foi apresentado o algoritmo *FP-Growth* (discutido na seção 5.7) que usa uma estrutura compacta, chamada *FP-tree*, para representar as transações da base. A *FP-tree* é construída usando *busca em profundidade* e *contagem direta do suporte*.

5.3.1 *Hash-Tree* (Árvore *Hash*)

Uma estrutura de dados comumente usada para facilitar o acesso a um *itemset* na determinação de seu suporte é a *Hash-Tree*. Os nós folhas da *Hash-tree* contêm os *itemsets* e os nós internos são tabelas *Hash*. O nó raiz tem profundidade 1. Para adicionar um *itemset*, a árvore é percorrida a partir da raiz onde, estando numa profundidade *d*, o

[1] A notação $|X|$ está sendo usada para representar o número de elementos do conjunto *X*.

próximo ramo a seguir é determinado aplicando uma função *Hash* sobre o elemento de posição *d* do *itemset*. O nó folha é convertido para um nó interior quando *itemsets* são adicionados a ele, excedendo sua capacidade.

Exemplo: A **Figura 5.4** exibe uma *Hash-tree*, que usa a função *Hash* também exibida na **Figura 5.4**, construída a partir do seguinte conjunto de *itemsets* candidatos: C = {{1,4,5}, {1,2,5}, {1,3,6} {5,6,7}, {3,6,7}, {4,5,8}, {1,5,9}, {2,3,4}}, onde cada nó folha tem capacidade para armazenar no máximo *2* itemsets, de modo que, quando um terceiro elemento é atribuído a um nó folha ele "se transforma" em uma tabela *Hash.*

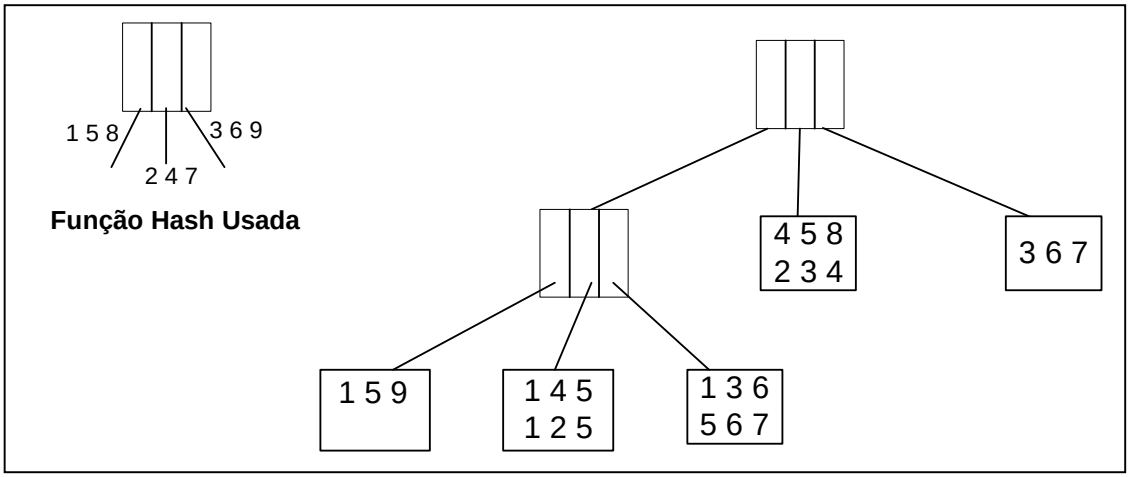


Figura 5.4: *Exemplo de Hash-Tree*

A **Figura 5.5** apresenta um exemplo de adição de um *itemset* na *Hash-Tree* da **Figura 5.4**.

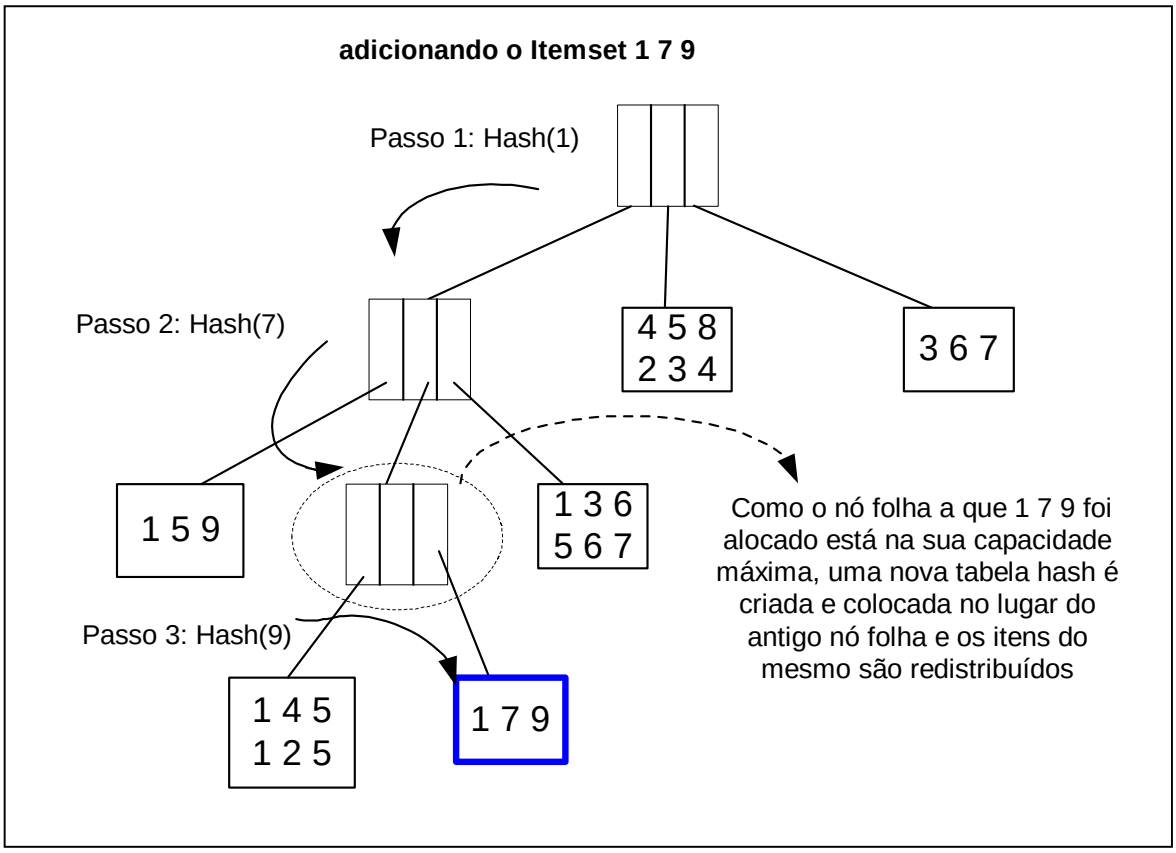


Figura 5.5: *Exemplo de Adição do itemset {1,7,9} na árvore Hash.*

A *Hash-tree* permite acessar rapidamente o *itemset* candidato para o incremento de seu suporte durante a leitura de uma transação, acelerando o processo de determinação de *itemsets* freqüentes.

Nas próximas seções são apresentados alguns algoritmos de mineração de regras de associação que usam as seguintes representações de conjunto (onde o índice *k* indica que o conjunto possui apenas *itemsets* de tamanho k):

- L_k: conjunto dos *itemsets-k* freqüentes, cada membro tem dois campos: um para armazenar o *itemset* e o outro para armazenar o contador de suporte.
- C_k: o conjunto dos *itemsets-k* candidatos, ou seja, é o conjunto dos *itemsets-k* potencialmente freqüentes. Assim como L_k, os elementos de C_k têm dois campos: um para armazenar o *itemset* e o outro para armazenar o contador de suporte.

5.4 Algoritmo Apriori

O algoritmo *Apriori* [Agrawal and Srikant, 1994], devido a sua simplicidade, é um dos algoritmos mais conhecidos de mineração de regras de associação. Ele atende ao problema de minerar regras de associação envolvendo itens categóricos em uma base de dados[1]. O algoritmo *Apriori* usa as estratégias de busca em largura e contagem direta de suporte para determinar o conjunto de *itemsets* freqüentes, realizando múltiplos passos iterativos, onde *itemsets-k-1* são usados para explorar os *itemsets-k*. O algoritmo *Apriori* é apresentado na **Figura 5.6**.

```
Algoritmo: Apriori
Entrada: A base de dados D, o valor de suporte mínimo minsup
Saída: conjunto L com todos os itemsets freqüentes
função Apriori-Alg(D, minsup){
1.      L1=itemsets freqüentes-1 de D;
2.      para(k = 2;Lk-1≠Φ;k++){
3.          Ck =Apriori-gen(Lk-1) ;
4.          para cada transação t ∈ D {
5.              para cada candidato c ∈ Ck, contido em t, faça
6.              c.contador++}}
7.          Lk = { c ∈ Ck | c.contador >= minsup }}
8.      retorne L = ∪k Lk}

função Apriori_gen(Lk-1){
9.    para cada itemset l1 ∈ Lk-1 {
10.       para cada itemset l2 ∈ Lk-1 {
11.           se( l1[1] = l2[1]  e  l1[2] = l2[2]  e ... e l1[k-2] = l2[k-2] e (l1[k-1] <  l2[k-1] ) ){
12.               c = l1[1] · l1[2] ···· l1[k–1] · l2[k–1]
13.               se has_infrequent_subset(c,Lk-1) então
14.                   apague c
15.               senão adicione c em Ck
16.               }
17.   retorne Ck}

função has_infrequent_subset(c, Lk-1)
18.   para cada subconjunto s  de k-1 elementos de c {
19.       se s ∉ Lk-1{
20.           retorne verdadeiro }}
21.   retorne falso
```

Figura 5.6: *Algoritmo Apriori* [Agrawal and Srikant, 1994]

[1] Os algoritmos freqüentemente trabalham sobre uma tabela preparada a partir da base de dados. Essa tabela é chamada de base de dados da análise. Na descrição dos algoritmos das próximas seções, usam-se os termos base de dados, tabela ou relação para fazer referência a essa tabela preparada para a análise.

Na **linha 1**, o algoritmo conta o número de ocorrências de cada item e determina L_1 (conjunto de *itemsets-1* freqüentes*)*. As **linhas de 2 a 7** consistem em determinar L_k (conjunto de *itemsets-k* freqüentes). Na **linha 3**, o conjunto L_{k-1}, encontrado no passo anterior, é usado para gerar C_k (o conjunto de *itemsets-k* candidatos) usando a função *Apriori-gen()*. Nas **linhas 4 a 6** é feita a contagem de cada *itemset-k* candidato, onde seu contador é incrementado de 1 para cada transação em que ele aparece (**linha 6**). Por último, somente os *itemsets-k* candidatos que têm suporte maior ou igual ao suporte mínimo são adicionados a L_k (**linha 7**).

A função *Apriori-gen( )* faz junção de L_{k-1} consigo mesmo, sendo que a condição de junção é que os *k-2* itens dos dados de junção sejam os mesmos (**linha 11**). Na **linha 13**, *Apriori-gen()* chama a função *has_infrequent_subset*, para verificar se o *itemset* gerado tem elementos não freqüentes; caso tenha o *itemset* gerado é apagado (**linha 14**), caso contrário ele é adicionado a C_k (**linha 15**).

A função *has_infrequent_subset* verifica se os *itemsets* resultantes da operação *join* têm algum subconjunto com *k-1* elementos que não faz parte dos L_{k-1}; caso tenha, essa função retorna verdadeiro, caso contrário retorna falso (**linhas 18 a 21**). Assim, é garantido que todos os subconjuntos não vazios de um *itemset* freqüente também são freqüentes.

A **Figura 5.7** mostra um exemplo de execução do *Apriori* sobre a base de dados *D*, considerando um suporte mínimo de 50%.

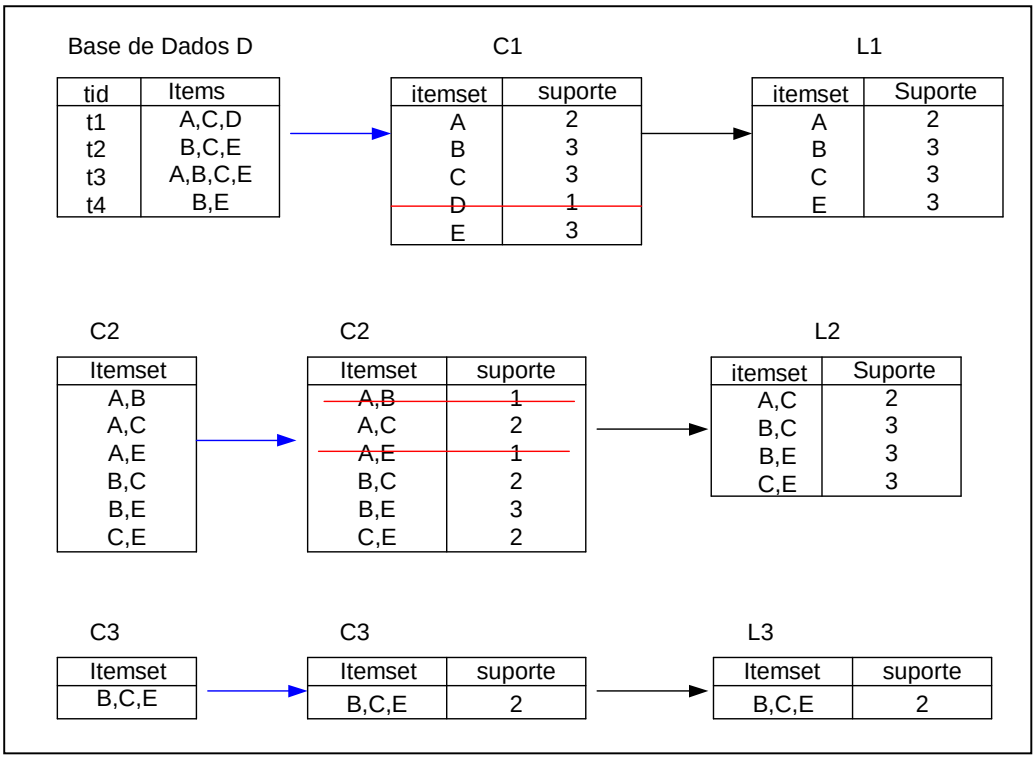


***Figura 5.7**: Exemplo de execução do algoritmo Apriori*

5.5 Algoritmo AprioriTID

O algoritmo *AprioriTID* [Agrawal and Srikant, 1994] também está voltado para a mineração de regras de associação *booleanas* envolvendo uma única tabela e usa as mesmas estratégias de determinação de suporte e confiança que o *Apriori* usa (busca em largura e contagem direta do suporte). A diferença é que no *AprioriTID*, uma estrutura de dados representando a base de dados é carregada na memória, de modo que, para $k \geq 2$, a base de dados não é mais acessada para determinar o suporte dos *itemsets-k* candidatos.

AprioriTID usa a mesma função *Apriori-gen* do *Apriori* para determinar os *itemsets-k* candidatos. A base de dados é percorrida uma vez, e após isso, o conjunto $\overline{C_k}$ é usado para a determinação do suporte. Cada membro de $\overline{C_k}$ é da forma:

$$< \text{TID}, X_k >$$

Onde X_k é um conjunto de *itemsets* potencialmente freqüentes presente na transação identificada por *TID*. Para $K=1$, $\overline{C_k}$ corresponde à base de dados *D*.

O algoritmo *AprioriTID* é apresentado na **Figura 5.8**. L_1 é inicializado com o conjunto de *itemsets-1* freqüentes (**linha 1**) e $\overline{C_1}$ é inicializado com a base de dados *D*. Os *itemsets-k* freqüentes, são determinados nas **linhas 3 a 13**. Os *itemsets-k* candidatos são determinados através da função *Apriori-gen* (**linha 4**). $\overline{C_k}$, para $k \geq 2$, é inicializado com vazio (**linha 5**). Os *itemsets-k* candidatos são procurados em cada transação *t* armazenada em $\overline{C_{k-1}}$, aqueles encontrados em *t* são armazenados em um conjunto temporário C_t (**linha 7**). O contador de um *itemset* candidato é incrementado toda a vez que ele é encontrado em C_t (**linha 9**). Se C_t for diferente de vazio (**linha 10**), ele, junto com o identificador da transação onde foi encontrado, são adicionados ao conjunto $\overline{C_k}$ (**linha 11**). Os *itemsets* pertencentes a C_k que satisfazem o suporte mínimo são adicionados a L_k (**linha 12**).

Algoritmo: AprioriTID
Entrada: A base de dados D, o valor de suporte mínimo *minsup*
Saída: conjunto L com todos os *itemsets* freqüentes

Função *AprioriTID*(D, minsup,minconf)**{**
01 L_1 = conjunto de *itemsets-1* freqüentes
02 $\overline{C_1}$ = base de dados D
03 **para** (K = 2; $L_{K-1} \neq \phi$; K ++)**{**
04 C_K = *Apriori-gen*(L_{K-1})
05 $\overline{C_k} = \phi$
06 **para cada** transação t Є $\overline{C_{k-1}}$ **faça {**
07 C_t = { c Є C_k | c Є t.conjunto_de_*itemsets*}
08 **para cada** candidato c Є C_t faça
09 c.contador ++
10 **se** $C_t \neq \phi$ **então**
11 $\overline{C_k}$ = $\overline{C_k}$ + <t.TID, C_t> **}**
12 L_K = { c Є C_k | c.contador >= min_sup } **}**
13 **retorne** L = $\cup_K L_K$**}**

Figura 5.8*: Algoritmo AprioriTID* [Agrawal and Srikant, 1994]

Um algoritmo híbrido, chamado *AprioriHibrid*, foi proposto em [Agrawal and Srikant, 1994], onde o *Apriori* é usado para os passos iniciais e, para os passos posteriores,

o *AprioriTID* é usado. A estratégia do algoritmo *AprioriHibrid* consiste em evitar a sobrecarga do algoritmo *AprioriTID* nos passos iniciais, pois, $\overline{C_k}$ nos passos iniciais é tão extenso quanto a base de dados.

5.6 Algoritmo Partition

O algoritmo *Partition* [Savarese et al, 1995] é semelhante ao *Apriori*, no que se trata de determinar o suporte de todos os *itemsets-k-1* antes de determinar o suporte dos *itemsets-k*. Porém, diferentemente do *Apriori*, o *Partition* utiliza a interseção das *tidlists* dos *itemsets-k-1* freqüentes para determinar o suporte dos *itemsets-k* candidatos. Como o tamanho das *tidlists* pode ultrapassar a capacidade da memória, o *Partition* divide a base de dados em fatias tratadas independentemente. O tamanho de cada pedaço é escolhido de maneira que todas as suas *tidlists* intermediárias caibam na memória.

Depois de determinar os *itemsets* freqüentes locais de cada partição, uma passagem extra sobre a base de dados é necessária para garantir que os *itemsets* freqüentes localmente também sejam freqüentes globalmente (na base de dados toda). A chave para o funcionamento do algoritmo é que um *itemset* freqüente em toda a base de dados é também freqüente em pelo menos uma das partições.

A **Figura 5.9** apresenta o algoritmo *Partition*, e a **Tabela 5.2** apresenta a notação utilizada.

p_i	*i*-ésima partição da base de dados *D*
L_k^i	Conjunto de *itemsets-k* freqüentes em uma partição p_i
C^G	Conjunto de candidatos globais
L^i	Conjunto de *itemsets* freqüentes locais em uma partição p_i
L^G	Conjunto de *itemsets* freqüentes globais

Tabela 5.2: *Notação usada no algoritmo Partition.*

Inicialmente a base de dados *D* é dividida logicamente em *n* partições (**linha 1**). Para cada partição o conjunto de *itemsets* freqüentes locais é determinado através da função *Partition-Gen* (**linha 3**). O procedimento *Partition-Gen* é semelhante ao *Apriori-gen* usado no algoritmo *Apriori*, inclusive a mesma função *has_infrequent_subset* é usada para verificar se existe um subconjunto não freqüente no *itemset* candidato gerado, para descartá-lo. A diferença entre os algoritmos *Partition-Gen* e *Apriori-Gen* é que o primeiro

determina o suporte através da intersecção de *tidlists* **(linha 19)** e já verifica se o *itemset* candidato gerado é freqüente **(linha 20)**, enquanto o segundo somente adiciona o *itemset* candidato a um conjunto, onde o suporte de seus elementos somente é determinado na próxima varredura da base.

Algoritmo: Partition

Entrada: base de dados *D*, suporte mínimo *minsup*

Saída: conjunto de *itemsets* freqüentes L^G

Função Partition(D, *minsup*) **{**

1. particione a base de dados *D* em *n* partições.
2. **para** cada uma das *n* partições $p_i \in P$
3. L^i =*Partition_Gen*(p_i)
4. $C^G = \cup_{i=1,2,...,n} L^i$;
5. percorra todas as n partições p_i
6. conte o suporte de todos os *itemsets* contidos em C^G
7. $L^G = \{c \in C^G \mid c.count \geq minsup\}$
8. **retorne** L^G **}**

Função *Partition_Gen*(p_i)**{**

9. L_1^i = *itemsets-1* freqüentes de p_i com suas *tidlists*
10. $L^i = \phi$
11. **para**($k=2$; $L_{k-1}^i \neq 0$; k++)**{**
12. **para** cada itemset $l_1 \in L_{k-1}^i$ **{**
13. **para** cada itemset $l_2 \in L_{k-1}^i$ **{**
14. **se**($l_1[1] = l_2[1]$ e $l_1[2] = l_2[2]$ e ... e $l_1[k-2] = l_2[k-2]$ e ($l_1[k-1] < l_2[k-1]$)**{**
15. $c = l_1[1] \cdot l_1[2] \cdots l_1[k-1] \cdot l_2[k-1]$;
16. **se has_infrequent_subset**(c, L_{k-1}^i) **então**
17. apague c
18. **senão**
19. c.tidlist = l_1.tidlist ∩ l_2.tidlist;
20. **se** |c.tidlist| / |D| ≥ minsup **então**
21. $L_k^i = L_k^i \cup \{c\}$
22. **}**
23. **}**
24. **}**
25. $L^i = L^i \cup L_k^i$
26. }
27. **retorne** L^i
28. **}**

Figura 5.9:*Algoritmo Partition*

A união dos *itemsets* freqüentes locais forma o conjunto de candidatos globais C^G (**linha 4**). Em posse de C^G, todas as partições são percorridas e o suporte de todos os elementos de C^G é determinado (**linhas 5 e 6**). Os *itemsets* que satisfazem o suporte mínimo são adicionados ao conjunto de *itemsets* freqüentes globais L^G (**linha 7**) que é retornado pelo algoritmo (**linha 8**).

5.7 Algoritmo FP-Growth

O algoritmo *FP-Growth* (*Frequent Pattern Growth*) [Han et al, 2000] , assim como os algoritmos *Apriori*, *AprioriTID* e *Partition*, também é destinado à mineração de regras de associação *booleanas*. O *FP-Growth* permite achar regras de associação sem gerar o conjunto de *itemsets* candidatos. Nesse método a base de dados é representada por uma estrutura de dados altamente condensada, chamada de *FP-tree* (*Frequent Pattern Tree*).

A *FP-tree* é usada pelo *FP-Growth* para a determinação dos *itemsets* freqüentes. Ela é construída através de uma busca em profundidade e contagem de ocorrências.Um nó *n* da *FP-tree* possui os seguintes campos:

- *item*: armazena o valor do item que ele representa.
- *pai*: ponteiro para o nó pai.
- *filho:* ponteiro para o nó filho.
- *próximo*: ponteiro para o próximo nó, que possui o mesmo valor do campo *item*. Usado para fazer uma lista ligando os nós de mesmo item na *FP-tree*.
- *contador*: armazena o número de ocorrências de seu ramo.

Quando um nó recebe a informação nula, o valor *null* é atribuído a todos os seus campos.

Para facilitar o acesso a um nó da *FP-tree*, uma estrutura auxiliar chamada tabela *Header* é usada junto à *FP-tree*. Cada elemento *h* da tabela *Header* contém os seguintes campos:

- *item:* armazena o valor do item que ele representa.
- *link:* armazena um ponteiro para os nós da árvore com o mesmo valor do campo *item.*

- *contador:* armazena o número de ocorrências do item que ele representa.

A **Figura 5.10** apresenta o algoritmo de construção da *FP-tree*.

```
Algoritmo:FPtree_Gen
Entrada: Base de dados D, suporte mínimo minsup
Saída: FP-Tree T

Função FPTree_Gen (D, minsup) {
1.  percorra a base de dados D
2.  determine o conjunto F de itemsets-1 freqüentes
3.  faça L = F com os itens ordenados na ordem decrescente de seus suportes (caso os
itemsets tenham o mesmo suporte, é usada a ordem lexicográfica)
4.  crie a tabela Header com os elementos de L
5.  crie o nó raiz da FP-tree, atribua-lhe informação nula e chame-o de raiz.
6.  para cada transação t ∈ D {
7.        considere somente itens i∈L de t e ordene-os na ordem de L
8.        chame de p o primeiro elemento de t, e chame de P a lista com os elementos
restantes de t, de maneira que t seja uma lista ordenada representada por [p|P]
9.        insere([p|P], raiz)
10.   }
11. retorne T;
12. }

Procedimento insere([p|P], T){
13.     se T tem um filho n tal que n.item= p. item
14.         n.contador++
15.     senão{
16.         n= novo nó(p.item, 1) // n.item= p.item;  n.contador = 1
17.         n.pai =T
18.         seja h o elemento da tabela Header, tal que h.item =p.item {
19.             n.próximo = h.link
20.             h.link = n }
21.     }
22.     se P≠φ {
23.         T=n;
24.         insere(P,T);
25.     }
26. }
```

Figura 5.10:*Função FPTree-Gen() usada para a construção da FP-tree*

Inicialmente, é feita uma varredura na base de dados para determinar o conjunto *L* de *itemsets-1* freqüentes (**linhas 1 e 2**). A seguir o conjunto *L* é ordenado em ordem decrescente de suporte dos seus itens (**linhas 3**). A *FP-tree* é inicializada com um único nó,

com informação nula **(linha 5)**. Para cada transação, com seus itens freqüentes ordenados na mesma ordem de *L* (**linhas 6 e 7**), é chamado o procedimento *insere* (**linhas 8 e 9**).

O procedimento *insere* verifica se já existe o primeiro item *p* da transação na árvore (**linha 13**). Se existe, o contador desse item será incrementado de 1, senão, um novo nó é criado e adicionado à árvore (**linhas 15 a 21**); os ponteiros do novo nó criado e da tabela *Header* são então atualizados (**linhas 18 a 20**). Enquanto existirem elementos *P*, não adicionados a *FP-tree*, o ponteiro *T* é atualizado para receber o valor do nó atual *n* (**linha 23**) e o procedimento *insere* é chamado recursivamente (**linha 24**).

Com base nos dados da **Tabela 5.3**, o próximo exemplo ilustra a construção da *FP-tree* através do algoritmo *FPtree-Gen.*

TID	ITENS
1	P1, P2, P5
2	P2, P4
3	P2, P3
4	P1, P2, P4
5	P1, P3
6	P2, P3
7	P1,P3
8	P1,P2,P3,P5
9	P1,P2,P3

Tabela 5.3: *Exemplo de base de dados*

O conjunto de *itemsets-1 freqüentes* é ordenado de acordo com a freqüência de ocorrência de seus itens. Considerando o suporte mínimo como 2 ocorrências, *L = [P2:7, P1:6, P3:6, P4:2, P5:2].* A tabela *Header* é criada com os itens de *L*. Então, para construir a *FP-tree*, inicialmente o nó raiz, com a informação nula, é criado. Novamente a base de dados é percorrida e os itens de cada transação são ordenados e processados. A primeira transação, seguindo a ordem de *L*, tem o seguinte conjunto de itens *{P2, P1, P5}*. Assim, tem-se a seqüência de execução: inicialmente *P2* é colocado como filho do nó *raiz*, depois *P1* é colocado como filho de *P2* e por fim, *P5* é colocado como filho de *P1* (ver **Figura 5.11**). Os contadores de cada um dos nós criados possuem, nesse momento, o valor 1.

A segunda transação tem o conjunto de itens *{P2, P4}* (na ordem de *L*); como existe um prefixo na árvore *{null, P2}*, esse prefixo é aproveitado e o contador de *P2* é

incrementado em uma unidade (passa a valer 2). A seguir, o nó *P4* é criado como filho de *P2*.

A **Figura 5.11** mostra a *FP-tree* construída a partir dos dados da **Tabela 5.3**.

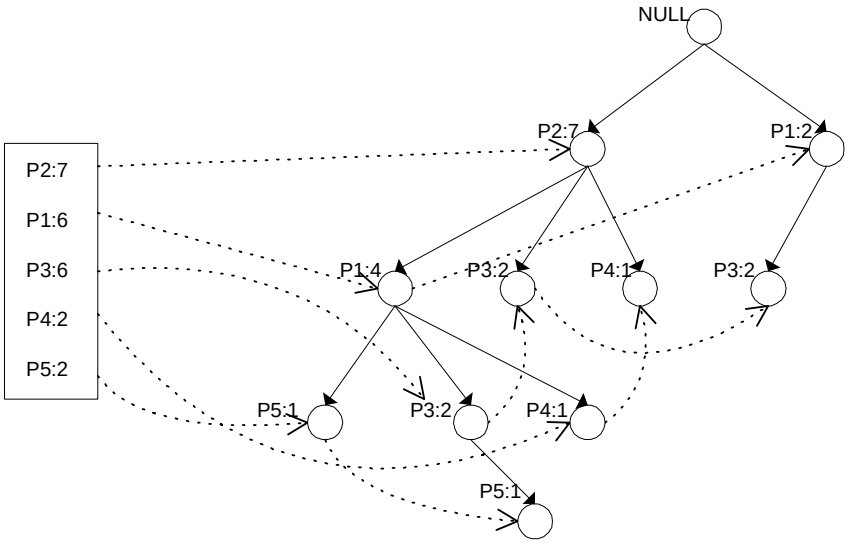


Figura 5.11: FP-tree gerada a partir dos dados da ***Tabela 5.3***

O algoritmo *FP-Growth*, apresentado na **Figura 5.12**, é chamado para a determinação dos *itemsets* freqüentes, (passando como parâmetros a árvore *FP-tree* construída e o valor *null*)

Algoritmo:*FP-Growth*
Entrada: *FP-tree tree, itemset α:*
Saída: Conjunto de *itemsets* freqüentes

Função FP-growth(tree: FP-tree, α: *itemset*) **{**
1. **se** *tree* contém apenas um caminho *P* **então**
2. **para** cada combinação β dos itens dos nós do caminho *P*
3. **gere** o padrão $\beta \cup \alpha$ com suporte = suporte mínimo dos nós em β**}**
4. **senão**
5. **para** cada elemento *h* da tabela de *Header* (iniciando do elemento com menor valor de contador**{**
6. **gere** o padrão $\beta = h.item \cup \alpha$ com suporte = *h.suporte*
7. **construa** a base de padrões condicionada de β e então construa a *FP-tree* condicionada de β e chame-a de $tree_\beta$
8. **se** $tree_\beta \neq \emptyset$ então
9. ***FP-growth***($tree_\beta$, β);**}**
10. }

Figura 5.12: *Algoritmo FP-Growth*

A mineração na estrutura *FP-tree*, consiste em, partindo dos nós folha com menor suporte, determinar os prefixos dos caminhos percorridos para atingi-los e, posteriormente, determinar os padrões freqüentes usados para gerá-los. Considere a *FP-tree* da **Figura 5.11**. Para atingir os nós com o item *P5*, existem 2 caminhos (chamados de *base condicional*): *{(P2,P1:1),(P2,P1,P3:1)}*. Uma *base condicional* de um *item* é representada pelo conjunto de caminhos que levam ao *item*. O valor colocado após cada caminho é o valor do contador do *item* naquele ramo. A partir desses caminhos, é construída uma árvore (chamada de *FP-tree condicional*), com somente os ramos "freqüentes" usados para chegar em *P5*.

A *FP-tree condicional* é uma nova *FP-tree* gerada a partir dos caminhos freqüentes da *base condicional*. Para a construção da *FP-tree condicional*, o nó *null* é criado e os caminhos são adicionados à árvore, de maneira que os caminhos freqüentes que possuam um prefixo em comum são agregados. A *FP-tree Condicional* de P_5 é *<P2:2,P1:2>*. Cada ramo da *FP-tree condicional* é representado por um conjunto de itens delimitados por *"<"* e *">"*. Cada *item* representa um nó adicionado a *FP-tree Condicional* seguido pelo o valor que é atribuído ao contador do nó adicionado. A função *FP-Growth* é novamente chamada e a *FP-tree condicional* é passada como parâmetro junto com o elemento *P5*. A **Figura 5.13** ilustra *as bases* e *FP-trees condicionais* dos *itemsets* $\{P_5\}$ *e* $\{P_3\}$. A **Tabela 5.4** mostra as *bases condicionais* e as *FP-tree condicionais* geradas para todos os itens de *L*.

Item	Base Condicional	*FP-tree* Condicional	Padrões Freqüentes Gerados
P5	{(P2 P1: 1), (P2 P1 P3: 1)}	<P2: 2, P1: 2>	P2 P5: 2, P1 P5: 2, P2 P1 P5: 2
P4	{ (P2 P1: 1), (P2 : 1) }	<P2: 2>	P2 P4: 2
P3	{(P2 P1: 2), (P2: 2), (P1: 2)}	<P2: 4, P1: 2> <P1: 2>	P2 P3: 4, P1 P3: 4, P2 P1 P3: 2
P1	{(P2: 4)}	<P2: 4>	P2 P1: 4

Tabela 5.4: *Padrões freqüentes gerados minerando a FP-tree*

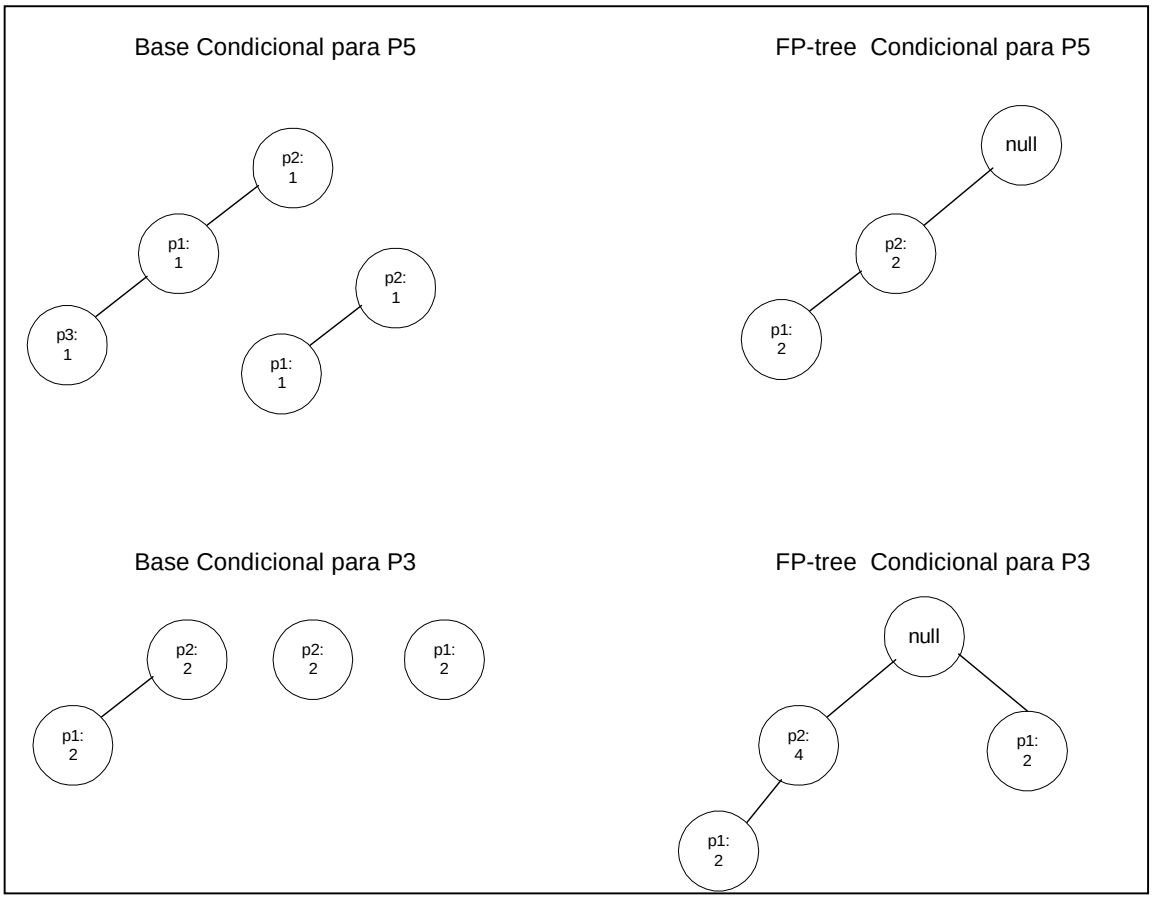


Figura 5.13: *Exemplo de Base Condicional e FP-tree Condicional*

Os padrões freqüentes são gerados quando a *FP-tree* passada como parâmetro para o *FP-Growth* possui somente um ramo, combinando os nós do ramo com o *itemset* passado como parâmetro.

5.8 Algoritmo Eclat

O algoritmo *Eclat* (*Equivalence Class Transformation*) [Zaki et al, 1997] também é um algoritmo usado para a mineração de regras de associação *booleanas* em uma tabela. Ele usa busca em profundidade para determinação de classes de equivalência e usa essas classes de equivalências para determinar os *itemsets* freqüentes .

Como, geralmente, a quantidade de memória e o poder de processamento impedem que todos os conjuntos de *itemsets* candidatos sejam processados simultaneamente, o

algoritmo decompõe o espaço de busca original em partições menores, de tal forma que cada partição possa ser processada independentemente. Essa decomposição é feita dividindo o espaço de busca em classes de equivalência.

Suponha que os itens de um *itemset* estejam ordenados alfabeticamente. Dizemos que dois *itemsets* pertencem à mesma classe de equivalência F_k se eles compartilham um mesmo prefixo de tamanho *k*, onde *k* representa o número de itens que compõem o prefixo. Como exemplo, considere os dados da **Tabela 5.5** (repetição da **Tabela 5.1**) que representam as transações em uma base de dados. A partir dos dados dessa tabela é possível obter as classes de equivalência apresentadas na **Figura 5.14**. A **Figura 5.14** mostra 3 classes de equivalência, com prefixo de tamanho 1 ({1}, {2},{3}). Os prefixos {1,2}, {1,3} e {1,4} pertencem a uma mesma classe de equivalência, pois compartilham o mesmo prefixo, {1}, de tamanho 1.

idTransação	**Itens**
T1	1,2,4
T2	1,2,3
T3	1,3,4
T4	1,4
T5	1,2,4

Tabela 5.5: *Repetição da Tabela 5.1 - Base de dados fictícia para I = {1,2,3,4}*

Uma classe de equivalência pode ser dividida em subclasses de tamanho menor aumentando o tamanho do prefixo. As classes de equivalência podem ser processadas independentemente, de tal modo que os *itemsets* produzidos por uma classe de equivalência sejam independentes dos *itemsets* produzidos por outra classe de equivalência.

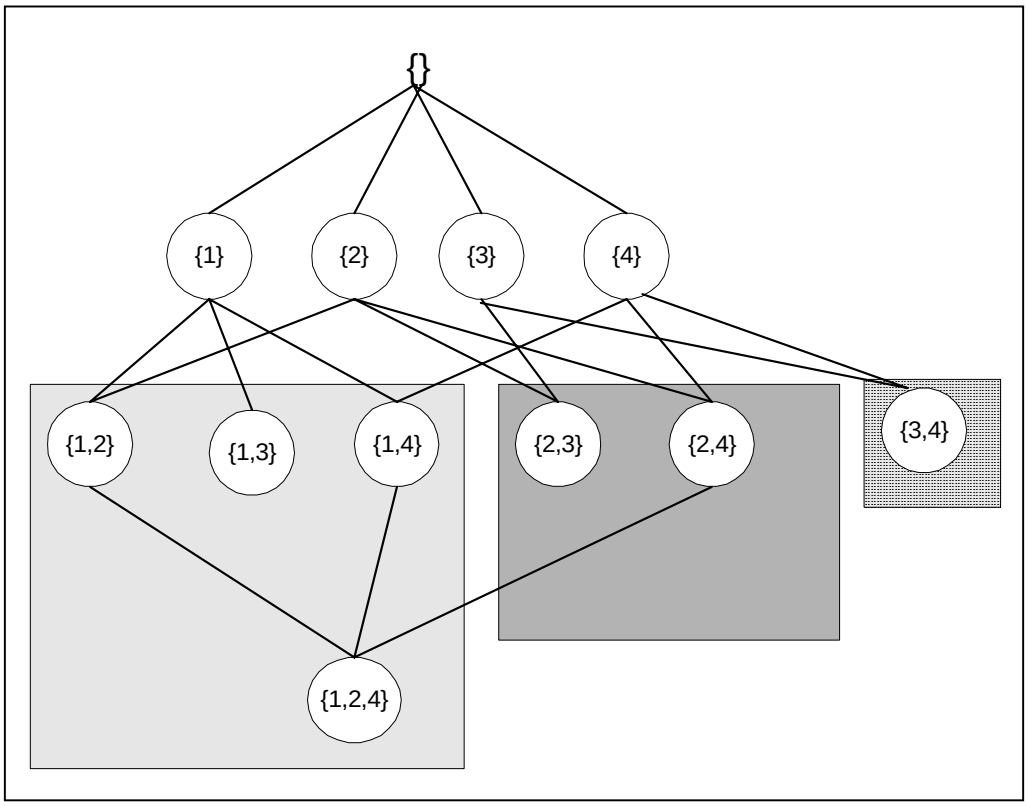


Figura 5.14: *Classes de equivalência relativas aos dados da tabela* ***Tabela 5.5***

O algoritmo *Eclat* usa uma projeção vertical da base de dados. A projeção vertical consiste em uma lista de *itemsets*, onde cada *itemset* possui uma *tidlist* associada (lista dos identificadores das transações onde o respectivo *itemset* ocorreu). Um exemplo de projeção vertical da base de dados da **Tabela 5.5** é mostrado **Tabela 5.6**.

Itemset	*Tidlist*
1	T1,T2,T3,T4,T5
2	T1,T2,T5
3	T2,T3
4	T1,T3,T4,T5

Tabela 5.6: *Projeção vertical da base da* ***Tabela 5.5***

Um acesso à base é feito para determinar os *itemsets-1* freqüentes e construir sua *tidlist*. Esses *itemsets-1* freqüentes são usados para gerar o conjunto de *itemsets-2* candidatos. Os *itemsets* candidatos de tamanho 2 que não forem freqüentes são descartados e os freqüentes são usados para criar as classes de equivalência com prefixo de tamanho 1. Para cada classe de equivalência criada no passo anterior, o algoritmo *Eclat* gera os *itemsets* candidatos de tamanho 3. Esse processo continua, com o conjunto de *itemsets-k-1* freqüentes sendo usado para determinar o conjunto de *itemsets-k* candidatos, até não existir mais *itemsets-k* candidatos.

5.9 Algoritmo FDM

Os algoritmos até aqui apresentados são aplicados no caso dos dados serem provenientes de uma única fonte centralizada. O algoritmo *FDM* (*Fast Distributed Mining of association rules)* foi proposto em [Cheung et al, 1996a] para atender a mineração distribuída envolvendo bases de dados particionadas horizontalmente. O problema tratado pelo algoritmo é encontrar todas as regras de associação cujos valores de suporte e confiança globais sejam maiores do que os valores de suporte e confiança estabelecidos pelo usuário. As etapas do algoritmo *FDM* são resumidas a seguir:

1. **Geração dos *itemsets* candidatos locais de cada partição:** Os *itemsets-k-1* freqüentes são utilizados para gerar os *itemsets-k* candidatos. Para isso, a mesma função *Apriori-Gen* do algoritmo *Apriori* é usada.
2. **Determinação dos *itemsets* freqüentes locais**: Para cada conjunto de *itemsets-k* candidatos, a base de dados de cada *site* é varrida para computar o suporte local dos candidatos. Se o *itemset* candidato é freqüente em um *site*, ele é incluído na lista de *itemsets* freqüentes locais.
3. **Envio dos *itemsets* freqüentes locais**: os conjuntos de *itemsets* freqüentes locais de um *site* são enviados para os demais *sites*. A união dos *itemsets* locais de todos os *sites* (ou partições) forma o conjunto C_G dos candidatos a *itemsets* freqüentes globais. Cada *site* computa o suporte local de cada *itemset* de C_G.

4. **Envio dos contadores de suporte**: Cada *site* envia para os demais *sites* os suportes contabilizados no passo anterior. Com base na soma dos valores de suporte recebidos, cada *site* computa os *itemsets* freqüentes globais.

Ao se trabalhar com bases de dados particionadas, assume-se que os dados das mesmas não possuam replicações. Uma propriedade usada na mineração envolvendo dados particionados horizontalmente é apresentada a seguir [Kantarcioglu and Clifton, 2002]:

"Para um itemset X ser globalmente freqüente ele necessita ser freqüente em pelo menos uma partição"

Observe que essa propriedade é usada no **passo 3** do algoritmo *FDM* onde o conjunto de candidatos a *itemsets* freqüentes globais é determinado a partir da união dos conjuntos de *itemsets* freqüentes locais de cada *site*.

O algoritmo *FDM* não revela os dados individuais de cada transação aos demais *sites*. No entanto, informações importantes relativas ao suporte local de cada *itemset* são reveladas para os demais *sites*. Portanto, o algoritmo *FDM* não preserva a privacidade dos dados (conforme a definição de preservação de privacidade apresentada no Capítulo 4).

Em [Kantarcioglu and Clifton, 2002] foram propostas algumas modificações no algoritmo *FDM* para que ele satisfizesse as restrições de preservação de privacidade. Essas modificações incluem usar um algoritmo de *união segura* de conjuntos na troca de *itemsets* freqüentes locais (etapa 3), de maneira que um *site* não saiba quais são os *itemsets* freqüentes dos demais *sites*, e usar um procedimento de *soma segura* para computar os *itemsets* freqüentes globais. Os procedimentos de *soma segura* e *união segura* estão detalhados em [Clifton et al, 2002].

5.10 Algoritmo de Vaidya e Clifton

Vaidya e Clifton em [Vaidya and Clifton, 2002] propuseram um algoritmo bastante interessante para a mineração de dados distribuídos em bases de dados particionadas verticalmente, com preservação de privacidade. Conforme discutido no capítulo anterior, em uma base de dados verticalmente particionada, os dados de uma transação se encontram distribuídos entre os diferentes *sites*.

O algoritmo de Vaidya e Clifton, apresentado na **Figura 5.15**, é baseado no algoritmo *Apriori* [Agrawal and Srikant, 1994]. A maioria das etapas do algoritmo pode ser executada localmente em cada *site*. No entanto, o passo de determinação do suporte (**linha 7**) requer cooperação entre os diferentes *sites*, e, neste ponto, algumas mudanças foram efetuadas para garantir a preservação da privacidade dos dados. O algoritmo de Vaidya e Clifton usa o cálculo do produto escalar para garantir a privacidade na determinação do suporte dos *itemsets*.

Algoritmo: Mine
Entrada: Base de dados *D*, suporte mínimo *minsup*
Saída: Conjunto de *itemsets* freqüentes *L*

função Mine(D, minsup){
1. L_1 = {*itemsets* freqüentes-1}
2. **para**(k=2;$L_{k-1}\neq\varnothing$;k++) faça{
3. C_k = *Apriori*-gen(L_{k-1});
4. **para** todos os candidatos $c \in C_k$ faça{
5. **se** todos os atributos de *c* estão em um *site* S_i
6. calcule independentemente c.count //suporte de c
7. **senão**
8. calcule c.count através da cooperação dos diferentes *sites*
9. $L_k = L_k \cup c \mid c.count \geq minsup$
10. }
11. }
12. retorne $\cup_k L_k$ }

Figura 5.15: *Algoritmo de mineração de regras de associação em dados particionados verticalmente* [Clifton et al, 2002] .

Seja um *itemset* $Z = A \cup B$ envolvendo $p+q$ atributos, onde A tem p atributos (A_1 até A_p) e B tem o restante dos atributos (B_1 até B_q). Uma transação envolvendo duas partições A e B é uma seqüência de $p+q$ 1s e 0s, onde o valor 1 representa presença de um item e 0 a ausência. Seja s o suporte mínimo da regra e n o número total de transações.

Seja $\vec{X}$ e $\vec{Y}$ representante de colunas da base de dados global P. O produto escalar entre esses dois vetores é definido como:

$$\vec{X}.\vec{Y} = \sum_{i=1}^{n} x_i^* y_i$$

Para determinar se um *itemset-2* é freqüente basta testar se $\vec{X}.\vec{Y} \geq s$. Para computar a freqüência de um *itemset-k* $Z= \{A_1, ..., A_p, B_1, ..., B_q\}$, onde $k=p+q$, cada elemento x_i de $\vec{X}$ e y_i de $\vec{Y}$ passa a ter o valor do produto dos elementos individuais, isto é, $x_i=\prod_{j=1}^{p} A_j$ e $y_i=\prod_{j=1}^{q} B_j$.

O procedimento de cálculo do produto escalar com preservação de privacidade é apresentado na **Figura 5.16**, onde os dois *sites A* e *B* fornecem como entrada para o algoritmo os vetores de cardinalidade *n* $X = (x_1,..,x_n)$ e $Y=(y_1,y_2,...,y_n)$.

Algoritmo: ProdutoEscalar
Entrada: vetores X e Y de cardinalidade *n*
Saída: produto escalar entre X e Y

Função ProdutoEscalar(X,Y){
1. ambos *A* e *B* juntos determinam uma matriz randômica linearmente independente $n \times n/2$ chamada C
2. **para** o *site A* faça **{**
3. *A* gera um vetor randômico *R* de cardinalidade $n/2$ ($\vec{R} = R_1,R_2,..., R_{n/2}$)
4. *A* gera uma matriz $n \times 1$ de adição *X'* multiplicando *C* com *R*, isto é $X' = C \times R$
5. *A* gera $X'' = X' + X$
6. *A* envia *X''* para *B*
7. **}** (primeira mensagem de *A* para *B*)
8. **para** o *site B* faça **{**
9. *B* gera o produto escalar *S'* de *X''* e *Y, ou seja,*

$$S' = \sum_{i=1}^{n} x_i'' * y_i$$

10. *B* também gera a matriz $Y' = C^T \times Y$
11. *B* envia ambas *S'* e *Y'* para *A*
12. **}** (primeira mensagem de *B* para *A*)
13. **para** o *site A* faça**{**
14. *A* gera o fator de subtração: $S'' = \sum_{i=1}^{n} Y_i' * R_i$
15. *A* gera o produto escalar $S= S'-S''$
16. **}** (segunda mensagem de *A* para *B*)
17. **retorne** S **}**

Figura 5.16: *Algoritmo de Produto Escalar com preservação de privacidade* [Vaidya and Clifton, 2002]

Além do método de produto escalar usado para a determinação do suporte dos *itemsets* com a preservação de privacidade, existe também o método baseado no tamanho

da intersecção dos conjuntos [Clifton et al, 2002]. Nesse último método, cada partição *i* representa a presença de um *sub-itemset* X_i através de um vetor S_i, que contém apenas as transações que suportam esse *sub-itemset*. Então o tamanho da intersecção dos conjuntos locais $|\cap_{i=1}^{n} S_i|$ fornece o suporte do *itemset* $X=\cup_{i=1}^{n} X_i$. Para determinar o tamanho dessa interseção é usado um método de criptografia comutativa descrito em [Pohlig and Hellman, 1978] apud [Clifton et al, 2002].

Conforme citado em [Clifton et al, 2002], apesar do método do produto escalar ser aplicado somente no caso de existir duas partições, ele é eficiente para grandes quantidades de dados e envolve menos processamento do que o método que determina o tamanho da interseção, sendo bastante adequado para a mineração de dados.

5.11 Algoritmo Warmr

Nesta seção, o termo consulta refere-se a consultas lógicas. Um exemplo de consulta seria:

?- pai (X, pedro)

No exemplo acima, supondo que existam dois predicados na base de dados consultada:

pai(marcela,pedro)

pai(renata,pedro)

A variável *X* pode assumir dois valores, fornecendo duas respostas à consulta: *X=marcela e X=renata*. O número de respostas a uma determinada consulta representa a **freqüência da consulta**.

O algoritmo *Warmr* foi proposto em [Deshape and Raedt, 1997] para mineração de regras de associação envolvendo múltiplas relações em uma base de dados dedutiva. Um conjunto de consultas *C*, uma freqüência mínima e uma variável *Key* de substituição são fornecidos como entrada para o algoritmo. O objetivo do algoritmo é determinar quais são as consultas de *C* que satisfazem a freqüência mínima estabelecida. A freqüência computada é o número de valores que a variável *Key* pode assumir. De maneira semelhante ao *Apriori*, as consultas lógicas mais complexas (que são as mais específicas) Q_{k+1} são geradas a partir de consultas mais simples Q_k.

O algoritmo *WARM* pode analisar múltiplas relações em conjunto, se a variável de substituição for um atributo que associa as múltiplas relações. Uma boa introdução à mineração *multi-relacional* envolvendo bases de dados dedutivas pode ser encontrada em [Džeroski, 2003].

5.12 Considerações Finais

A mineração de regras de associação consiste em primeiro encontrar os itens freqüentes para, posteriormente, gerar regras de associação fortes. Os algoritmos diferem na maneira em que delimitam o espaço de busca e geram o conjunto de *itemsets* freqüentes. O primeiro algoritmo, AIS [Agrawal et al, 1993], gera um grande número de candidatos desnecessários. O Algoritmo *Apriori* [Agrawal and Srikant, 1994] usa um procedimento de geração de *itemsets* candidados mais eficiente do que o AIS. O *Partition* [Savarese et al, 1995], reduz a sobrecarga de operações de entrada/saída, realizando apenas duas leituras da base de dados. O algoritmo *FP-Growth* não gera *itemsets* candidatos, favorecendo desempenho. No entanto, carrega a árvore *FP-tree* na memória, podendo a memória ser um fator limitante ao aplicar esse algoritmo. Os algoritmos que atendem ao problema de associação clássico foram estendidos para minerar regras de associação em situações mais particulares. Os algoritmos *FDM* [Cheung et al, 1996a] e o algoritmo de Vaidya e Clifton [Vaidya and Clifton, 2002] estendem o algoritmo *Apriori* para a mineração de regras em bases de dados particionadas. O algoritmo *Warmr* também usa os princípios do algoritmo *Apriori* para a mineração envolvendo múltiplas relações de uma base de dados dedutiva.

Os algoritmos discutidos nesse Capítulo foram considerados nesta pesquisa de mestrado, para definir as estratégias adotadas durante o desenvolvimento dos algoritmos de mineração que permitem relacionar múltiplas tabelas fato de um *data warehouse*.

Um resumo das principais características dos algoritmos discutidos neste Capítulo é apresentado na **Tabela 5.7**.

Apriori	• usa busca em largura e contagem direta de ocorrências para determinação do suporte. • usa um procedimento eficiente de geração de *itemsets* candidatos. • percorre muitas vezes a base de dados.
AprioriTID	• usa busca em largura e contagem direta de ocorrências para determinação do suporte. • é semelhante ao *Apriori*. Porém, varre a base de dados apenas uma vez e a carrega na memória. Depois, acessa somente os dados da memória. • transações sem *itemsets* freqüentes são eliminadas da memória. • necessidade de grande montante de memória nos passos iniciais do algoritmo.
Partition	• usa busca em largura e intersecção de *tidlists* para determinação do suporte. • divide a base de dados em partições e processa os *itemsets* freqüentes locais de cada partição. • percorre a base de dados apenas 2 vezes. • carrega na memória as *tidlist*s para a determinação dos *itemsets* freqüentes locais de cada partição.
FP-Growth	• usa busca em profundidade e contagem direta de ocorrências para determinação do suporte. • não gera *itemsets* candidatos. • carrega a *FP-tree* em memória. • necessidade de grande capacidade de memória para carregar a *FP-tree.* • um dos algoritmos mais eficientes .
Eclat	• usa busca em profundidade e intersecção de *tidlists*. • para determinação do suporte divide a base de dados em classes de equivalência. • processa a mineração de *itemsets* freqüentes para cada classe de equivalência.
FDM	• usado para minerar bases de dados particionadas horizontalmente, sem preservar privacidade. • o suporte é determinado através de busca em largura e contagem direta de suporte.
Alg. de Vaiyda e Clinfton	• usado para minerar bases de dados particionadas verticalmente, com preservação de privacidade. • o suporte dos itemsets de uma única partição é determinado através de busca em largura e contagem direta de suporte. • o suporte dos itemsets provindos de partições distintas é determinado através do cálculo do produto escalar entre vetores.
Warmr	• usado para minerar bases de dados lógicas e dedutivas (podendo envolver mais de uma relação). • requer o uso de predicados lógicos. • requer o uso de um conjunto de predicados que delimitam as regras mineradas. • o suporte é determinado pela freqüência da consulta.

Tabela 5.7:*Principais características dos algoritmos de mineração de regra de associação*

6 Mineração de regras de associação *multifatos*

6.1 Introdução

Uma tendência em KDD é o uso conjunto das tecnologias de *data warehouse* e de mineração de dados. Quando os dados de um *data warehouse* são submetidos a um processo de mineração de dados, eles podem revelar padrões que são úteis para serem empregados em uma variedade de finalidades como, por exemplo, o processo de tomada de decisão das grandes empresas. A análise conjunta de múltiplas tabelas fato permite relacionar múltiplos assuntos de um *data warehouse*, sendo muito útil para a exploração do potencial do *data warehouse* como fonte de conhecimento.

As estratégias de mineração existentes anteriormente a este trabalho não permitem a extração de padrões envolvendo múltiplas tabelas fato, pois elas requerem a transferência dos dados para uma única tabela para poderem ser aplicadas. No entanto, essa transferência é feita através de operações de junção que podem acarretar duplicações e alteração na proporção nos dados. Como a mineração considera a proporção dos dados, os padrões obtidos sobre a tabela resultante de junção podem ser distorcidos e não confiáveis.

As regras que relacionam dados provenientes de múltiplas tabelas fato são chamadas, neste trabalho, de regras de associação *multifatos*. Neste Capítulo é definido um novo tipo de mineração de regras de associação que permite a obtenção de padrões através da análise de múltiplas tabelas fato de um *data warehouse*.

Este Capítulo está organizado como segue. Na seção 6.2 é feita a formalização do problema de minerar regras de associação *multifatos*. Na seção 6.3 são apresentadas as estratégias usadas para a mineração *multifatos* e os algoritmos propostos para efetuar essa mineração. Na seção 6.4 são apresentados os resultados de testes realizados com a aplicação do algoritmo *Connection* sobre bases de dados reais. Na seção 6.5 o problema de minerar regras relacionando múltiplas tabelas fato de um *data warehouse* é generalizado, de modo que a mineração *multifatos* possa ser aplicada em situações que envolvam múltiplas relações de uma base de dados relacional e, também, para que possa ser aplicada na mineração envolvendo múltiplas relações de bases distintas. Finalmente, na seção 6.6 são apresentadas as considerações finais deste Capítulo.

6.2 Definição do Problema

Na **Figura 6.1** é apresentado um exemplo de *data warehouse* com dados referentes ao domínio de ensino a distância.

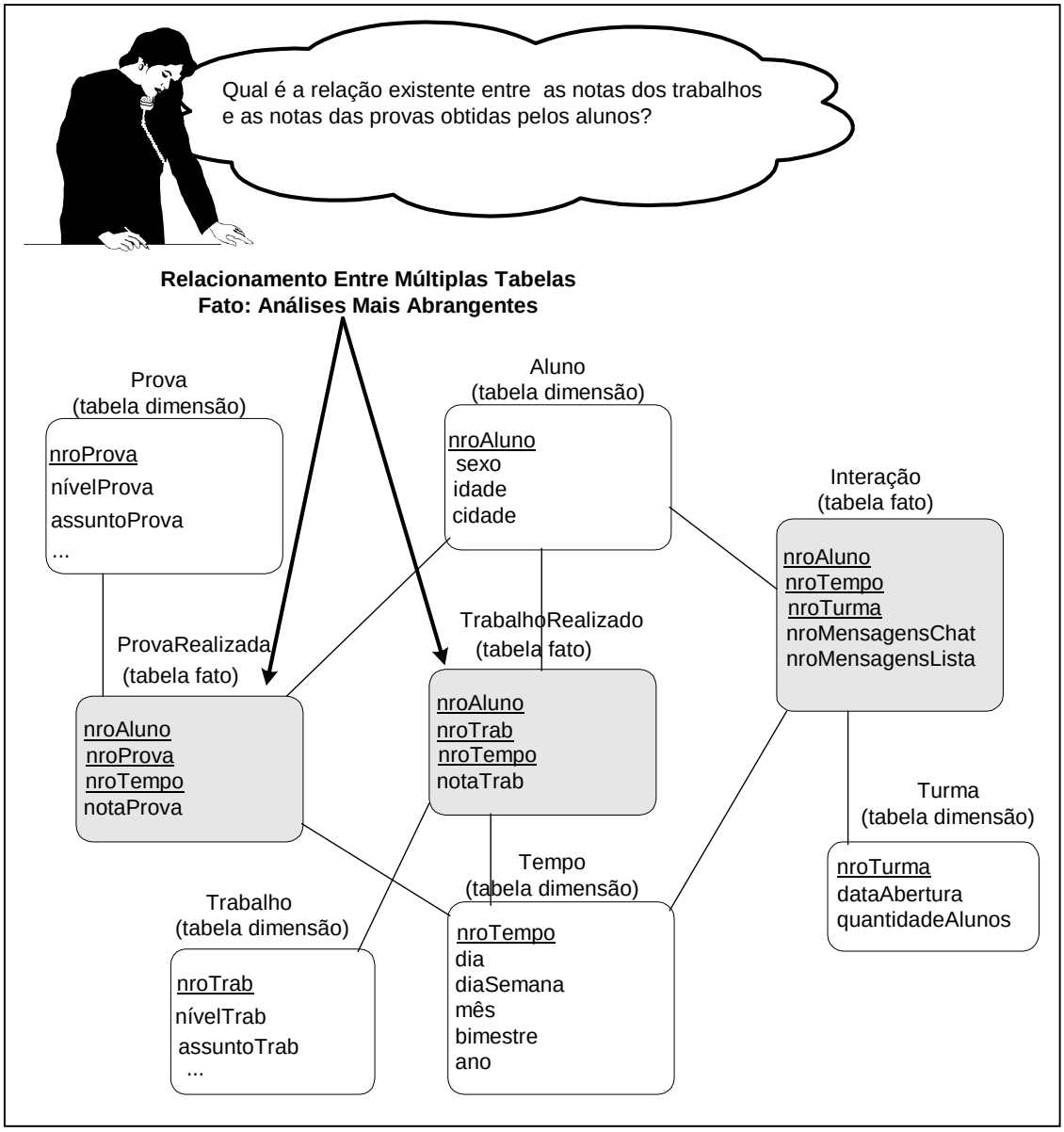


Figura 6.1: Exemplo de aplicação da análise multifatos

O *data warehouse* da **Figura 6.1** possui três tabelas fato, *ProvaRealizada*, *TrabalhoRealizado* e *Interação*, que possuem dados referentes a três assuntos: as provas realizadas pelos alunos, os trabalhos realizados pelos alunos e as interações feitas pelos

alunos com o ambiente de ensino. Um professor pode desejar encontrar padrões que relacionem as notas das provas e as notas dos trabalhos, embora os dados de provas e de trabalhos não estejam diretamente relacionados. Assim, aqui é apresentada uma nova abordagem de mineração de dados: a mineração *multifatos*, que permite que esse tipo de análise seja realizada. Em uma análise minuciosa da literatura na área, não foi encontrada nenhuma abordagem que resolva o problema em questão.

Neste Capítulo, o termo tabela fato é usado para referenciar qualquer tabela envolvendo dados de uma tabela fato e suas dimensões. Assim, uma tabela fato estará representando, além dos dados de seus próprios atributos, também os dados dos atributos de suas dimensões.

A *definição 1*, a seguir, apresenta a definição de regras de associação *multifatos*, cujo propósito é relacionar itens de diferentes tabelas fato. Para facilitar o entendimento, primeiro é apresentado o problema de mineração envolvendo *duas* tabelas fato. Na *seção 6.2.1*, a abordagem é generalizada para *n* tabelas fato.

Definição 1: *Seja F_1 e F_2 duas tabelas fato de um data warehouse. Uma regra de associação multifatos é uma expressão da forma $X \rightarrow Y$, onde X e Y são itemsets, tal que $X \in F_1$ e $Y \in F_2$.*

A *definição 1* estabelece que uma regra de associação *multifatos* é uma regra de associação envolvendo itens de duas tabelas fato, onde o *itemset* antecedente e o *itemset* conseqüente provém de tabelas fato distintas.

Suponha que se deseja analisar conjuntamente as tabelas fato *TrabalhoRealizado* e *ProvaRealizada* do *data warehouse* da **Figura 6.1**, para investigar possíveis influências das notas dos trabalhos nas notas de provas obtidas pelos alunos. Um exemplo de regra de associação *multifatos* que poderia ser obtida a partir dessa análise seria:

$$nroTrab=1,\ notaTrab=A \rightarrow nroProva=2,\ notaProva=A$$

Na regra acima, considere que *X* represente o *itemset* antecedente dessa regra e *Y* represente o conseqüente. Observe que *X* provém da tabela fato *TrabalhoRealizado,* e *Y* provém da tabela fato *ProvaRealizada*, de maneira que, a ocorrência de *X* em *TrabalhoRealizado* leva a ocorrência de *Y* em *ProvaRealizada*, significando que os alunos

que tiram nota *A* no primeiro trabalho também tendem a tirar nota *A* na segunda prova. Note que essas tabelas fato tratam assuntos distintos, mas que estão semanticamente relacionados e, portanto, sendo de interesse a sua análise conjunta.

A mineração *multifatos* tem sentido somente quando as tabelas fato F_1 e F_2 compartilham pelo menos uma dimensão. Assim, é possível relacionar as tabelas fato através das chaves primárias das dimensões compartilhadas *D*.

Para a discussão que se segue, considere *D* um conjunto de dimensões que são comuns a F_1 e F_2 e seja *PK(D)* a união dos atributos chave primária de todas as dimensões $D_i \in D$. Os atributos *PK(D)* são comuns as tabelas fato F_1 e F_2, e seus valores identificam um conjunto de transações das tabelas fato chamado *bloco*. Um *bloco*, cuja definição é fornecida a seguir, é a unidade de análise do processo de mineração *multifatos*.

Definição 2: *Seja* $ID(t_j)$ *o valor dos atributos PK(D) de uma transação* $t_j \in F_i$*, onde* $i=1$ *ou* $i=2$*. O conjunto de transações* $b=\{t_1,t_2,..,t_m\}$ *pertencentes a* F_i*, tal que* $ID(t_k)= ID(t_j)$*,* $\forall\ k, j \mid 1 \leq k \leq m,\ 1 \leq j \leq m$*, onde* $m=|b|$*, é chamado* ***bloco*** *de* F_i*.*

Considere que desejamos relacionar os dados das tabelas fato *ProvaRealizada* e *TrabalhoRealizado* do *data warehouse* da **Figura 6.1**. Observe que a dimensão *Aluno*, cuja chave primária é *nroAluno*, é comum a essas duas tabelas fato, sendo que esse atributo *nroAluno* está presente em ambas as tabelas fato. Assim, suponha que a aluna *Marcela* tenha um conjunto de transações referentes às provas realizadas. Conforme ilustrado na **Figura 6.2**, esse conjunto de transações forma um *bloco* b_1 na tabela fato *ProvaRealizada*, cujo $ID(b_1)=Marcela$. O conjunto de transações da aluna *Marcela* na tabela fato *TrabalhoRealizado* também forma um bloco b_2, cujo $ID(b_2)=Marcela$. O bloco é representado pelo conjunto de transações que ele envolve. Assim, os blocos b_1 e b_2 podem ser representados por:

b_1={<Marcela,nroProva=1,notaProva=C>, <Marcela,nroProva=2,notaProva=B>, <Marcela,nroProva=3,notaProva=A>, <Marcela,notaProva=4,notaProva=A>}

b2 ={<Marcela,nroTrab=1,notaTrab=B>, <Marcela,nroTrab=2,notaTrab=A>, <Marcela,nroTrab=3,notaTrab=A >}

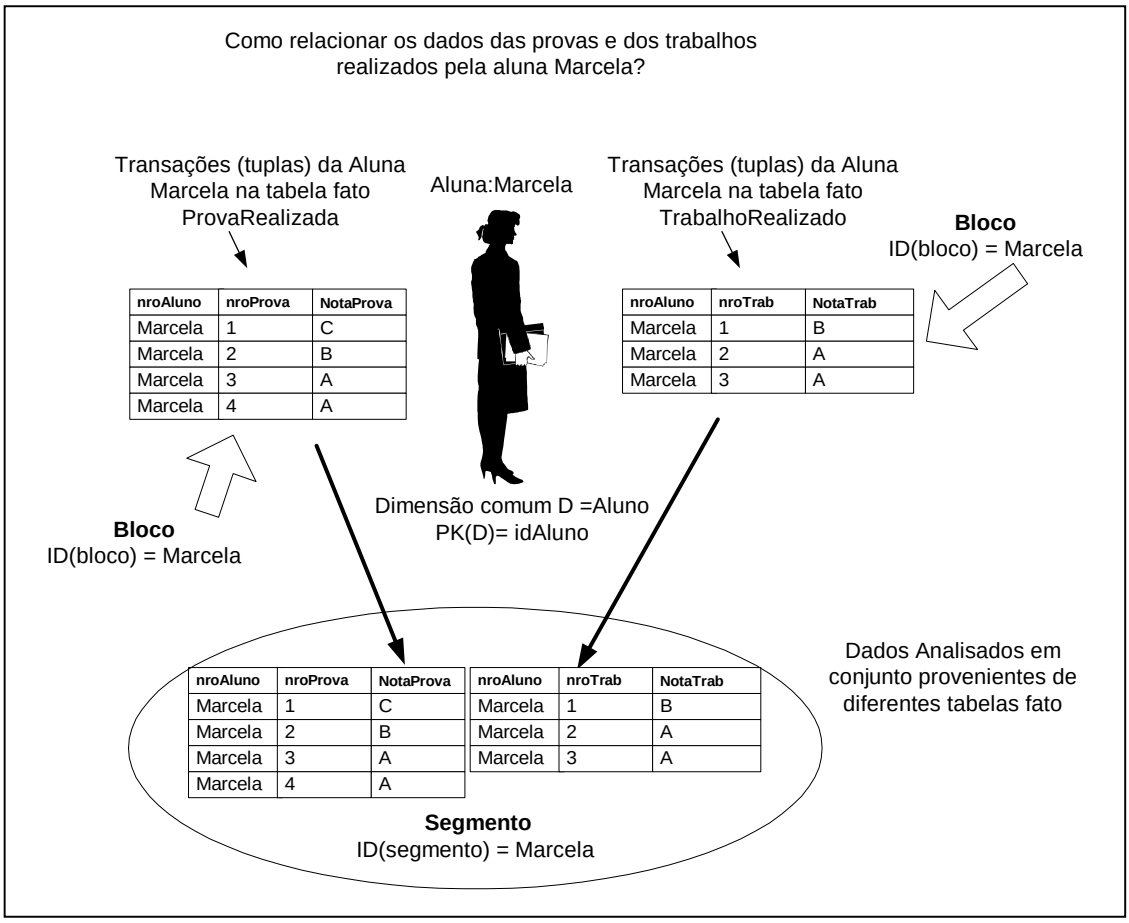


Figura 6.2: Ilustração das unidades de análise usadas na mineração multifatos.

Blocos de diferentes tabelas fato, que possuem o mesmo valor para os atributos *PK(D)*, são relacionados através do processo de mineração de regras de associação *multifatos*. Esse conjunto de *blocos* relacionados é chamado de ***segmento***. Uma definição mais formal de segmento é fornecida a seguir.

Definição 3: *Seja $ID(b_j)$ o valor dos atributos PK(D) para um bloco b_j. O conjunto de blocos $s=\{b_1,b_2\}$ é chamado de* ***segmento*** *se $ID(b_1)=ID(b_2)$, tal que $b_1 \in F_1$ e $b_2 \in F_2$.*

Considere os blocos b_1 e b_2 das tabelas fato *ProvaRealizada* e *TrabalhoRealizado* ilustrados na **Figura 6.2**. Conforme ilustrado nessa figura, o conjunto formado por esses dois blocos forma o *segmento s*:

$$s= \{b_1, b_2\}$$

s={{<Marcela,nroProva=1,notaProva=C>, <Marcela,nroProva=2,notaProva=B>, <Marcela,nroProva=3,notaProva=A>, <Marcela,notaProva=4,notaProva=A>}, {<Marcela,nroTrab=1,notaTrab=B>, <Marcela,nroTrab=2,notaTrab=A>, <Marcela,nroTrab=3,notaTrab=A >}}

De maneira similar à mineração tradicional de regras de associação, que revela tendências de itens ocorrerem juntos em transações de uma relação, a mineração *multifatos* revela tendências de itens ocorrerem juntos em segmentos.

Considere que os dados das tabelas fato *ProvaRealizada* e *TrabalhoRealizado* do *data warehouse* da **Figura 6.1** sejam conforme ilustrados na **Figura 6.3**.

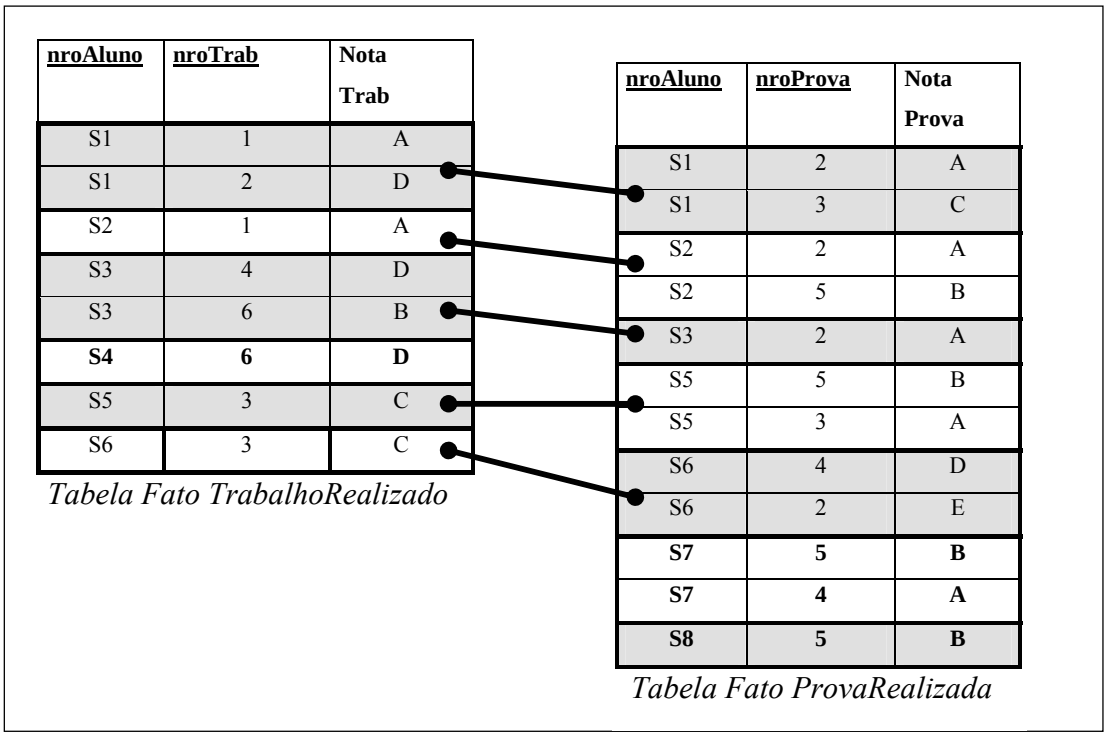


Figura 6.3: Exemplo de Blocos e Segmentos

A **Figura 6.3** mostra os *blocos* de cada tabela fato em cores alternadas. Cada conjunto de *blocos* conectados representa um segmento. Podem existir, em uma tabela fato, *blocos* sem correspondente nas demais tabelas fato, como por exemplo, o bloco *{<S_4 nroTrab=6, notaTrab=D>}* da tabela *TrabalhoRealizado*; neste caso dizemos que o *bloco*

não forma *segmento*. Os *blocos* cujos dados aparecem em negrito não formam *segmento*. A **Figura 6.1** mostra os dados da **Figura 6.3** na representação de blocos e *segmentos*.

Blocos da tabela TrabalhoRealizado
{<S_1,nroTrab=1, notaTrab=A>,<S_1, nroTrab=2, notaTrab=D>}
{<S_2,nroTrab=1, notaTrab=A>}
{<S_3,nroTrab=4, notaTrab=D>,<S_3, nroTrab=6, notaTrab=B>}
{<S_4 nroTrab=6, notaTrab=D>}
{<S_5,nroTrab=3, notaTrab=C>}
{<S_6 nroTrab=3, notaTrab=C>}

Blocos da tabela ProvaRealizada
{<S_1, nroProva=2, notaProva=A>,<S_1, nroProva=3, notaProva=C>}
{<S_2, nroProva=2, notaProva=A>,<S_2, nroProva=5, notaProva=B>}
{<S_3, nroProva=2, notaProva=A>}
{<S_5, nroProva=5, notaProva=B>,<S_5, nroProva=3, notaProva=A>}
{<S_6, nroProva=4, notaProva=D>,<S_6, nroProva=2, notaProva=E>}
{<S_7, nroProva=5, notaProva=B>,<S_7, nroProva=4, notaProva=A>}
{<S_8, nroProva=5, notaProva=D>}

Segmentos de TrabalhoRealizado e ProvaRealizada
{{<S_1,nroTrab=1,notaTrab=A>,<S_1,nroTrab=2,notaTrab=D>},{<S_1,nroProva=2,notaProva=A>, <S_1, nroProva=3, notaProva=C>}}
{{<S_2,nroTrab=1,notaTrab=A>},{<S_2,nroProva=2,notaProva=A><S_2, nroProva=5, notaProva=B>}}
{{<S_3,nroTrab=4, notaTrab=D>,<S_3, nroTrab=6, notaTrab=B>}{<S_3, nroProva=2, notaProva=A>}}
{{<S_5,nroTrab=3, notaTrab=C>}, {<S_5, nroProva=5,notaProva=B><S_5,nroProva=3,notaProva=A>}}
{{<S_6 nroTrab=3, notaTrab=C>}{<S_6, nroProva=4,notaProva=D><S_6,nroProva=2,notaProva=E>}}

Figura 6.4: *Dados da* ***Figura 6.3*** *na representação de blocos e segmentos.*

Uma questão envolvendo a mineração de regras de associação *multifatos* é como quantificar o interesse da regra. Embora várias medidas tenham sido propostas na literatura para quantificar o interesse de uma regra [Bayardo and Agrawal, 1999, Dong and Li, 1998, Freitas, 1998, Freitas, 1999, Hilderman and Hamilton, 1999, Omiecinski, 2003], nenhuma delas é diretamente aplicável à mineração de regras de associação *multifatos*. Dessa maneira, uma nova medida de interesse chamada *peso* foi proposta (*definições 4* e *5*) e as medidas de suporte e confiança foram adaptadas para serem aplicadas para os casos que envolvam múltiplas tabelas fato (*definições 6 a 8*).

Nas definições a seguir considere que F_1 e F_2 sejam as tabelas fato de um *data warehouse* e que X representa um *itemset* de F_1 e Y representa um *itemset* de F_2.

Ao analisar as tabelas fato F_1 e F_2 em conjunto, a medida de interesse ***peso*** (detalhada nas *definições 4 e 5*), quantifica o grau de expressividade da ocorrência de um *item* $x \in X$ em um segmento envolvendo F_1 e F_2, frente a sua ocorrência no total de blocos de F_1.

Definição 4*: *O* *peso *de um* ***item*** $x \in F_i$, *é a razão entre o número de segmentos que contém x e o número de blocos da tabela fato* F_i *em que x ocorre.*

Considere os *blocos* da tabela fato *ProvaRealizada* apresentados na **Figura 6.4**. Considere o item *nroProva=4*; observe que esse item está presente em *dois* blocos da tabela *ProvaRealizada*, porém somente está presente em *um* segmento, portanto seu peso é ½. O significado do peso do item *nroProva=4* ser ½, é que somente 50% de suas ocorrências em *ProvaRealizada* estão relacionadas com dados de *TrabalhoRealizado*. Itens de uma tabela R_1 que possuem uma porcentagem maior de ocorrência conjunta com itens da tabela R_2, frente ao total de sua ocorrência em R_1, são mais interessantes para gerar padrões envolvendo as duas tabelas. Assim, itens com pesos mais expressivos são mais interessantes na geração de regras de associação *multifatos*.

A definição apresentada a seguir estabelece que para uma regra de associação *multifatos* satisfazer o *peso* mínimo, todos os seus itens também devem satisfazer o valor de *peso* mínimo estabelecido pelo usuário.

Definição 5: *Uma regra A satisfaz o peso mínimo se todos os seus itens satisfazem o peso mínimo.*

Considere os dados das tabelas fato *TrabalhoRealizado* e *ProvaRealizada* apresentados **Figura 6.3**. Se o valor estabelecido de peso mínimo for p=0.8, temos que a regra *notaTrab=A* $\rightarrow$ *notaProva=A* satisfaz o peso mínimo, pois todos os seus itens também o satisfazem:

$$peso(notaTrab=A) = 2/2 = 1 \geq p$$

$$peso(notaProva=A) = 4/5 = 0.8 \geq p$$

A *definição 6*, apresentada a seguir, adapta a medida de suporte tradicional para considerar somente a ocorrência dos *itemsets* nos segmentos.

Definição 6: *O* ***suporte#*** *de X é a razão entre o número de segmentos em que X ocorre e o número total de segmentos*.

Conforme a *definição 6*, o *suporte#* de um *itemset X* é a porcentagem dos segmentos em que o *itemset X* ocorre. Considere os segmentos apresentados na **Figura 6.4**. O *suporte#* de *notaProva=E* é 1/5, onde 5 é o número total de *segmentos* e 1 é o número de *segmentos* onde o item *notaProva=E* ocorre. Esse valor de *suporte#* indica que dentre os alunos que fizeram trabalhos e provas (5 alunos), apenas 1 deles tirou (pelo menos) uma nota *E* nas provas realizadas.

Suponha que o segmento apresentado na **Figura 6.4** do aluno S_6 seja alterado para :

{{<S_6 nroTrab=3, notaTrab=C>}{<S_6, nroProva=4, ***notaProva=E*** *><S_6,nroProva=2,* ***notaProva=E*** *>}}*

Nesse caso, o *suporte#* de *notaProva=E* continua sendo 1/5, pois o *suporte#* de *um itemset* é determinado pelo número de segmentos em que ele ocorre. Assim, se um *itemset* ocorrer mais de uma vez em um segmento, somente uma ocorrência do mesmo é contabilizada para o *suporte#*.

Definição 7**:** *O* ***suporte#*** *de uma regra de associação multifatos X→Y é a razão entre o número de segmentos em que X ∪ Y ocorre e o número total de segmentos.*

O *suporte#* de uma regra *X→Y* é a porcentagem dos segmentos em que o *itemset X∪ Y* ocorre. Considere os segmentos apresentados na **Figura 6.4**. O *suporte#* da regra: *notaTrab=A →notaProva=C* é 1/5, que corresponde ao *suporte#* do *itemset {notaTrab=A, notaProva=C}*. Esse valor de suporte indica que dentre os alunos que fizeram trabalhos e provas, apenas um deles teve (pelo menos) uma nota de trabalho *A* e uma nota de trabalho *C*.

Definição 8: *A* ***confiança#*** *de uma regra de associação multifatos X→Y é a razão entre o número de segmentos em que X ∪ Y ocorre e o número de segmentos em que X ocorre.*

A *confiança#* indica qual é a probabilidade de *Y* ocorrer na tabela fato F_2, dado que o *itemset X* ocorreu na tabela fato F_1. Considere os segmentos apresentados na **Figura 6.4**. A *confiança#* da regra: *notaTrab=A→notaProva=C* é 1/2,onde 1 é o número de *segmentos* que contém o *itemset {notaTrab=A, notaProva=C}* e 2 é o número de *segmentos* que contém o *itemset {notaTrab=A}*. Esse valor de *confiança#* indica que, dentre os alunos que fizeram trabalhos e provas e tiveram pelo menos uma nota de trabalho A (2 alunos), um deles obteve (pelo menos) uma nota *C* de prova.

Observe que os cálculos do *suporte#* e *confiança#* consideram apenas os dados dos segmentos. O valor do *peso* de um item também quantifica qual a porcentagem de sua ocorrência nos *segmentos*. Assim, o valor do peso de um item também indica a porcentagem de ocorrências desse item que foi usada no processo de mineração para o cálculo do *suporte#* e *confiança#* das regras em que esse item aparece.

A definição de *itemset freqüente#* (que é uma adaptação da definição de itemset freqüente tradicional aplicada à análise *multifatos*) é apresentada a seguir.

Definição 9: *Um itemset X é* ***freqüente#*** *se todos os seus elementos satisfazem o peso mínimo e se ele satisfaz o suporte# mínimo.*

A *definição 9* estabelece que um *itemset freqüente#* somente pode ser formado por itens que satisfazem o peso mínimo. Considere os segmentos apresentados na **Figura 6.4**. Se forem estabelecidos os valores de peso mínimo *p=0.8* e *suporte#* mínimo *s=0.4*, tem-se que o *itemset X={nroProva=1,notaProva=A}* é *freqüente#*, pois todos os seus itens satisfazem o peso mínimo e ele satisfaz o *suporte#* mínimo:

$$peso(nroProva=2) = 4/4 = 1 \geq p$$

$$peso(notaProva=A) = 4/5=0.8 \geq p$$

$$suporte\#(\{nroProva=1,notaProva=A\})=3/5=0.6 \geq s$$

A definição de regras de associação *multifatos forte#* é apresentada a seguir.

Definição 10: *Se uma regra A satisfaz os mínimos valores estabelecidos de confiança#, suporte# e peso, então essa regra é uma regra* ***forte#****.*

Considere os segmentos apresentados na **Figura 6.4**. Se forem estabelecidos os valores de peso mínimo *p=0.8*, de *suporte#* mínimo *s=0.4* e de *confiança#* mínima *c= 0.7*, tem-se que a regra *notaTrab=A → notaProva=A* é forte# pois:

$$notaTrab=A \rightarrow notaProva=A$$
$$Peso(notaTrab=A) = 1 \geq p$$
$$Peso(notaProva=A) = 4/5 \geq p$$
$$suporte\#(notaTrab=A \rightarrow notaProva=A) = 2/5 \geq s$$
$$confiança\#(notaTrab=A \rightarrow notaProva=A) = 1 \geq c$$

A regra acima indica que, dentre os alunos que fizeram trabalhos e provas, os alunos que tiraram *A* em pelo menos um trabalho tendem a tirar *A* em pelo menos uma prova.

O problema de minerar regras de associação *multifatos* consiste em encontrar todas as regras *fortes#* existentes nos dados de um conjunto de tabelas fatos.

Observe que a *definição 9* é muito importante para restringir o espaço de busca na determinação dos *itemsets freqüentes#*, pois, ela permite eliminar do espaço de busca todos os *itemsets* que tiverem elementos cujo peso é menor do que o peso mínimo especificado.

O problema de mineração *multifatos* foi aqui definido para analisar em conjunto dados de duas tabelas fato. Tendo como base as definições acima, na próxima seção o problema de mineração *multifatos* é estendido para a análise conjunta de *n* tabelas fato.

6.2.1 Generalizando a definição do problema

O problema de mineração *multifatos* envolvendo dados de duas tabelas fato, apresentado na seção anterior, pode ser estendido para a análise de múltiplas tabelas fato em conjunto. Para isso, as definições de regra de associação *multifatos* (*definição 1*) e a definição de segmentos (*definição 3*) devem ser adaptadas.

A *definição 1.1* é uma adaptação da *definição 1*, que ao invés de relacionar dados de duas tabelas fato, define o problema para dois conjuntos de tabelas fato S_1 e S_2.

Definição 1.1: *Seja* $F = \{F_1, F_2,..., F_n\}$ *um conjunto de tabelas fato de um data warehouse e seja* $S_1, S_2 \subseteq F$ *conjuntos de tabelas fato. Seja* $t_{ji} \in R_j$ *uma transação de uma tabela* $R_j \in S_i$. *Uma regra de associação multifatos é uma expressão da forma* $X \rightarrow Y$, *onde X e Y são itemsets, tal que* $X \subseteq \cup_{j=1}^{m} t_{j1}$, $m=|S_1|$, $Y \subseteq \cup_{j=1}^{p} t_{j2}$, $p=|S_2|$, *e* $S_1 \cap S_2 = \phi$.[1]

A *definição 1.1* estabelece que uma regra de associação *multifatos* é uma regra de associação envolvendo itens de múltiplas tabelas fato, onde o *itemset* antecedente e o *itemset* conseqüente provêm de tabelas fato distintas.

Suponha que se deseja analisar conjuntamente as tabelas fato *TrabalhoRealizado*, *ProvaRealizada* e *Interação* do *data warehouse* da **Figura 6.1**, para investigar possíveis influências dos trabalhos realizados e das atividades de interação nas notas de provas obtidas pelos alunos. Um exemplo de regra de associação *multifatos* que poderia ser obtida a partir dessa análise seria:

nroTrab=1, notaTrab=A, nroMensagensChat>10 → *nroProva=2, notaProva=A*

Na regra acima, considere que *X* represente o *itemset* antecedente da regra e *Y* represente o conseqüente. Observe que *X* provém das tabelas fato *TrabalhoRealizado* e *Interação,* e *Y* provém da tabela fato *ProvaRealizada*, de maneira que, a ocorrência de *X* em *TrabalhoRealizado* e *Interação* leva a ocorrência de *Y* em *ProvaRealizada*, significando que os alunos que tiram nota *A* no primeiro trabalho e têm mais de 10 mensagens de *chat* enviadas, tendem a tirar nota *A* na sua segunda prova.

Observe que a análise conjunta de várias tabelas fato somente apresentará regras resultantes se essas tabelas compartilharem pelo menos uma dimensão.

Para analisar múltiplas tabelas fato a definição de segmento (*definição 3*) deve ser adaptada. A *definição 3.3* estabelece que um segmento é um conjunto de *n* blocos com o mesmo identificador, sendo que cada bloco pertence a uma das tabelas fato analisadas.

---

[1] As transações t_{j1} e t_{j2} estão sendo consideradas como conjuntos de valores ao invés de listas ordenadas de valores, uma vez que, a ordem dos valores não é importante aqui.

Definição 3.3: *Seja $F=\{F_1, F_2,..., F_n\}$ um conjunto de tabelas fato. Seja $ID(b_j)$ o valor dos atributos PK(D) para um bloco b_j. O conjunto de blocos $B=\{b_1,b_2,...,b_n\}$ é chamado de* ***segmento*** *de F se $ID(b_i)=ID(b_j)$, $\forall i, j \mid 1 \leq i, j \leq n$, $b_i \in F_i$, $b_j \in F_j$ e $n=|F|$.*

Suponha que se deseje fazer uma análise envolvendo os dados das tabelas fato *TrabalhoRealizado*, *ProvaRealizada e Interação.* Considere os dados das tabelas fato *TrabalhoRealizado* e *ProvaRealizada* apresentados **Figura 6.4** e os dados da tabela *Interação* apresentados na **Figura 6.5**. Observe que os segmentos resultantes da análise dessas três tabelas em conjunto, ilustrados na **Figura 6.5**, são formados por apenas blocos provenientes das três tabelas analisadas.

nroAluno	nroMsgChat
S1	<10
S2	>10
S3	>10
S4	**<10**
S5	>10

Tabela Fato Interação

Blocos da tabela Iteração
{<S_1,nroMsgChat<10}
{<S_2,nroMsgChat>10}
{<S_3,nroMsgChat>10}
{<S_4,nroMsgChat<10}
{<S_5,nroMsgChat>10}

Segmentos de F={TrabalhoRealizado, ProvaRealizada e Interação}
{{<S_1,nroTrab=1,notaTrab=A><S_1,nroTrab=2,notaTrab=D>},{<S_1,nroProva=2,notaProva=A>, <S_1, nroProva=3, notaProva=C>}{<S_1,nroMsgChat<10}}
{{<S_2,nroTrab=1,notaTrab=A>},{<S_2,nroProva=2,notaProva=A><S_2, nroProva=5, notaProva=B>}, {<S_2,nroMsgChat>10}}
{{<S_3,nroTrab=4, notaTrab=D><S_3, nroTrab=6, notaTrab=B>}{<S_3, nroProva=2, notaProva=A>},{<S_3,nroMsgChat>10}}
{{<S_5,nroTrab=3, notaTrab=C>}, {<S_5, nroProva=5,notaProva=B><S_5,nroProva=3,notaProva=A>},{<S_5,nroMsgChat>10}}

Figura 6.5: *Segmentos de F={TrabalhoRealizado, ProvaRealizada e Interação}*

O restante das definições apresentadas para análise de duas tabelas fato podem ser diretamente aplicadas para relacionar *n* ($n \geq 2$) tabelas fato.

Tendo como base as definições apresentadas anteriormente, a próxima seção descreve as técnicas de mineração desenvolvidas para encontrar regras de associação *multifatos*

fortes# em um *data warehouse* e apresenta dois novos algoritmos de mineração *multifatos*: o *Relation* e o *Connection*.

6.3 Mineração de Regras de Associação Multifatos

A mineração de regras de associação *multifatos* é mais complexa que a mineração de regras de associação tradicional devido principalmente à necessidade de identificação dos *segmentos* para relacionar os dados das múltiplas tabelas. Em geral, a mineração de regras de associação *multifatos* pode ser dividida em 4 fases:

- ***Fase 1:*** Identificar os *segmentos*.
- ***Fase 2:*** Encontrar o *suporte#* e o *peso* dos *itemsets* de cada tabela fato, determinando os *itemsets freqüentes#* locais.
- ***Fase 3****:* Encontrar os *itemsets freqüentes#* globais.
- ***Fase 4****:* Gerar as regras de associação *multifatos*.

Dois algoritmos foram desenvolvidos para a mineração *multifatos*: o algoritmo *Relation* e o algoritmo *Connection*. O *Relation* é um algoritmo baseado no *Apriori* e foi o primeiro a ser desenvolvido devido à sua simplicidade. No entanto, o algoritmo *Relation* se mostrou muito lento ao computar grandes quantidades de dados e, então, em uma tentativa de obter um melhor desempenho, o algoritmo *Connection* foi desenvolvido. O *Connection* é um algoritmo baseado no *FP-Growth* que permitiu uma melhora no desempenho em frente ao seu antecessor *Relation*. Esses algoritmos diferem basicamente na *fase 2* da mineração *multifatos*, que é a fase mais custosa em termos de processamento.

As estratégias de cálculo de suporte e determinação dos *itemsets* freqüentes dos algoritmos tradicionais (apresentadas no capítulo 5) podem ser usadas na *fase 2* da mineração de regras de associação *multifatos*. Na *fase 2* da mineração *multifatos*, o algoritmo *Relation*, assim como o *Apriori*, usa contagem direta de suporte e busca em largura para determinar os *itemsets freqüentes#* locais. Já o algoritmo *Connection*, de maneira semelhante ao *FP-Growth*, usa busca em profundidade e contagem direta de suporte para determinar os *itemsets freqüentes#* locais. Na *etapa 3* de mineração *multifatos*, ambos os algoritmos, *Relation* e *Connection*, assim como o *Partition*, usam

interseção de *tidlist* para determinar os *itemsets* globais, que são aqueles que envolvem elementos de relações distintas.

6.3.1 Algoritmo *Relation*

O algoritmo *Relation* é apresentado na ***Figura 6.6***.

Algoritmo: *Relation*
Entrada: O conjunto de tabelas fato *F*, valores de suporte# mínimo s, confiança# mínima c, e peso mínimo p
Saída: O conjunto *C* de regras de associação *multifatoss* mineradas

Função *Relation*(F,s,c,p**) {**
1 $T=\{ID(b_i)|\ ID(b_i)=ID(b_j),\ b_i \in F_i \text{ e } b_j \in F_j,\ \forall\ i,j \mid 1 \leq i \leq |F| \text{ e } 1 \leq j \leq |F|\}$
2 $R=\phi$
3 **para** cada tabela fato $F_i \in F$ **faça {**
4 *determine o conjunto* R_i *de itemsets freqüentes# locais*
5 $R = R \cup R_i$
6 ***}***
7 *Encontre o conjunto G de itemsets freqüentes# globais usando R*
8 *C= Gere as regras multifatos a partir de G.*
9 **retorne C }**

Figura 6.6: *Algoritmo Relation*

Na **linha 1**, o conjunto de *segmentos* de *F* é encontrado. Para isso, todas as tabelas fato $F_i \in F$ são percorridas, os identificadores $ID(b_i)$ dos ***blocos*** b_i que estão presentes em todas as tabelas fato $F_i \in F$ são adicionados a *T*. Após a execução da **linha 1**, *T* possui os identificadores de todos os *segmentos* de *F*.

Nas **linha 2**, o algoritmo *Relation* inicializa o conjunto *R* com vazio. *R* é usado para armazenar os conjuntos de *itemsets freqüentes*# locais R_i de todas as tabelas fato F_i.

No *loop* das **linhas 3 a 6** os *itemsets freqüentes#* locais de cada tabela fato F_i são encontrados. Na implementação da **linha 4**, o mesmo procedimento de determinação dos *itemsets* freqüentes do algoritmo *Apriori* foi usado, com as seguintes alterações:

- *Diferentemente do Apriori, em que apenas um contador (o contador de suporte) é associado a cada itemset, no Relation dois contadores são associados a um itemset: o contador de segmentos e o contador de blocos. O contador de blocos é incrementado sempre que o itemset em questão ocorre em um bloco. O contador de segmento é incrementado somente quando o itemset em questão ocorre em um bloco b_j que forma segmento (ou seja, quando $ID(b_j) \in T$).*
  - *No Relation, uma tidlist é mantida para cada itemset. Sempre que um itemset X ocorre em um segmento s_i, o $ID(s_i)$ é adicionado em X.tidlist.*
  - *No Relation é calculado o suporte# ao invés do suporte tradicional, que é computado no Apriori. O suporte# de um itemset é indicado pelo valor de seu contador de segmento.*
  - *No Relation, o peso de cada item é computado. O peso de um item é determinado pela razão entre o valor de seu contador de segmento e o valor de seu contador de bloco, sendo que somente os itens que satisfazem o peso mínimo estabelecido podem ser usados para formar itemsets candidatos.*

Na **linha 7**, o conjunto *R* é usado para determinação do conjunto de *itemsets freqüentes#* globais. Seja $n=|F|$ o número de tabelas fato analisadas, R_i é o conjunto contendo todos os *itemsets freqüentes#* da tabela fato F_i retornado no passo anterior do algoritmo, e r_i é um *itemset* pertencente a R_i. A *tidlist* de um *itemset* r_i é obtida interseccionando as *tidlist*s de todos os elementos que formam esse *itemset*. O processo de determinação do conjunto *G* de *itemsets* freqüentes globais consiste em encontrar todos os *itemsets* $g = r_1 \cup r_2 \cup_{...} \cup r_n$, tal que $|r_1.tidlist \cap r_2.tidlist,..., \cap r_n.tidlist|/|T| \geq s$, onde *s* é o *suporte#* mínimo estabelecido pelo usuário.

Na **linha 8**, o conjunto *G* é usado para gerar o conjunto *C* de regras de associações *multifatos fortes#*. O algoritmo gera todas as combinações de *X* e *Y* que satisfazem a restrição da *definição 1*. O conjunto C de regras de associação *multifatos* é então retornado na **linha 9**.

A seguir é feita uma comparação entre os algoritmos *Apriori* e *Relation* para permitir uma avaliação comparativa do desempenho do *Relation*, porém o algoritmo *Apriori* não fornece o conjunto de regras obtidas pelo algoritmo *Relation*. Suponha que um conjunto de *n* tabelas fato F_i sejam submetidas ao algoritmo *Relation*. O algoritmo *Relation* acessa uma tabela fato duas vezes a mais (para determinar os segmentos e para carregar as tidlist dos itemsets freqüentes#) do que o número de vezes que o algoritmo *Apriori* acessa. Considerando que todo o processamento dos dados é feito em memória, o grau de complexidade do algoritmo *Relation* é linear (em operações de entrada e saída) com relação ao *Apriori*: se o *Apriori* faz uma média de *A* acessos a cada tabela, o algoritmo *Relation* faz *n.A+2n* acessos.

A operação de varredura das tabelas é uma operação lenta, pois envolve acesso a disco. O algoritmo *Relation* executa várias vezes a varredura de cada uma das tabelas analisadas: na determinação dos *segmentos* e na determinação dos *itemsets freqüentes#* para cada conjunto de *itemsets* candidatos gerados, resultando em um baixo desempenho.

Uma versão preliminar do algoritmo *Relation* foi implementada. No entanto, devido ao baixo desempenho conduzido nos testes envolvendo grandes volumes de dados, formas mais eficientes de determinar o conjunto de *itemsets freqüentes#* locais foram pesquisadas. Assim, foi desenvolvido o algoritmo *Connection* que reduziu o número de varreduras efetuadas nas tabelas. Esse algoritmo é detalhado na próxima subseção.

6.3.2 Algoritmo *Connection*

O algoritmo *Connection* usa uma estrutura chamada *MFP-tree* para a determinação dos *itemsets freqüentes#* locais. A MFP-tree foi uma estrutura que foi desenvolvida durante o projeto de pesquisa para permitir que os *itemsets freqüentes#* locais fossem determinados com somente duas varreduras das tabelas fato envolvidas. A *MFP-tree* é uma estrutura baseada na *FP-tree* usada no algoritmo *FP-Growth* [Han et al, 2000], com as seguintes modificações:

- Na *MFP-tree* dois campos, um que armazena a *tidlist* e um campo que armazena o valor do *peso* de cada item, são adicionados aos elementos da tabela *Header* da estrutura original da *FP-tree*.

Cada nó da *MFP-tree* corresponde a um item *freqüente#* e cada ramo corresponde a um *itemset* encontrado em uma ou mais transações dos *segmentos* de uma tabela fato.

Basicamente, o algoritmo *Connection* encontra os *itemsets freqüentes#* locais de cada relação usando a *MFP-tree* e obtém as *tidlist* desses *itemsets*. Posteriormente, os *itemsets freqüentes#* provenientes de relações distintas são combinados para formar os *itemsets* globais, sendo que suas *tidlist*s são interseccionadas para determinar os *suportes#* dos *itemsets* globais encontrados. Então, os *itemsets freqüentes#* globais são usados para formar as regras *fortes#*. O algoritmo *Connection* é apresentado na **Figura 6.7** (no *Apêndice A* esse algoritmo é apresentado com maior refinamento).

Na **linha 1**, o conjunto de *segmentos* de F é encontrado seguindo um procedimento semelhante ao procedimento do algoritmo *Relation*: todas as tabelas fato $F_i \in F$ são percorridas, e os identificadores $ID(b_i)$ dos blocos $\boldsymbol{b_i}$, que estão presentes em todas as tabelas fato $F_i \in F$, são adicionados a T.

Nas **linhas 2 e 3**, o algoritmo *Connection* inicializa dois conjuntos, M e R, com vazio. Esses conjuntos são usados respectivamente para armazenar as *MFP-tree*s M_i e o conjunto de *itemsets freqüentes#* locais R_i de todas as tabelas fato F_i.

No *loop* das **linhas 4 a 9**, todas as tabelas fato F_i são varridas. A *MFP-tree* M_i da tabela fato F_i é construída na **linha 5**. A construção da *MFP-tree* é baseada na construção da *FP-tree* (apresentada no Capítulo 5) com algumas mudanças. A tabela fato F_i é varrida uma vez e o *suporte#* e o *peso* de cada item são calculados. Somente os itens que satisfazem o suporte# mínimo e o peso mínimo são considerados para construir a *MFP-tree*. O campo *peso* de um elemento da tabela *Header* é inicializado com o valor do *peso* do item que ele representa. Para preencher a *MFP-tree* M_i, apenas dados originando dos *blocos* $b_j \in F_i$ tal que $ID(b_j) \in T$ são processados, diferentemente da *FP-tree* em que todos os dados são considerados. Além disso, $ID(b_j)$ é adicionado a *tidlist* de todos os elementos da tabela *Header* que representam os itens encontrados no *bloco* b_j.

Na **linha 6,** a *MFP-tree* M_i é usada para determinar o conjunto R_i (o super-conjunto de todos os *itemsets freqüentes#* da tabela fato F_i). Considere que *ntransações(X)* e *nsegmentos(X)* sejam, respectivamente, o número de ocorrências do *itemset* X em

transações[1] dos blocos F_i que formam *segmentos* e o número de ocorrências do *itemset* X nos *segmentos*. O algoritmo varre a *MFP-tree* M_i, adicionando ao conjunto R_i todos os *itemsets* X tal que $ntransações(X) \geq s \times |T|$, onde s é o valor de suporte mínimo. Como um *itemset* X é *freqüente#* se $nsegmentos(X) \geq s \times |T|$, e $ntransações(X) \geq nsegmentos(X) \geq s \times |T|$, tem-se que R_i é um super-conjunto contendo todos os *itemsets freqüentes#* da tabela fato F_i. Nas **linhas 7 e 8**, a *MFP-tree* M_i e o conjunto R_i de cada relação são adicionados respectivamente ao conjunto M e R.

Algoritmo: *Connection*
Entrada: O conjunto de tabelas fato F, valores de suporte# mínimo s, confiança# mínima c, e peso mínimo p
Saída: O conjunto C de regras de associação *multifatoss* mineradas

Função *Connection*(F,s,c,p**) {**
1. $T=\{ ID(b_i)|ID(b_i)=ID(b_j), b_i \in F_i \text{ e } b_j \in F_j , \forall i,j \mid 1 \leq i \leq |F| \text{ e } 1 \leq j \leq |F| \}$
2. M= ϕ
3. R = ϕ
4. para cada tabela fato $F_i \in F$ **faça {**
5. *Construa a MFP-Tree M_i de F_i*
6. *Encontre o conjunto R_i de itemsets freqüentes# locais da MFP-Tree M_i*
7. $M = M \cup M_i$
8. $R = R \cup R_i$
9. *}*
10. *Encontre o conjunto G de itemsets freqüentes# globais usando os conjuntos M e R*
11. *Gere o conjunto* ***C*** *de regras de associação multifatos a partir de* ***G***
12. retorne C }

Figura 6.7: *Algoritmo* Connection.

A *tidlist* de cada *itemset freqüente#* pode ser obtida através da tabela *Header,* já que a mesma armazena a *tidlist* de cada item *freqüente#.* Na **linha 10,** os conjuntos M e R são usados pelo algoritmo *Connection* para determinar o conjunto G de *itemsets freqüentes#* globais. Esse processo é feito usando interseção de *tidlist* dos *itemsets freqüentes#* locais de maneira semelhante ao algoritmo *Relation*. Na **linha 11,** o conjunto G é usado para gerar o

[1] Na construção da árvore *MFP-Tree* é necessário considerar os *itemsets* de cada transação individual para que os ramos da *MFP-Tree* contenham as associações corretas dos itens. No entanto isso não afeta o cálculo do *suporte#* de um *itemset*, que continua sendo por segmentos. A determinação do suporte de um *itemset* é feita através da interseção do conjunto de identificadores dos segmentos que contém os seus itens.

conjunto *C* de regras de associações *multifatos*. O algoritmo gera todas as combinações de *X* e *Y* que satisfazem as restrições da *definição 1* e da *definição 10*. Finalmente, o conjunto *C* com as regras de associação *multifatos fortes#* é retornado na **linha 12**.

Pode-se usar o algoritmo *FP-Growth* para permitir um parâmetro de comparação de desempenho com o algoritmo *Connection*, lembrando que o algoritmo *FP-Growth* não fornece o mesmo conjunto de regras obtidas pelo algoritmo *Connection*. Suponha que um conjunto de *n* tabelas fato F_i sejam submetidas ao algoritmo *Connection*. O algoritmo *Connection* varre uma vez todas as *n* tabelas fato para determinar os *segmentos* e aplica, para cada tabela fato analisada, um procedimento que faz o mesmo número de varreduras que o *FP-Growth* faz em uma tabela. Considerando que todo o processamento dos dados é feito em memória, o grau de complexidade do algoritmo *Connection* é linear com relação ao *FP-Growth*: se o *FP-Growth* faz 2 acessos a uma tabela, o algoritmo *Connection* faz $2n+n=3n$ acessos.

Como os *itemsets freqüentes#* locais são gerados separadamente para cada tabela fato, pode-se aumentar a performance do algoritmo *Connection* através do processamento paralelo, fazendo com que diferentes processadores encontrem os *itemsets freqüentes#* locais de cada tabela fato separadamente e, posteriormente, um processador central encontre o conjunto de *itemsets freqüentes#* globais e gere as regras de associação *multifatos*.

Para exemplificar o funcionamento do algoritmo *Connection* considere os dados das tabelas fato *TrabalhoRealizado* e *ProvaRealizada* ilustradas na **Figura 6.3**. Considere os valores de *suporte#* mínimo *s=0.4, confiança#* mínima *c=0.8* e peso mínimo *p*=0.7. As *MFP-tree*s M_1 e M_2, mostradas na **Figura 6.9** e **Figura 6.10**, são obtidas executando o algoritmo *Connection* sobre o conjunto de tabelas fato *F={TrabalhoRealizado, ProvaRealizada}*. Cada nó da *MFP-tree* consiste de um item, seguido pelo número de ocorrências dos itens nas *transações* dos segmentos.

A seguir é ilustrado o procedimento de construção da FP-tree M_1 a partir dos dados da tabela fato *TrabalhoRealizado*. Antes de iniciar a construção da *MFP-tree* M_1, o conjunto *T* dos identificadores dos segmentos é encontrado.

$T = \{S_1, S_2, S_3, S_5, S_6\}$

Posteriormente, o conjunto *L* de *itemsets freqüentes#* é identificado e ordenado na ordem decrescente do suporte de seus suportes (ver construção da FP-Tree no Capítulo 5).

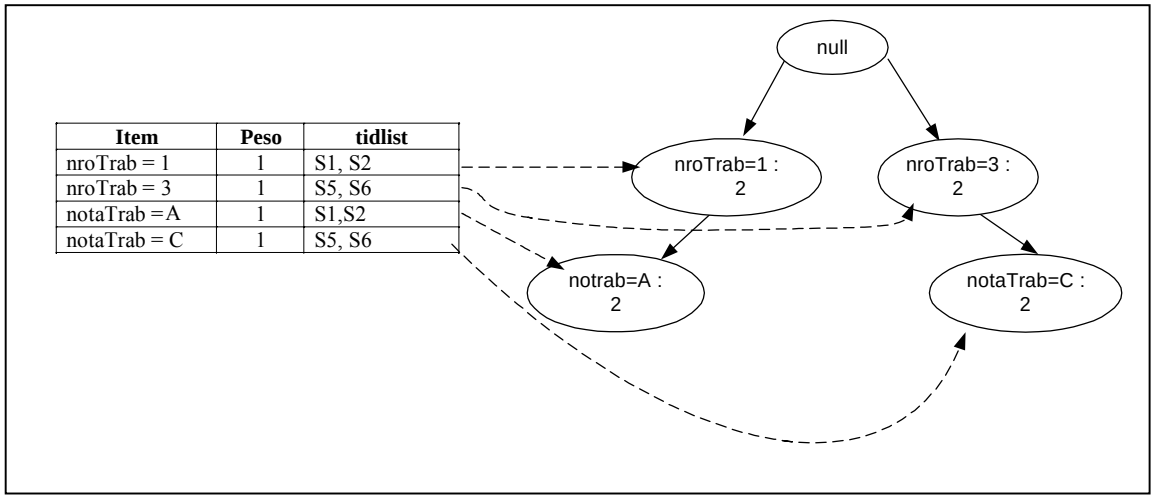
L = {nroTrab=1, nroTrab=3, notaTrab=A, notaTrab=C}

Somente os blocos de *TrabalhoRealizado* que pertencem a um *segmento* (ou seja, aqueles cujo identificador está em *T*) e somente os itens *freqüentes#* das transações desses blocos são considerados para a construção da *MFP-Tree* M_1. A **Figura 6.8** mostra os blocos de *TrabalhoRealizado*, com somente os itens *freqüentes#* (ordenados na ordem de *L)*, que serão usadas para a construção da *MFP-tree* M_1. Na **Figura 6.8** o *"símbolo"* representa um *itemset* infreqüente que será desconsiderado para a construção da *FP-tree*.

Blocos da tabela TrabalhoRealizado
{<S_1,nroTrab=1, notaTrab=A>,<S_1, -, ->}
{<S_2,nroTrab=1, notaTrab=A>}
{<S_3,-, ->,<S_3, -, ->}
{<S_5,nroTrab=3, notaTrab=C>}
{<S_6 nroTrab=3, notaTrab=C>}

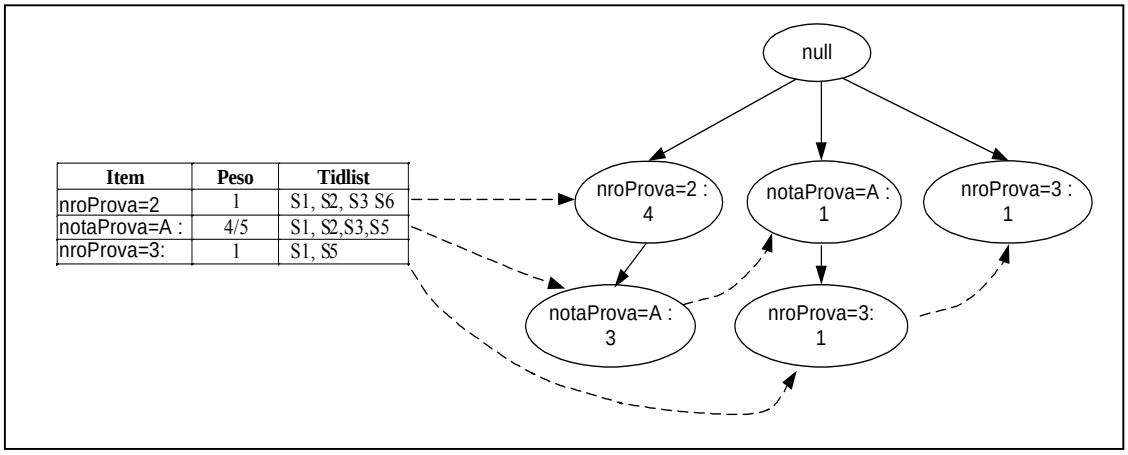
Figura 6.8: Blocos pertencentes aos segmentos da tabela TrabalhoRealizado, com apenas a os itemsets freqüentes# (seguindo a ordenação de L).

Para a construção da MFP-tree M_1, o algoritmo inicia processando o primeiro bloco da tabela fato *ProvaRealizada*, que é o bloco identificado por S_1. Ao ler a primeira transação de S_1, o algoritmo cria um novo nó *nroTrab=1* como filho do nó raiz, e cria um novo nó *notaTrab=A* como filho de do nó *nroTrab=1*. O campo *contador* de cada nó criado é sempre inicializado com o valor 1. Além disso S_1 é adicionado nas *tidlists* dos elementos da tabela *Header* correspondentes a *nroTrab=1* e *notaTrab=A*. Uma vez que a segunda transação de A_1 não possuí itens *freqüentes#*, ela é desconsiderada pelo algoritmo. O algoritmo, ao ler o segundo bloco identificado por S_2, verifica que o prefixo {*nroTrab=1, notaTrab=A}* já existe na árvore. Então, os campos *contador* dos nós *nroTrab=1* e *notaTrab=A* são incrementados em uma unidade, e S_2 é adicionado na *tidlist* dos elementos da tabela *Header* correspondente a esses itens. Esse procedimento se repete, até varrer todos os blocos da tabela *TrabalhoRealizado* que pertencem ao conjunto de segmentos, resultando na MFP-tree M_1 ilustrada na **Figura 6.9**. O mesmo procedimento é aplicado aos dados da tabela *ProvaRealizada* resultando na MFP-tree M_2, ilustrada na **Figura 6.10**.



Item	Peso	tidlist
nroTrab = 1	1	S1, S2
nroTrab = 3	1	S5, S6
notaTrab =A	1	S1,S2
notaTrab = C	1	S5, S6

Figura 6.9: MFP-tree M_1 da tabela fato TrabalhoRealizado



Item	Peso	Tidlist
nroProva=2	1	S1, S2, S3 S6
notaProva=A :	4/5	S1, S2,S3,S5
nroProva=3:	1	S1, S5

Figura 6.10: *MFP-tree M_2 da tabela fato ProvaRealizada*

As regras *fortes#* mineradas para as tabelas fato *F={ProvaRealizada, TrabalhoRealizado}* são fornecidas abaixo, onde *s#* e *c#* indicam, respectivamente, os valores de *suporte#* e *confiança#* da regra. O valor do *peso* de cada item *i* de uma regra é indicado como pr(*i*).

nroTrab=1 → nroProva=2
s#=0.4, c#=1, pr(nroTrab=1)=1, pr(nroProva=2) = 1

nroTrab=1 → notaProva=A
s#=0.4, c#=1, pr(nroTrab=1)=1, pr(notaProva=A)=0.8

notaTrab=A → nroProva=2
s#=0.4, c#=1, pr(notaTrab=A)=1, pr(nroProva=2) = 1

notaTrab=A → notaProva=A
s#=0.4, c#=1, pr(notaTrab=A) = 1, pr(notaProva=A)=0.8

nroTrab=1 → nroProva=2, notaProva=A
s#=0.4, c#=1, pr(nroTrab=1) = 1, pr(nroProva=2) = 1, pr(notaProva=A)= 0.8

notaTrab=A →nroProva=2, notaProva=A
s#=0.4, c#=1, pr(notaTrab=A) = 1, pr(nroProva=2) = 1, pr(notaProva=A)= 0.8

nroTrab=1, notaTrab=A → nroProva=2
s#=0.4, c#=1, pr(nroTrab=1) = 1, pr(notaTrab=A) = 1, pr(nroProva=2) = 1

nroTrab=1, notaTrab=A → notaProva=A
s#=0.4, c#=1, pr(nroTrab=1) = 1, pr(notaTrab=A) = 1, pr(notaProva=A)= 0.8

nroTrab=1, notaTrab=A → nroProva=2, notaProva=A
s#=0.4, c#=1, pr(nroTrab=1) = 1, pr(notaTrab=A) = 1, pr(nroProva=2) = 1, pr(notaProva=A)= 0.8

A regra minerada *notaTrab=A →notaProva=A* indica que os alunos que tiram nota *A* em pelo menos um trabalho tendem a tirar nota *A* em pelo menos uma das provas. O valor 0.4 de *suporte#* indica que o *itemset {notaTrab=A, notaProva=A}* está presente em 40% dos *segmentos*. O valor 1 de *confiança#* indica que o item *notaProva=A* está presente em todos os *segmentos* que contêm o item *notaTrab=A*. Os valores de *peso*, *pr(notaTrab=A)=1* e *pr(notaProva=A)=0.8*, indicam que todas as ocorrências de *notaTrab=A* e 80% das ocorrências de *notaProva=A* estão presentes nos segmentos analisados e, conseqüentemente, foram usadas no processo de mineração *multifatos*.

Os resultados obtidos seriam diferentes se um algoritmo de mineração de regras de associação fosse aplicado em uma tabela resultante da junção das tabelas fato envolvidas, pois durante a operação de junção, os dados das tabelas originais são replicados levando ao cálculo errôneo dos valores de suporte e confiança das regras.

Para o cálculo correto das medidas de interesse, a técnica aqui apresentada mantém os dados das múltiplas tabelas separadas, conservando a estrutura das transações originais intacta, de maneira a evitar replicações. O uso de *blocos* e *segmentos* para o cálculo do

suporte# e *confiança#* permite que essas medidas de interesse sejam mais adequadas para representar as relações entre múltiplas tabelas do que as tradicionais medidas de suporte e confiança.

6.4 Resultados Experimentais

Esta seção apresenta os resultados dos testes aplicados em dados envolvendo o *data warehouse D* que é descrito a seguir. Os experimentos foram executados em um computador pessoal Pentium-III com 256 megabytes de memória principal, com o sistema operacional Windows/XP. O algoritmo *Connection* foi escrito na linguagem de programação Java.

O *data warehouse D* mostrado na **Figura 6.11** é derivado de um sistema acadêmico real[1].

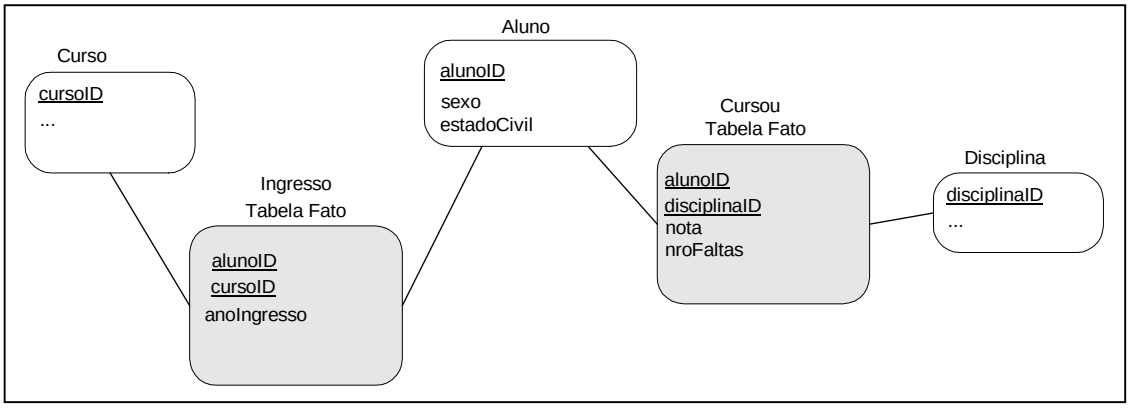


Figura 6.11: Data warehouse *D*

A tabela dimensão *Aluno* contém informações sobre todos os estudantes matriculados na universidade. A tabela fato *Cursou* tem informações sobre a performance dos alunos nas disciplinas. A tabela fato *Ingresso* contém informações sobre o ingresso do aluno na universidade (curso, ano ingresso, ...). Para a nossa análise, a seguinte operação de junção foi executada nos dados do *data warehouse D*, resultando na tabela *IngressoAluno*:

$IngressoAluno \leftarrow Aluno * Ingresso$

[1] O sistema acadêmico foi cedido por uma universidade brasileira, porém por motivo de privacidade o nome dessa universidade não será citado.

Onde $*$ representa a operação *Junção Natural* da *Álgebra Relacional.*

Um exemplo de regra minerada aplicando o algoritmo *Connection* no conjunto de tabelas fato *F={Cursou, IngressoAluno}* é apresentado abaixo, onde os valores de nota $\in$ *[0..10]* foram discretizados em intervalos de *2* unidades, e "GA" representa a disciplina Geometria Analítica.

sexo=masculino, curso=Engenharia → disciplinaID=GA, nota=[7..8];
suporte#=36%; confiança#=70%;
peso(sexo=masculino)=21%; peso(curso=Engenharia)=97%;
peso(disciplinaID=AG)=93%; peso(grade=[7..8])=93%.

A regra acima mostra que os alunos do sexo masculino, que são do curso Engenharia, tendem a tirar notas variando de 7 a 8 nas disciplinas de Geometria Analítica. O valor do *suporte#* indica que *36%* dos alunos, que possuem entrada em ambas as tabelas fato, são do sexo masculino, fazem Engenharia e obtiveram notas de 7 a 8 na disciplina de Geometria Analítica. O valor da *confiança#* indica que, dos estudantes de sexo masculino que fazem Engenharia, 70% têm notas de 7 a 8 na disciplina de Geometria Analítica. Uma vez que, somente os dados relacionados foram considerados para calcular o *suporte#* e a *confiança#*, a porcentagem de ocorrência dos itens *sexo=masculino, curso=Engenharia, disciplinaID=GA* e *nota=[7..8]* que estavam presentes nos *segmentos* (dados relacionados de ambas as tabelas fato) e foram usadas no cálculo do *suporte#* e *confiança#* são, respectivamente, *21%*, *97%, 93%* e *93%*. Observe que essa regra só pôde ser minerada porque as transações de cada estudante foram agrupadas, o que permitiu que os valores de *suporte#* ficassem acima do valor mínimo especificado.

A **Tabela 6.1** mostra o número de transações de cada tabela fato analisada.

Relação	**Número de Transações**
IngressoAluno	22592
Cursou	215219

Tabela 6.1: Número de Transações

Para verificar o efeito dos valores da nova medida de interesse *peso* no processo de mineração, aplicamos o algoritmo *Connection* no mesmo conjunto de dados, mantendo os valores de *suporte#* e *confiança#* e variando o valor de peso mínimo usado como parâmetro de entrada do algoritmo. Observou-se que o tempo de execução do algoritmo *Connection* é bastante dependente do valor do peso mínimo usado na mineração. Os gráficos da ***Figura 6.12*** e ***Figura 6.13*** mostram, respectivamente, a variação do número de regras geradas e a variação do tempo de execução em função do peso mínimo usado ao aplicar o algoritmo *Connection* no conjunto de tabelas *{IngressoAluno e Cursou}*.

Como pode ser visto nos gráficos da **Figura 6.12** e **Figura 6.13**, quando o *peso* aumenta, existe uma tendência para o número de regras mineradas e o tempo de execução reduzir até atingir um valor constante. O tempo de mineração diminui com o aumento do valor do peso mínimo porque a restrição de peso mínimo elimina do espaço de busca os elementos que não a satisfaz. A eliminação desses elementos faz com que o número de regras mineradas também diminua.

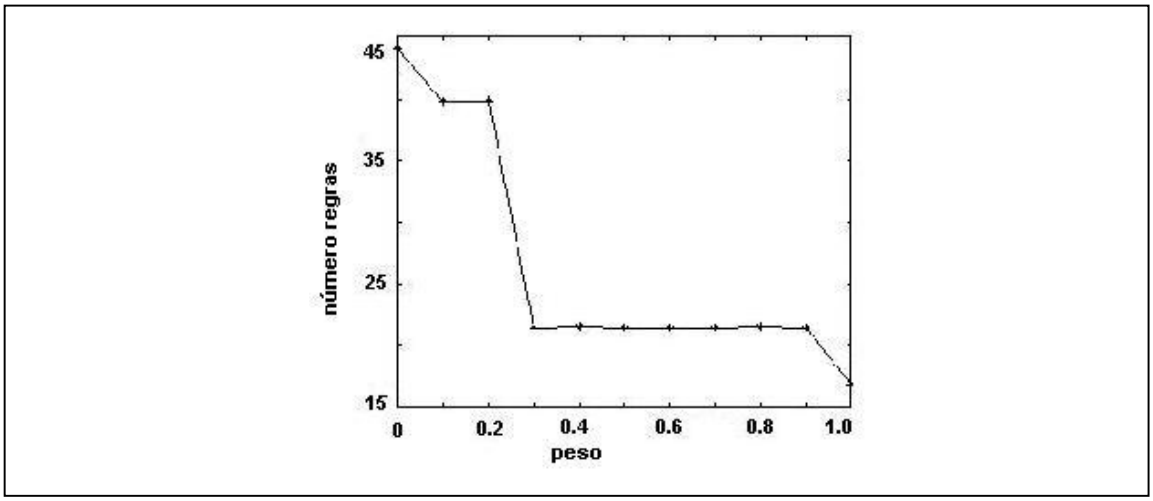


Figura 6.12: *Variação do número de regras mineradas pelo algoritmo Connection aplicado as tabelas fato {IngressoAluno,Cursou} em função do peso mínimo*

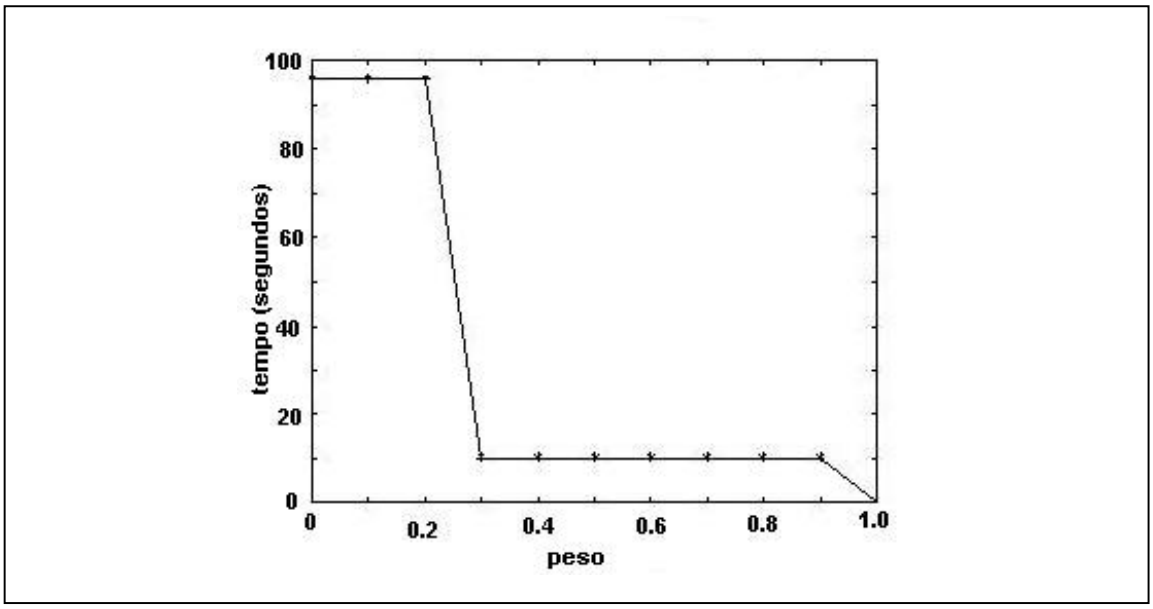


Figura 6.13: *Variação do tempo de execução do algoritmo Connection aplicado as tabelas fato {IngressoAluno,Cursou} em função do peso mínimo*

6.5 Estendendo o uso da análise multifatos

As técnicas de mineração *multifatos* podem ser diretamente aplicadas para relacionar múltiplas relações de uma base de dados relacional desde que essas múltiplas relações possuam atributos em comum. Na **Figura 6.14** é apresentado um esquema de um banco de dados, onde a entidade *Aluno* participa de dois relacionamentos de cardinalidade $n \times m$: *FazTrabalho* e *FazProva*. Os dados das relações originadas desses relacionamentos podem ser analisados através da mineração *multifatos* através do atributo comum *idAluno*.

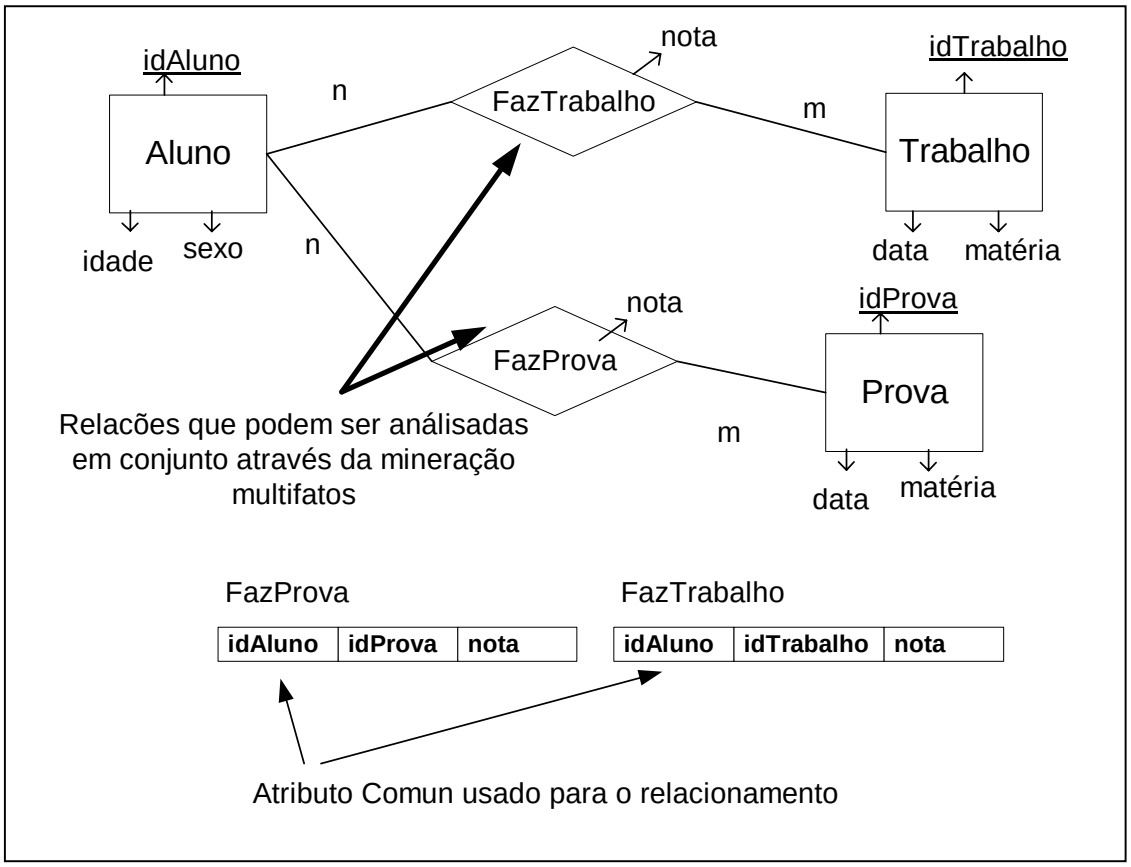


Figura 6.14: *Aplicação da mineração multifatos para analisar múltiplas relações em conjunto.*

A análise *multifatos* também pode ser aplicada, com algumas adaptações para analisar múltiplas relações de bases de dados distintas. No entanto, alguns cuidados devem ser tomados para que a análise envolvendo múltiplas bases de dados possa ser efetuada [Zhang et al, 2003]:

- Se as relações provenientes das bases possuírem dados em formatos diferentes, deve ser aplicado um processo às bases analisadas de maneira a tornar compatíveis os atributos envolvidos na análise. Além disso, as informações duplicadas devem ser removidas.
- Se os dados das bases analisadas não podem ser divulgados, procedimentos que garantam a privacidade devem ser tomados ao relacionar dados de bases distintas, de maneira que as informações sobre as transações internas de cada base não sejam reveladas para as demais.

A adaptação do algoritmo *Connection* para minerar múltiplas relações de bases distintas não é trivial, principalmente no que se refere ao aspecto de manter a privacidade dos dados no processo de determinação do suporte dos *itemsets* globais, onde informações provenientes das múltiplas tabelas devem ser trocadas. Existem dois modos para determinar o suporte dos *itemsets* globais mantendo a privacidade: o primeiro modo consiste em usar as técnicas de criptografia comutativa descritas em [Pohlig and Hellman, 1978] para interseccionar informações relativas a transações de bases distintas, mantendo a privacidade dos dados das mesmas; o segundo modo consiste em usar o método do produto escalar descrito em [Vaidya and Clifton, 2002] (discutido no Capítulo 5) para contabilizar o suporte dos *itemsets* globais.

6.6 Considerações Finais

Neste Capítulo apresentamos o fruto do trabalho de pesquisa realizado no mestrado. Inicialmente o problema foi definido e, posteriormente, dois algoritmos de mineração *multifatos* foram propostos. O primeiro algoritmo desenvolvido foi o *Relation* . O *Relation* é um algoritmo baseado no *Apriori*, que se mostrou excessivamente lento ao computar grandes quantidades de dados. Buscando alternativas para melhorar o desempenho, o algoritmo *Connection*, baseado no *FP-Growth*, foi desenvolvido. O algoritmo *Connection* foi implementado e testado usando dados reais. Os testes mostraram resultados bastante interessantes, não obtidos pelas demais técnicas de mineração.

Os resultados deste trabalho foram bastante promissores. A mineração *multifatos* é mais complexa do que as técnicas de mineração tradicionais, porém ela permite uma análise diferente daquelas realizadas pelos outros algoritmos existentes.

As técnicas aqui apresentadas, embora tenham sido desenvolvidas para serem aplicadas na análise conjunta de múltiplas tabelas fato de um *data warehouse*, também podem ser aplicadas para analisar em conjunto múltiplas relações de uma base de dados relacional, desde que as mesmas possuam atributos em comum.

7 Conclusões

7.1 Introdução

Nesta monografia foi apresentado o problema de mineração de regras de associação *multifatos* que permite analisar múltiplas tabelas fato de um *data warehouse*. Foram apresentados os conceitos básicos envolvidos na tecnologia de *data warehouse*. A tarefa de associação foi detalhada a fundo: os principais tipos de mineração de regras de associação foram discutidos e os principais algoritmos de mineração de regras de associação foram apresentados. Assim, foi possível criar o embasamento teórico para a compreensão da técnica de mineração de regras de associação *multifatos*. A mineração de regras de associação *multifatos* atende a um tipo de problema não abordado anteriormente na literatura, não se encaixando nos tipos de mineração discutidos no Capítulo 4 e nem podendo ser realizada usando os algoritmos de mineração apresentados no Capítulo 5.

A técnica de mineração *multifatos* aqui apresentada, ao analisar múltiplas tabelas fato em conjunto, permite uma análise diferente daquelas realizadas pelos outros algoritmos existentes, também podendo ser aplicada na análise de múltiplas relações de uma base de dados relacional.

7.2 Contribuições

A maior contribuição deste trabalho foi o desenvolvimento de uma técnica para a mineração de regras de associação *multifatos*, relacionando dados de diferentes assuntos de um *data warehouse*. O desenvolvimento dessa técnica resultou nas seguintes contribuições:

- Definição do problema de mineração de regras de associação multifatos.
- Desenvolvimento de dois algoritmos de mineração de regras de associação multifatos.
- Criação da estrutura de dados MFP-tree, que é uma adaptação da FP-tree [Han et al, 2000], que foi usada no algoritmo Connection, para melhorar o desempenho da mineração de regras de associação multifatos.

O método aqui proposto também pode ser aplicado para relacionar itens de múltiplas relações de uma base de dados relacional, desde que as mesmas tenham atributos em comum, como é o caso das relações originadas dos relacionamentos que envolvem uma mesma entidade. Embora as técnicas aqui apresentadas requeiram um esforço adicional em comparação com as técnicas de mineração tradicionais, elas permitem que novos padrões, não obtidos através de outros métodos de mineração, sejam descobertos.

Este trabalho representa um avanço na capacidade de análise dos dados do *data warehouse*, uma vez que, os outros métodos de mineração existentes permitiam apenas a descoberta de padrões envolvendo uma única tabela fato do *data warehouse.* A partir daqui, padrões relacionando múltiplas tabelas fato e, conseqüentemente, múltiplos assuntos de um *data warehouse*, podem ser obtidos.

7.3 Trabalhos Futuros

A partir deste trabalho, pioneiro na análise *multifatos*, várias direções de pesquisas e trabalhos futuros podem ser sugeridas:

1. ***Análise conjunta de múltiplas tabelas fato semanticamente relacionadas:*** Uma situação muito interessante, não abordada neste trabalho, é a análise de múltiplas tabelas fato indiretamente relacionadas através de tabelas fato intermediárias, que não necessariamente possuem uma dimensão em comum, mas que mesmo assim, existe interesse na análise conjunta das mesmas devido à semântica envolvida entre as informações dessas tabelas. Esse tipo de problema representa um objeto de pesquisa para trabalhos futuros que poderá proporcionar outros tipos de análises interessantes sobre os dados de um *data warehouse*. A **Figura 7.1** apresenta um exemplo desse tipo de problema. No *data warehouse* ilustrado **Figura 7.1** existem três tabelas fato: *Prova*, *Trabalho* e *Assistência*, sendo que a tabela *Prova* contém dados relativos às provas realizadas pelos alunos, a tabela *Trabalho* contém informações relativas ao trabalho realizados pelos alunos, e a tabela *Assistência* contem informações relativas à monitoria prestada para os trabalhos realizados. Uma análise interessante, que poderia ser obtida através do relacionamento das tabelas fato *Prova* e *Monitoria*, seria determinar a influência dos dados da

monitoria prestada (por exemplo, número de horas da assistência, perfil do monitor, etc) nas notas de provas obtidas pelos alunos.

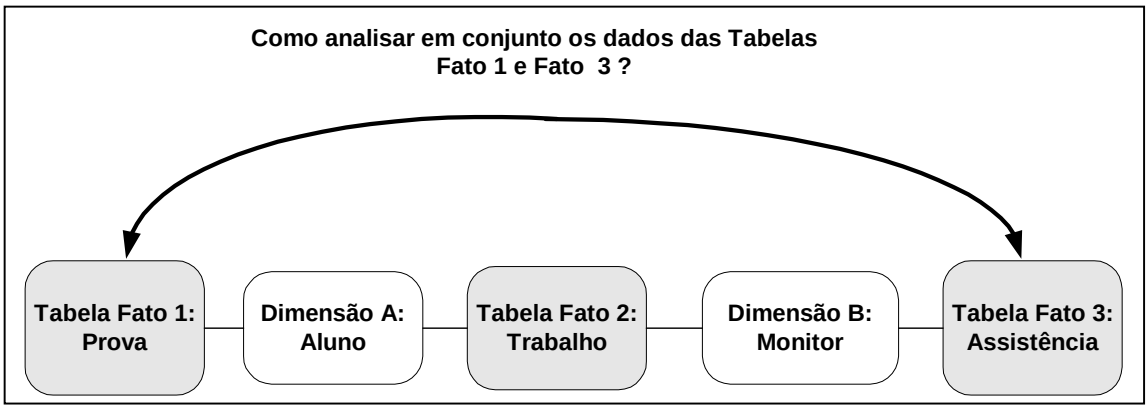


Figura 7.1: *Análise conjunta de múltiplas tabelas fato semanticamente relacionadas*

2. ***Estender a análise multifatos para outras tarefas de mineração:*** Este trabalho é o passo inicial de uma nova frente de pesquisa: *a mineração multifatos*. Além das regras de associações relacionando múltiplas tabelas fatos, outros tipos de padrões podem ser futuramente explorados:

   - *Classificação Multifatos.* Classificar dados de uma tabela baseado não só no comportamento de seus próprios atributos, como também, no comportamento dos atributos de outras tabelas relacionadas. Por exemplo, através da classificação *multifatos* seria possível classificar os registros de consultas dos clientes de um determinado posto de saúde (armazenadas em uma tabela fato) considerando os dados das transações de compras de remédio efetuadas pelo mesmo (armazenadas em outra tabela fato).

   - *Padrões Seqüenciais Multifatos.* Um exemplo desse tipo de padrão seria: usando o *data warehouse* da **Figura 6.1**, determinar qual será a próxima nota de prova do aluno, baseado não só nas notas de suas provas anteriores, mas também, nas notas de seus trabalhos anteriores.

3. ***Análise multifatos em dados quantitativos:*** As técnicas desenvolvidas neste projeto de pesquisa foram direcionadas para a mineração de dados *booleanos*. Trabalhos futuros podem adaptar as técnicas de mineração aqui apresentadas para que elas também possam ser aplicadas diretamente aos casos que

envolvam dados de outros domínios, como por exemplo, o domínio quantitativo.

4. ***Utililizar técnicas de preservação de privacidade na análise multifatos:*** Trabalhos futuros podem adicionar técnicas de mineração com preservação de privacidade na mineração multifatos para que dados de múltiplas tabelas fato, localizadas em diferente sites, possam ser analisados em conjunto de uma maneira segura, sem que informações individuais de cada tabela fato sejam reveladas.
5. ***Complexidade do Algoritmo:*** Aqui, a eficiência dos algoritmos desenvolvidos (*Relation* e *Connection*) foi medida comparando seu desempenho com os algoritmos de mineração tradicionais *Apriori* e *FP-Growth*. Um trabalho futuro seria medir a complexidade do algoritmo em termos de operações de memória e também em termos de operações de entrada e saída.
6. ***Criar outros tipos de regras:*** A técnica desenvolvida durante este projeto de pesquisa permite a mineração de regras que relacionam itens que ocorrem pelo menos uma vez nos segmentos (blocos de transações relacionados). Um exemplo desse tipo de regra é notaProva=A→notaTrab=B, significando que os alunos que tiram pelo menos uma nota A de prova tendem a tirar pelo menos uma nota B de trabalho. Um trabalho futuro seria estender a técnica aqui desenvolvida para relacionar itens que ocorrem na maioria das transações dos segmentos, assim a seguinte regra poderia ser gerada: notaProva=A→notaProva=B, significando que os alunos que tiram nota A na maioria das provas efetuadas, também tiram nota B na maioria dos trabalhos realizados. A medida de interesse peso também poderia ser validada para essa nova solução, visando determinar se ela pode ser aplicada na mineração desse novo tipo de regra.

8 Referências

[Agrawal and Aggarwal, 2001] Agrawal, D.; Aggarwal, C.C. *On the Design and Quantification of Privacy Preserving Data Mining Algorithms*. In Proc. of the 20th ACM Symposium on Principles of Database Systems, Santa Barbara, California, USA, 2001. p. 1-12.

[Agrawal et al, 1993] Agrawal, R.; Imielinski, T.; Swami, A. *Mining association rules between sets of items in large databases*. In Proc. of the ACM SIGMOD Int'l Conf. on Management of Data, Washington, D.C., USA, 1993. p. 207-216.

[Agrawal and Srikant, 1994] Agrawal, R.; Srikant, R. *Fast algorithms for mining association rules*. In Proc. of the Int'l Conf. on Very Large Databases, Santiago de Chile, Chile, 1994.

[Agrawal and Srikant, 1995] Agrawal, R.; Srikant, R. *Mining Sequencial Patterns*. In Eleventh International Conference on Data Engineering, Taipei, Taiwan, 1995. p. 3-14.

[Agrawal and Ramakrishnan, 2000] Agrawal, R.; Ramakrishnan, S. *Privacy-preserving Data Mining*. In Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of Data, Dallas, Texas, USA, 2000.

[Agrawal et al, 2003] Agrawal, R.; Evfimievsk, A.; Srikant, R. *Information sharing across private databases*. In ACM SIGMOD international conference on on Management of data, San Diego, USA, 2003. p. 86-97.

[Bayardo and Agrawal, 1999] Bayardo, R.; Agrawal, R. *Mining the most interesting rules*. In Proc. of the ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining, San Diego, CA, USA, 1999.

[Chaudhuri and Dayal, 1997] Chaudhuri, S.; Dayal, U. *An Overview of Data Warehousing and OLAP Technology*. In ACM SIGMOD, 1997.

[Cheung et al, 1996a] Cheung, D.W.; Han, J.; Ng, V.T.; Fu, A.W.; Fu, Y. *A fast distributed algorithm for mining association rules*. In Fourth International Conference on Parallel and Distributed Information Systems, Miami Beach, Florida, USA, 1996a. p. 31-42.

[Cheung et al, 1996b] Cheung, D.W.; Ng, V.T.; Fu, A.W.; Fu, Y.J. *Efficient Mining of Association Rules in Distributed Databases.* IEEE Transactions on Knowledge and Data Engineering, vol. 8, December 1996b.

[Clifton et al, 2002] Clifton, C.; Kantarcioglu, M.; Vaidya, J.; Lin, X.; Zhu, M.Y. *Tools for Privacy Preserving Data Mining.* SIGKDD Explorations, vol. 4, 2002. p. 28-34.

[Datcu and Seidel, 2000] Datcu, M.; Seidel, K. *Image information mining: exploration of image content in large archives*. In Aerospace Conference, 2000. p. 253 - 264.

[Deshape and Raedt, 1997] Deshape, L.; Raedt, L. *Mining association rules in multiple relations*. In Proc. of the 7th International Workshop on Inductive Logic Programming, Prague, Czech Republic, 1997. p. 125-132.

[Dong and Li, 1998] Dong, G.; Li, J. *Interestingness of Discovered Association Rules in Terms of Neighborhood-based Unexpectedness*. In Proc. of the Second Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'98), Melbourne, Austrália, 1998.

[Džeroski and Raedt, 2002] Džeroski, S.o.; Raedt, L. *Multi-relational Data Mining: a Workshop Report.* ACM SIGKDD Explorations Newsletter, vol. 4, 2002. p. 122-124.

[Džeroski, 2003] Džeroski, S.o. *Multi-relational data mining: an introduction.* ACM SIGKDD Explorations Newsletter, vol. 5, 2003. p. 1 - 16.

[Elmasri and Navathe, 2000] Elmasri, R.; Navathe, S. *Fundamentals of Database Systems*. Addison-Wesley, 3a edição, 2000.

[Evfimievski et al, 2002] Evfimievski, A.; Srikant, R.; Agrawal, R.; Gehrke, J. *Privacy Preserving Mining of Association Rules*. In Proc. of the 8th ACM SIGKDD Int'l Conf. on Knowlegde Discovery and Data Mining, Edmonton, Alberta, Canada, 2002.

[Evfimievski et al, 2003] Evfimievski, A.; Gehrke, J.; Srikant, R. *Limiting Privacy Breaches in Privacy Preserving Data Mining*. In Proc. of the 22th ACM Symposium on Principles of Database Systems, San Diego, California, USA, 2003.

[Fayyad et al, 1996] Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P. *From Data Mining to Knowledge Discovery in Databases*. 1996. p. 37-54.

[Freitas, 1998] Freitas, A. *On Objective Measures of Rule Surprisingness*. In Proc. of the Second European Conference on the Principles of Data Mining and Knowledge Discovery (PKDD'98), Nantes, França, 1998.

[Freitas, 1999] Freitas, A. *On rule interestingness measures.* Knowledge-Based Systems, vol. 12, 1999.

[Gray and Orlowska, 1998] Gray, B.; Orlowska, M. *CCAIIA: Clustering Categorical Attributes Into Interesting Association Rules*. In Proc. of the Second Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'98), Melbourne, Austrália, 1998.

[Han et al, 2000] Han, J.; Pei, J.; Yin, Y. *Mining frequent patterns without candidate generation*. In Proc. of the ACM SIGMOD Int'l Conf. on Management of Data, Dallas, Texas, USA, 2000.

[Han and Kamber, 2001] Han, J.; Kamber, M. *Data Mining - Concepts and Techniques*. Morgan Kaufmann, Nova York, 1a edição, 2001.

[Hilderman and Hamilton, 1999] Hilderman, R.J.; Hamilton, H.J. *Knowledge discovery and interestingness measures: a survey*. Technical Report CS 99-04, Department of Computer Science, University of Regina, Canada, 1999.

[Hipp et al, 2000] Hipp, J.; Güntzer, U.; Nakhaeizadeh, G. *Algorithms for Association Rule Mining - A General Survey and Comparison.* SIGKDD Explorations, vol. 2, 2000. p. 58-64.

[Hong et al, 1999] Hong, T.Z.; Kuo, C.S.; Chi, S.C. *A data mining algorithm for transaction data with quantitative values*. In The Eighth International Fuzzy Systems Association World Congress, Taipei, Taiwan, 1999. p. 1-34.

[Houtsma and Swami, 1993] Houtsma, M.; Swami, A. *Set-oriented mining of association rules*. Research Report, IBM Almaden Research Center, 1993.

[Hsu et al, 2002] Hsu, W.; Zhang, J.; Lee, M.L. *Image Mining: Trends and Developments.* Journal of Intelligent Information System (JISS) : Special Issue on Multimedia Data Mining, vol. 2002.

[Hsu et al, 2003] Hsu, W.; Dai, J.; Lee, M.L. *Mining viewpoint patterns in image databases*. In The ninth ACM SIGKDD international conference on Knowledge discovery and data mining, Washington, D.C., USA, 2003. p. 553-558.

[Hufford, 1996] Hufford, D. *Data Warehouse Quality Data Management Review.* The Premier Publication for Business Intelligence and Analytics (http://, vol. February-March 1996 1996.

[Kantarcioglu and Clifton, 2002] Kantarcioglu, M.; Clifton, C. *Privacy-preserving Distributed Mining of Association Rules on Horizontally Partitioned Data*. In The ACM SIGMOD Workshop in Research Issues on Data Mining and Knowledge Discovery, Madison, Wisconsin, USA, 2002.

[Kimball, 1996a] Kimball, R. *The Data Warehouse Toolkit - Practical Techniques for Building Dimensional Data Warehouses*. Jonh Wiley Professional, 1996a.

[Kimball, 1996b] Kimball, R. *Data Warehouse Architect*, in *DBMS and Internet System, http://www.dbmsmag.com/9603d05.html*. 1996b. p. 14-19.

[Kimball, 1997] Kimball, R. *A dimensional modeling manifesto*, in *DBMS Magazine, http://www.dbmsmag.com/9708d15.html*. 1997. p. 58-70.

[Klemettinen et al, 1994] Klemettinen, M.; Mannila, H.; Ronkainen, P.; Toivonen, H.; Verkamo, A. *Finding Interesting Rules from Large Sets of Discovered Association Rules*. In Proc. of the Third International Conference on Information and Knowledge Management (CIKM'94), Gaithersburg, Maryland, USA, 1994.

[Kuok et al, 1998] Kuok, C.M.; Fu, A.; Wong, M.H. *Mining fuzzy association rules in databases*, in *ACM SIGMOD Record*. 1998. p. 41-46.

[Liu et al, 1999] Liu, H.; Lu, H.; Feng, L.; Hussain, F. *Efficient Search of Reliable Exceptions*. In Proc. of the Third Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'99), Beijing, China, 1999.

[Nijssen and Kok, 2001] Nijssen, S.; Kok, J. *Faster Association Rule for Multiple Relations*. In In Proc. of the 7th International Joint Conference on Artificial Intelligence (IJCAI 2001), Seattle, USA, 2001. p. 891-896.

[Omiecinski, 2003] Omiecinski, E. *Alternative interest measures for mining associations in databases.* IEEE Transactions on Knowledge and Data Engineering, vol. 15 no. 1, January/February 2003.

[Ordonez and Omiecinski, 1999] Ordonez, C.; Omiecinski, E. *Discovering Association Rules based on Image Content*. In IEEE Forum on Research and Technology Advances in Digital Libraries, 1999. p. 38-49.

[Otey et al, 2003] Otey, M.; Veloso, A.A.; Wang, C.; Parthasarathy, S.; Meira, W. *Mining for Frequent Itemsets in Distributed and Dynamic Databases*. In Proc. of the Third IEEE Int'l Conf. on Data Mining, Melbourne, Florida, USA, 2003.

[Pohlig and Hellman, 1978] Pohlig, S.C.; Hellman, M.E. *An improved algorithm for computing logarithms over GF(p) and its cryptographic significance.* IEEE Transactions on Knowledge and Data Engineering, vol. 1, 1978. p. 106-110.

[Ribeiro and Vieira, 2004] Ribeiro, M.X.; Vieira, M.T.P. *A New Approach for Mining Association Rules in Data Warehouses (accepted paper)*. In 6th International Conference On Flexible Query Answering Systems, Lyon, France, 2004.

[Roden et al, 1999] Roden, J.; Burl, M.C.; Fowlkes, C. *Mining for Image Content*. In Systemics, Cybernetics, and Informatics / Information Systems: Analysis and Synthesis, Orlando, FL, EUA, 1999.

[Savarese et al, 1995] Savarese, A.; Omiecinski, E.; Navathe, S. *An Efficient Algorithm for Mining Association Rules in Large Databases*. In Proc. of the 21st Conf. on Very Large Databases (VLDB'95), 1995.

[Silva, 2002] Silva, D.R. *Uma Ferramenta para Descoberta de Conhecimento com Suporte de Data Warehousing e sua Aplicação para Acompanhamento do Aluno em Educação a Distância*. Dissertação de Mestrado. Departamento de Computação, Universidade Federal de São Carlos, São Carlos, Brasil, 2002. p. 108.

[Srikant and Agrawal, 1996] Srikant, R.; Agrawal, R. *Mining Quantitative Association Rules in Large Relational Tables*. In In Proc. of the1996 ACM SIGMOD International Conference on Management of Data, Montreal, Canada, 1996. p. 1-12.

[Tan et al, 2002] Tan, P.N.; Kumar, V.; Srivastava, J. *Selecting the right interestingness measure for association patterns*. In Eight ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002.

[Thuraisingham, 2002] Thuraisingham, B. *Data mining, national security, privacy and civil liberties.* ACM SIGKDD Explorations Newsletter, vol. 4, 2002. p. 1-5.

[Vaidya and Clifton, 2003] Vaidya, J.; Clifton, C. *Privacy Preserving K-Means Clustering over Vertically Partitioned Data*. In The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, 2003. p. 206-215.

[Vaidya and Clifton, 2002] Vaidya, J.S.; Clifton, C. *Privacy Preserving Association Rule Mining in Vertically Partitioned Data*. In Proc. of the 8th ACM SIGKDD Int'l Conf. on Knowlegde Discovery and Data Mining, Edmonton, Alberta, Canada, 2002.

[Vassiliadis and Sellis, 1999] Vassiliadis, P.; Sellis, T. *A Survey of Logical Models for OLAP Databases.* ACM SIGMOD Record, vol. 28, 1999. p. 64-69.

[Veloso et al, 2003] Veloso, A.A.; Meira, W.; Parthasarathy, S.; Carvalho, M.B. *Efficient, Accurate and Privacy-Preserving Data Mining for Frequent Itemsets in Distributed Databases*. In Proc. of the 18th Brazilian Symposium on Databases, Manaus, Amazonas, Brazil, 2003.

[Yao, 1986] Yao, A.C. *How to generate and exchange secrets*. In 27th IEEE Symposium on Foundations of Computer Science, 1986. p. 162-167.

[Zaïane et al, 1998] Zaïane, O.R.; Han, J.; Li, Z.-N.; Hou, J. *Mining Multimedia data*. In The 1998 conference of the Centre for Advanced Studies on Collaborative research, Toronto, Ontario, Canada, 1998. p. 24 - 41.

[Zaki et al, 1997] Zaki, M.; Parthasarathy, S.; Ogihara, M.; Li, W. *New algorithms for fast discovery of association rules*. In Proc. of the ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining, Newport Beach, CA, USA, 1997.

[Zhang et al, 2003] Zhang, S.; Wu, X.; Zhang, C. *Multi-Database Mining.* IEEE Computational Intelligence Bulletin, vol. 2, 2003. p. 5-13.

[Zhong et al, 2001] Zhong, N.; Ohshima, M.; Yao, Y.Y.; Ohsuga, S. *Interestingness, peculiarity, and multi-database mining*. In IEEE International Conference on Data Mining, 2001. p. 566-573.

Apêndice A - Detalhamento do Algoritmo *Connection*

O algoritmo *Connection* é apresentado na **Figura A.1**.

Algoritmo: *Connection*
Entrada: O conjunto de tabelas fato *F*, valores de *suporte# mínimo* s, *confiança# mínima* c, e *peso mínimo* p
Saída: O conjunto *C* de regras de associação *multifatos* mineradas

Função *Connection*(F,s,c,p**) {**
1. $T=\{ ID(b_i)|ID(b_i)=ID(b_j)$, *tal que* $b_i \in F_i$ *e* $b_j \in F_j$ *para* $1 \leq i \leq |F|$ *e* $1 \leq j \leq |F| \}$
2. M=ϕ
3. F=ϕ
4. para cada tabela fato $F_i \in F$ **faça {**
5. M_i = *buildMFPTree*(F_i,*T*))
6. R_i = *findPerRelation*(M_i)
7. $M = M \cup M_i$
8. $R = R \cup R_i$
9. }
10. *G = findGlobalFrequent(M,R);*
11. *C* = *genMultifactRules*(*G*);
11. **return C**
12.}

Figura A.1: *Algoritmo Connection*

Na **linha 1**, o conjunto de segmentos é encontrado. Para isso, todas as tabelas fato $F_i \in F$ são percorridas, o identificador $ID(b_i)$ de cada **bloco b_i**, que está presente em todas as tabelas fato $F_i \in F$, é adicionado a *T*. *T* representa o conjunto de segmentos, possuindo os identificadores de todos os segmentos.

Nas **linhas 2 e 3**, o algoritmo *Connection* inicializa dois conjuntos, *M* e *R*, com vazio. Esses conjuntos são usados respectivamente para armazenar as *MFP-trees* M_i e o conjunto de *itemsets freqüentes#* R_i de todas as tabelas fato F_i.

No loop das **linhas 4 a 9,** todas as tabelas fato F_i são varridas. A *MFP-tree* M_i da tabela fato F_i é construída através da função *buildMFPTree* (**linha 5**). Essa *MFP-tree* é usada para determinar o conjunto R_i, que é um super-conjunto de todos os *itemsets freqüentes#* da tabela fato F_i, através da função *findPerRelation* **(linha 6)**. Nas **linhas 7 e**

8, a *MFP-tree* M_i e o conjunto R_i de cada tabela fato são adicionados respectivamente aos conjuntos *M* e *R*. Na **linha 10**, esses conjuntos são usados pela função *findGlobalFrequent* para determinar o conjunto *G* de *itemsets* globais *freqüentes#* e, na **linha 11**, o conjunto *G* é usado pela função *genMultifactRules* para gerar o conjunto *C* de regras de associação *multifatos fortes#*, que é o conjunto retornado pelo algoritmo.

As funções usadas pelo algoritmo *Connection* são detalhadas nas subseções seguintes.

A Função buildMFPtree

A função *buildMFPTree* constrói a *MFP-tree* M_i de uma tabela fato F_i, da seguinte maneira:

1. *Inicialização da MFP-tree:* Os blocos $b_j \in F_i$, tal que $ID(b_j) \in T$, são percorridos. Para cada item encontrado, é calculado o **peso** e o ***suporte#*** do mesmo, e o conjunto *L* de itens *freqüentes#* é determinado. Os itens do conjunto *L* são ordenados na ordem decrescente de seus valores de ***suporte#***. A tabela *Header* é criada com uma entrada para cada item de *L*, onde o campo *peso* é inicializado com o valor do **peso** do item e o campo *tidlist* é inicializado com o valor nulo.
2. *Preenchimento da FP-tree:* Esta etapa é semelhante à construção da *FP-tree* do algoritmo *FP-Growth* [Han et al, 2000]. A tabela fato F_i é percorrida pela segunda vez. O nó raiz *Z* é criado e seus campos recebem o valor *null*. Para cada transação processada, somente seus itens *freqüentes#*, ordenados na ordem de *L*, são considerados. Para a primeira transação a ser processada, um ramo na árvore com um novo nó para cada item da transação é criado, e o valor do campo *contador* de cada nó é inicializado com o valor 1. No processamento dos itens da próxima transação, é verificado se a transação possui um prefixo já existente na *MFP-tree*; se sim os contadores dos elementos desse prefixo são incrementados, se não, um novo nó é adicionado à árvore. Para o preenchimento da *MFP-tree* M_i, somente são processados os dados provenientes dos blocos $b_j \in F_i$, tal que $ID(b_j) \in T$, diferentemente da *FP-tree* em que todos os dados de F_i são considerados. Além disso, $ID(b_j)$ é adicionado a *tidlist* de todos os elementos da tabela *Header* que correspondem aos itens encontrados no bloco b_j. Assim, após a construção da

MFP-tree, a *tidlist* de um elemento h_j da tabela *Header*, referente a um item i_j, possui todos os identificadores dos segmentos em que i_j ocorreu.

Considere o conjunto de tabelas fato $F= \{H_1,H_2\}$ apresentado na **Figura A.2**. Para facilitar a discussão, as **Figuras A.3** e **A.4** mostram os blocos das tabelas fato H_1 e H_2 respectivamente, que são as transações agrupadas pelo atributo *atrib1*. Os blocos cujo fundo está em cinza, não pertencem ao conjunto de segmentos, portanto para o cálculo do *suporte#* e *confiança#* eles são desconsiderados. Esses blocos somente são considerados para determinar se um item satisfaz ou não o *peso mínimo*. Os segmentos, cujos identificadores (valores do atributo $atrib_1$) formam o conjunto *T*, são apresentados na **Figura A.5**.

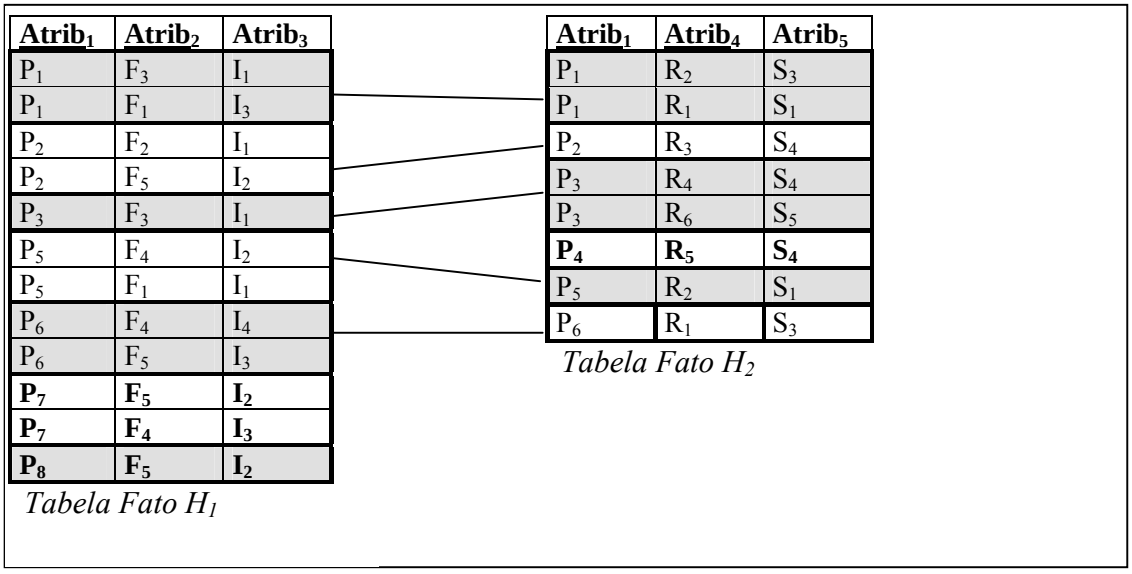


Figura A.2: *Tabelas Fato H_1 e H_2*

{< P_1, F_3, I_1>, < P_1,F_1, I_3>}
{< P_2, F_2, I_1>,< P_2, F_5, I_2>}
{< P_3, F_3, I_1>}
{< P_5, F_4, I_2>, < P_5, F_1, I_1>}
{< P_6, F_4, I_4>, < P_6, F_5, I_3>}
{< P_7, F_5,I_2>,< P_7, F_4, I_3>}
{< P_8, F_5, I_2>}

Figura A.3: *Blocos de H_1*

{< P_1, R_2, S_3 >, < P_1, R_1, S_1>}
{< P_2, R_3, S_4 >}
{< P_3, R_4, S_4 >,< P_3, R_6,S_5>}
{< P_4, R_5, S_4 >}
{< P_5, R_2, S_1 >}
{< P_6,R_1, S_3 >}

Figura A.4: *Blocos de H_2*

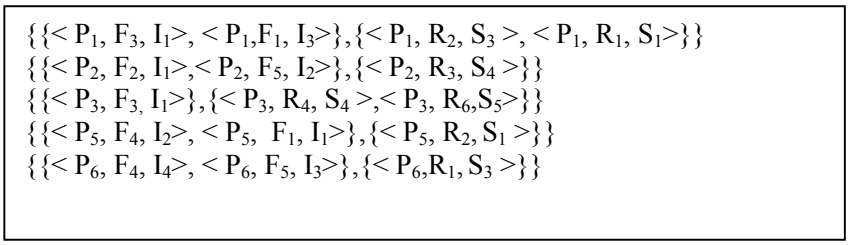


Figura A.5: *Segmentos para F= {H1, H2}*

Inicialmente, a *MFP-tree* M_1 de H_1 é construída. No primeiro passo, a tabela fato H_1 é percorrida; o *suporte#* e o valor do peso de cada item são determinados. Então, é obtido o conjunto *L* de itens que satisfazem o *suporte# mínimo s* e o *peso mínimo p*. A **Figura A.5**, apresentada anteriormente, mostra um total de 5 segmentos, então para um item satisfazer o *suporte# mínimo s*=0.4, ele deve ocorrer no mínimo em 5×0.4=2 segmentos. Do mesmo modo, para um item satisfazer o *peso mínimo p=0.6*, no mínimo *60%* das ocorrências desse item devem ser em segmentos. Para facilitar a compreensão, na **Tabela A.1** são apresentados os itens de H_1, seus números de *ocorrência* e seus pesos. O item I_3 ocorre em 3 blocos (P_1, P_6 e P_7), no entanto, somente 2 blocos (P_1, P_6) pertencem ao conjunto de segmentos, portanto o peso de I_3 é 2/3. Os itens I_2 e I_5 apresentam os valores de *suporte#* e peso iguais respectivamente a 2/5 e 2/4, satisfazendo o *suporte# mínimo (s=0.4)*, porém não satisfazendo o *peso mínimo (p=0.6)*. Então esses itens não são adicionados ao conjunto *L*. Os itens do conjunto *L* são ordenados na ordem decrescente de seu *suporte#*. Assim, o conjunto *L* da tabela H_1 é :

$L = \{I_1, I_3, F_1, F_3, F_4\}$

A tabela *Header* é criada, com os itens do conjunto *L*, conforme mostrado na **Tabela A.2**.

item	ocorrências	peso
I_1	4	1
I_2	2	¾
I_3	2	⅔
I_4	1	1
F_1	2	1
F_2	1	1
F_3	2	1
F_4	2	⅔
F_5	2	¾

Tabela A.1: *Itens da tabela H_1*

item	peso	*tidlist*
I_1	1	null
I_3	0.66	null
F_1	1	null
F_3	1	null
F_4	0.66	null

Tabela A.2: *Estado inicial da tabela Header de H_1*

Para facilitar o entendimento das próximas etapas do algoritmo. Os blocos de H_1 pertencentes ao conjunto de segmentos, com os itens *freqüentes#* ordenados na ordem de *L*, são apresentados na **Figura A.6**.

P_1:< I_1,F_3>, <I_3,F_1>
P_2:< I_1>
P_3:< I_1, F_3>
P_5:<F_4,>, <I_1,F_1>
P_6:<F_4>, < I_3 >

Figura A.6: Itens *freqüentes#* de H_1 na ordem de *L*

Na **Figura A.6** o identificador de cada bloco é colocando antes de seus itens.

O algoritmo inicia processando o primeiro bloco, que é o bloco identificado por P_1. Ao ler a primeira transação de P_1, cria um novo nó I_1 como filho do nó raiz, e cria um novo nó F_3 como filho de I_1. O campo *contador* de cada nó criado é sempre inicializado com o valor 1. Ao ler a segunda transação de P_1, o algoritmo cria um novo nó I_3 como filho do nó raiz, e cria um novo nó F_1 como filho de I_3. Além disso, P_1 é adicionado na *tidlist* dos elementos da tabela *Header* correspondentes a I_1, F_3, I_3 *e* F_1. O algoritmo ao ler o segundo bloco identificado por P_2, verifica que o prefixo I_1 já existe na árvore. Então, o campo *contador* do nó I_1, é incrementado em uma unidade, e P_2 é adicionado na *tidlist* do elemento da tabela Header correspondente a I_1. Ao processar a primeira transação de P_3, o prefixo *"*I_1,F_3*"*, já existente na árvore é aproveitado. Os campos *contador* dos nós de I_1 e F_3, são incrementados em uma unidade, e P_3 é adicionado na *tidlist* dos elementos da tabela

Header correspondente aos itens I_1 e F_3. Esse procedimento se repete, até todos os blocos da tabela H_1, que pertencem ao conjunto de segmentos, serem varridos. A **Figura A.7** mostra a *MFP-tree* M_1 resultante desse procedimento.

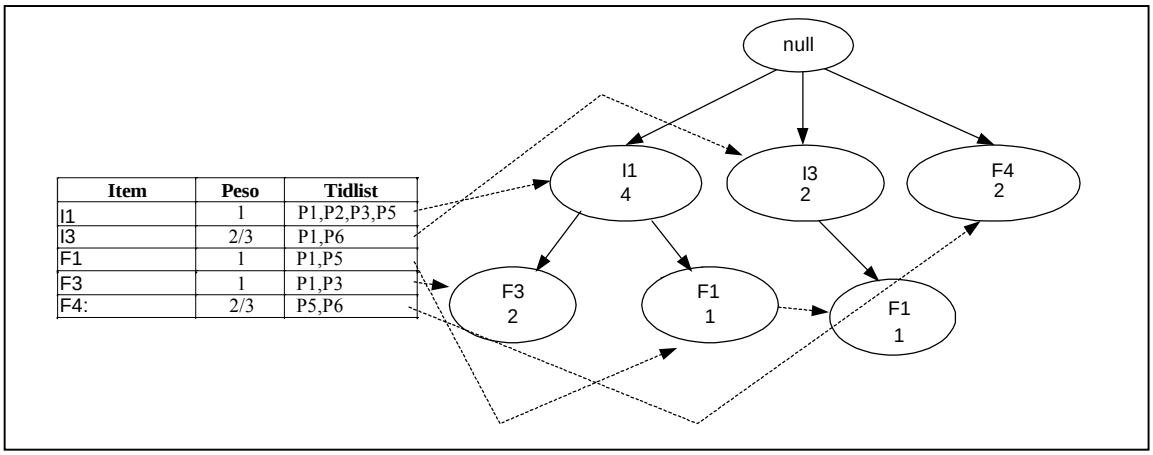


Figura A.7: *MFP-tree* M_1

Esse mesmo procedimento é aplicado para a construção da *MFP-tree* da tabela fato H_2. A ***Figura A.8*** mostra a *MFP-tree* M_2 construída a partir dos dados da tabela H_2.

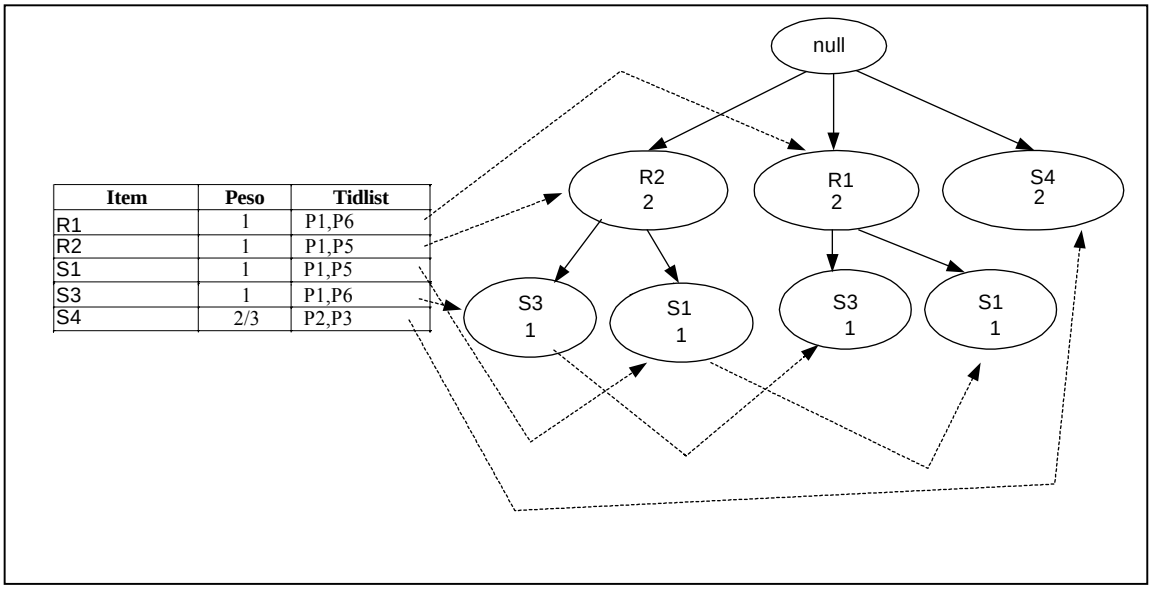


Figura A.8: *MFP-tree* M_2

A função *findPerRelation*

A função *findPerRelation* do algoritmo *Connection* é responsável por percorrer uma *MFP-tree* M_i e encontrar o conjunto R_i, de todos os *itemsets freqüentes#* de uma tabela fato F_i. A função *findPerRelation* é descrita a seguir.

Seja *ntransações(X)* e *nsegmentos(X)*, respectivamente, o número de ocorrências do *itemset X* em **transações** dos segmentos e o número de ocorrências do *itemset X* nos segmentos. Seja o *Prefixo(x)*, o conjunto dos itens antecessores de um nó *n* que representa o item *x*. A função *findPerRelation* varre a *MFP-tree* M_i e, para cada item *x* da tabela *Header*, encontra todos os *itemsets* $X=\{x\}\cup Prefixo(x)$, tal que $ntransações(X) \geq s\times|T|$, onde *s* é o *suporte# mínimo* estabelecido e $|T|$ o número de segmentos, e adiciona em R_i todos os *itemsets X*. Como $ntransações(X) \geq nsegmentos(X)$ e um *itemset X* da *MFP-tree* M_i é *freqüente#* se $nsegmentos(X) \geq s\times|T|$, tem-se que o conjunto R_i, retornado pela função *findPerRelation*, é um super-conjunto, que contém todos os *itemsets freqüentes#* da tabela fato F_i.

Ao percorrer as *MFP-tree* das tabelas fato H_1 e H_2 (**Figuras A.7 e A.8**), são adicionados ao conjunto F_i, todas as combinações de *x* com *Prefixo(x),* para os *itemsets* $X=\{x\}\cup Prefixo(x)$ cujo $ntransações(X) \geq 0.4\times5=2$, onde 5 é o numero de segmentos e 0.4, o mínimo suporte. Os itens *x*, juntamente com seus prefixos e os *itemsets* que são adicionados a *F*, a partir das *MFP-trees* M_1 e M_2, são apresentados nas **Tabela A.3 e A.4**. Cada *itemset X* é representado pela lista de seus itens seguida pelo valor de *ntransações(X)*.

Item *x*	*Prefixo(x)*	*
F_4:2	{}	{F_4:2}
F_3:2	{I_1: 2}	{F_3:2}; {F_3,I_1:2}
F_1:2	{I_1:1}, {I_3 1}	{F_1:2}
I_3:4	{}	{I_3: 2}
I_1:4	{}	{I_1:4}

***** *Itemsets* adicionados a F_1

Tabela A.3: *Conjunto F_1 obtido a partir de H_1*

Item *x*	*Prefixo(x)*	*
S_4:2	{}	{S_4:2}
S_3:2	{R_2:1}, {R_1:1}	{S_3:2}
S_1:2	{R_2:1}, {R_1:1}	{S_1:2}
R_2:2	{}	{R_2:2}
R_1:2	{}	{R_1:2}

***** *Itemsets* adicionados a F_2

Tabela A.4: *Conjunto F_2 obtido a partir de H_2*

Observe que o próprio item (itemset-1) é retornado no conjunto de itemsets freqüentes#.

A função *findGlobalFrequent*

Os conjuntos *R* e *M* são usados para determinar o conjunto global de *itemsets freqüentes#* , através da função *findGlobalFrequent*, que é descrita a seguir:

Seja *F* um conjunto de tabelas fato, onde $n = |F|$. Seja $F_i \in F$ uma tabela fato, onde $1 \leq i \leq n$. Seja R_i o conjunto com todos *itemsets freqüentes#* da tabela fato F_i retornado na etapa anterior do algoritmo. Seja r_i um *itemset* pertencente a R_i. A *tidlist* de um *itemset* r_i é obtida interseccionado as *tidlists* de cada elemento que forma o *itemset* r_i. O processo de determinar o conjunto global dos *itemsets freqüentes# G* consiste em encontrar todos os *itemsets* $g = r_1 \cup r_2 \ldots \cup r_n$, tal que $|r_1.tidlist \cap r_2.tidlist,\ldots, \cap r_n.tidlist|$ satisfaça o *suporte# mínimo s*.

Considere o conjunto de tabelas fato $R=\{H_1,H_2\}$ apresentado na **Figura A.2**. Interseccionando as *tidlists* dos *itemsets* (ver tabelas *Header* das **Figuras A.7** e **A.8**) pertencentes a F_1 e F_2, que são apresentadas respectivamente nas **Tabelas A.3 e A.4**, é obtido o conjunto global de *itemsets freqüentes#*, que é apresentado na **Tabela A.5**.

Itemset freqüente#	Segmentos
$\{I_1, R_2\}$	P_1, P_5
$\{I_1, S_1\}$	P_1, P_5
$\{I_1, S_4\}$	P_2, P_3
$\{I_3, R_1\}$	P_1, P_6
$\{I_3, S_3\}$	P_1, P_6
$\{F_1,R_2\}$	P_1, P_5
$\{F_1,S_1\}$	P_1, P_5

Tabela A.5: *Conjunto global de itemsets freqüentes# G, para* $F=\{H_1, H_2\}$.

A função *genMultifactRules*

A função *genMultifactRules* é reponsável pela fase final da mineração de regras de associação *multifatos*. Essa função gera o conjunto de regras *fortes#* a partir do conjunto global de *itemsets freqüentes# G*, ou seja, ela gera todas as combinações *X* e *Y*, onde *X* e *Y* são *itemsets*, tal que a regra $X \rightarrow Y$ satisfaça a *confiança# mínima* e

$REL(X) \cap REL(Y) = \phi$, onde $REL(X)$ representa o conjunto de tabelas fato nas quais os itens do *itemset X* aparecem.

Considere os valores de *suporte# mínimo s*=0.4, *confiança# mínima c*=0.8 e *peso mínimo p*=0.6. As regras mineradas para o conjunto de tabelas fato $F = \{H_1, H_2\}$ são apresentadas abaixo, onde *s#* e *c#* indicam respectivamente o valor de *suporte#* e valor de *confiança#* . O valor de peso de cada item *i* da regra é indicado em p(*i*).

$R_2 \rightarrow I_1$ s#=0.4, c#=1, $p(R_2) = 1$, $p(I_1) = 1$

$S_1 \rightarrow I_1$ s#=0.4, c#=1, $p(S_1)=1$, $p(I_1)=1$

$S_4 \rightarrow I_1$ s#=0.4, c#=1 $p(S_4)=0.66$, $p(I_1)=1$

$I_3 \rightarrow R_1$ s#=0.4, c#=1 $p(I_3)=0.66$, $p(R_1) = 1$

$R_1 \rightarrow I_3$ s#=0.4, c#=1 $p(R_1)=1$, $p(I_3) = 0.66$

$I_3 \rightarrow S_3$ s#=0.4, c#=1, $p(I_3)=0.66$, $p(S_3)=1$

$S_3 \rightarrow I_3$ s#=0.4, c#=1, $p(S_3)=1$, $p(I_3)= 0.66$

$F_1 \rightarrow R_2$ s#=0.4, c#=1, $p(F_1) =1$, $p(R_2)=1$

$R_2 \rightarrow F_1$ s#=0.4, c#=1, $p(R_2)=1$, $p(F_1)=1$

$F_1 \rightarrow S_1$ s#=0.4, c#=1, $p(F_1)=1$, $p(S_1)=1$

$S_1 \rightarrow F_1$ s#=0.4, c#=1, $p(S_1)=1$, $p(F_1)=1$

Considere que a tabela fato H_1 tenha dados sobre pessoas portadoras do vírus de HIV e que a tabela fato H_2 tenha informações sobre seus familiares[1]. Considere também, que o atributo comum (*atrib1*) entre essas duas tabelas fato seja a identificação das pessoas portadoras do vírus HIV. Seja R_1 o item *"fumante=sim"*, pertencente à tabela fato H_2, referente aos dados dos **familiares** dos portadores do vírus HIV, e I_3 o item *"doenças respiratórias=sim"*, pertencente à tabela fato H_1, referente aos dados dos portadores do vírus HIV. Então a regra $R_1 \rightarrow I_3$ corresponde a regra:

"fumante=sim" → *"doenças respiratórias=sim"*

A interpretação da regra acima é: "os portadores do vírus HIV, que possuem familiares fumantes, tendem a apresentar doenças respiratórias".

[1] Esse exemplo foi baseado Banco de Dados da pesquisa "Comportamento Sexual da População Brasileira e Percepções do HIV/Aids" disponível no site http://www.aids.gov.br/