UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"

Pós-Graduação em Ciência da Computação

Fernando Takeshi Oyama

Mineração multirrelacional de regras de associação em grandes bases de dados

UNESP

2010

Fernando Takeshi Oyama

Mineração multirrelacional de regras de associação em grandes bases de dados

Orientador: Prof. Dr. Carlos Roberto Valêncio

Dissertação apresentada para obtenção do título de Mestre em Ciência da Computação, Área de Concentração em Sistemas de Computação, junto ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.

UNESP

2010

Fernando Takeshi Oyama

Mineração multirrelacional de regras de associação em grandes bases de dados

Dissertação apresentada para obtenção do título de Mestre em Ciência da Computação, Área de Concentração em Sistemas de Computação, junto ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.

BANCA EXAMINADORA

Prof. Dr. Carlos Roberto Valêncio
IBILCE/UNESP – São José do Rio Preto
Orientador

Profª. Drª. Cristina Dutra de Aguiar Ciferri
ICMC/USP – São Carlos

Profª. Drª. Rogéria Cristiane Gratão de Souza
IBILCE/UNESP – São José do Rio Preto

São José do Rio Preto, 22 de Fevereiro de 2010.

AGRADECIMENTOS

A Deus, pelo dom da vida e por ter me dado forças para concluir este trabalho.

A minha esposa Andressa, pelo carinho, confiança, paciência e companheirismo despendido ao meu favor, bem como pelo apoio incondicional, o qual foi determinante para a conclusão desta jornada.

A meu orientador, Prof. Dr. Carlos Roberto Valêncio, pelo apoio, estímulo e exemplo na vida acadêmica.

A meus pais, pela educação que me proporcionaram.

A minha irmã, a meus sogros e meus cunhados, pelo estímulo constante.

Aos Profs. Drs. Rogéria Cristiane Gratão de Souza e Mário Luiz Tronco, pelas contribuições em ocasião do Exame Geral de Qualificação.

Aos Profs. Drs. Cristina Dutra de Aguiar Ciferri e Rogéria Cristiane Gratão de Souza, por terem aceitado examinar esse trabalho por ocasião da Defesa, bem como pelas contribuições dadas.

Aos colegas, professores e funcionários do Programa de Pós-Graduação em Ciência da Computação do IBILCE – UNESP.

À Capes, pelo auxílio financeiro proporcionado durante o período de vigência da bolsa.

À minha esposa,
com muito carinho,
gratidão e amor.

"Bem-aventurado o homem que acha sabedoria, e o homem que adquire conhecimento; Porque é melhor a sua mercadoria do que artigos de prata, e maior o seu lucro que o ouro mais fino."
(Provérbios 3:13-14)

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CGT	*Candidate Generate-and-Test*
fDMMR	Ferramenta de data mining multirrelacional
GB	*Gigabyte*
HC	Hospital do Câncer
IL	*ItemList*
ILP	*Inductive Logic Programming*
IM	*ItemMap*
KB	*Kilobyte*
KDD	*Knowledge Discovery in Databases*
MB	*Megabyte*
MDMR	Mineração de dados multirrelacional
minsup	Suporte mínimo
MRDM	*Multi-relational data mining*
PG	*Pattern-growth*
SGBD	Sistema gerenciador de banco de dados
SIVAT	Sistema de Vigilância de Acidentes de Trabalho

RESUMO

O crescente avanço e a disponibilidade de recursos computacionais viabilizam o armazenamento e a manipulação de grandes bases de dados. As técnicas típicas de mineração de dados possibilitam a extração de padrões desde que os dados estejam armazenados em uma única tabela. A mineração de dados multirrelacional, por sua vez, apresenta-se como uma abordagem mais recente que permite buscar padrões provenientes de múltiplas tabelas, sendo indicada para a aplicação em bases de dados relacionais. No entanto, os algoritmos multirrelacionais de mineração de regras de associação existentes tornam-se impossibilitados de efetuar a tarefa de mineração em grandes volumes de dados, uma vez que a quantia de memória exigida para a conclusão do processamento ultrapassa a quantidade disponível. O objetivo do presente trabalho consiste em apresentar um algoritmo multirrelacional de extração de regras de associação com o foco na aplicação em grandes bases de dados relacionais. Para isso, o algoritmo proposto, MR-RADIX, apresenta uma estrutura denominada *Radix-tree* que representa comprimidamente a base de dados em memória. Além disso, o algoritmo utiliza-se do conceito de particionamento para subdividir a base de dados, de modo que cada partição possa ser processada integralmente em memória. Os testes realizados demonstram que o algoritmo MR-RADIX proporciona um desempenho superior a outros algoritmos correlatos e, ainda, efetua com êxito, diferentemente dos demais, a mineração de regras de associação em grandes bases de dados.

Palavras-chave: MR-RADIX, mineração de dados multirrelacional, regras de associação, mineração de *itemsets* frequentes, base de dados relacional

ABSTRACT

The increasing spread and availability of computing resources make feasible storage and handling of large databases. Traditional techniques of data mining allows the extraction of patterns provided that data is stored in a single table. The multi-relational data mining presents itself as a more recent approach that allows search patterns from multiple tables, indicated for use in relational databases. However, the existing multi-relational association rules mining algorithms become unable to make mining task in large data, since the amount of memory required for the completion of processing exceed the amount available. The goal of this work is to present a multi-relational algorithm for extracting association rules with focus application in large relational databases. For this the proposed algorithm MR-RADIX presents a structure called Radix-tree that represents compressly the database in memory. Moreover, the algorithm uses the concept of partitioning to subdivide the database, so that each partition can be processed entirely in memory. The tests show that the MR-RADIX algorithm provides better performance than other related algorithms, and also performs successfully, unlike others, the association rules mining in large databases.

Keywords: MR-RADIX, multi-relational data mining, association rules, frequent itemsets mining, relational database

Capítulo 1
Introdução

1.1 Considerações iniciais

A consolidação e a constante evolução dos sistemas de gerenciamento de dados têm oferecido um suporte robusto à captação e à manipulação dos dados provenientes das mais diversas áreas de aplicação. Juntamente com o advento desses sistemas, tem-se visto sofisticações em nível de *hardware*, o que torna perfeitamente possível o armazenamento eficiente de grandes volumes de dados, tarefa antes vista como desafiadora e custosa, tanto em termos computacionais quanto em termos econômicos. Não obstante, essa disponibilidade de recursos para o armazenamento de dados acabou gerando repositórios não-analisáveis, de modo que há oferta ou riqueza de dados, mas pobreza de informações (HAN; KAMBER, 2006).

A mineração de dados surgiu como um campo de estudo visando o desenvolvimento de ferramentas e técnicas para a prospecção de grandes repositórios de dados, com o intuito de obter informações novas, valiosas, não-triviais e implicitamente existentes (KANTARDZIC, 2003). A mineração pode ser vista como a mais importante etapa de um processo mais amplo denominado Descoberta de Conhecimento em Bases de Dados (*Knowledge Discovery in Database* - KDD). O KDD apresenta ainda as etapas de pré-processamento, que visam preparar e selecionar os dados, e as etapas de pós-processamento, que auxiliam na compreensão dos padrões obtidos (FAYYAD; PIATETSKY-SHAPIRO; PADHRAIC, 1996).

Os algoritmos tradicionais de mineração de dados realizam o processamento levando em conta que os dados estão dispostos em uma única estrutura, geralmente,

um arquivo ou uma tabela. Essa limitação não possibilita que tais algoritmos sejam utilizados eficientemente em contextos nos quais os dados encontram-se estruturados como, por exemplo, uma base de dados relacional que se constitui de diversas tabelas semanticamente relacionadas (KNOBBE, 2004). Um procedimento que é comumente realizado é a geração de uma tabela global que reúne, por meio de operações de junção, as diversas tabelas existentes. A dificuldade nesse caso é que a compressão ou junção de dados provenientes de várias tabelas demanda um alto custo computacional e, principalmente, pode acarretar a perda de informações ou a geração de imprecisões nos padrões extraídos (DZEROSKI; RAEDT; WROBEL, 2003).

A mineração de dados multirrelacional, por sua vez, é uma abordagem mais recente, que surgiu na tentativa de propor técnicas que supram as limitações dos algoritmos tradicionais. A grande vantagem dessa abordagem é justamente a possibilidade de extrair conhecimento proveniente de múltiplas tabelas de modo direto, sem a necessidade da operação de junção de dados (DZEROSKY, 2003). Com essa aplicação, os algoritmos multirrelacionais ampliam o leque de aplicações, uma vez que se gera a possibilidade de mineração em situações nas quais é importante a manutenção da estrutura ou relacionamento das múltiplas tabelas devido a sua relevância semântica (PAGE; CRAVEN, 2003; HABRARD; BERNARD; JACQUENET, 2003).

1.2 Motivação e escopo

As bases de dados relacionais são amplamente utilizadas pelas mais variadas aplicações (GUO; VIKTOR, 2005), tornando-as fontes de inestimáveis conhecimentos ainda ocultos e implícitos que, devido à impossibilidade de análise humana por conta das suas dimensões, somente serão descobertos com a aplicação de algoritmos de mineração de dados (HAN; KAMBER, 2006).

Os algoritmos tradicionais de mineração de dados, tais como o APRIORI (AGRAWAL; SRIKANT, 1994) e o FP-GROWTH (HAN; PEI; YIN, 2000), buscam por padrões em uma única tabela. Por esse motivo, a aplicação desses algoritmos nas bases de dados fica condicionada à necessidade de que todos os dados estejam disponíveis em uma única relação. Uma forma de se aplicar tais técnicas é a criação

de uma relação universal que é resultado da junção de todas as tabelas. Entretanto, é necessário ressaltar que essa operação pode resultar em uma relação universal cujo tamanho torne inviável a aplicação de técnicas tradicionais. Por esse motivo foi desenvolvida uma forma alternativa, a qual consiste em aplicar tais algoritmos na chamada tabela fato, que é uma tabela central composta de atributos que sumarizam ou agregam informações encontradas nas outras tabelas. Entretanto, esta técnica apresenta como desvantagens a geração de tabelas com muitos atributos e a repetição de dados (KNOBBE *et al*., 1999).

Os algoritmos de mineração de dados multirrelacional, então, surgem como propostas viáveis frente à limitação dos algoritmos tradicionais, tornando possível a extração de padrões provenientes de múltiplas relações de maneira direta e eficiente, sem a necessidade da transferência dos dados para uma única relação (DZEROSKY; RAEDT; WROBEL, 2003; DOMINGOS, 2003). As propostas iniciais de mineração de dados multirrelacional tiveram origem no campo da Programação Lógica Indutiva (*Inductive Logic Programming* - ILP), com técnicas baseadas na representação dos padrões na forma de programas lógicos (KNOBBE *et al*., 1999). Posteriormente, começaram a surgir técnicas voltadas especificamente à mineração em bases de dados relacionais, tais como os algoritmos CONNECTION (RIBEIRO, 2004), MRFP-GROWTH (KANODIA, 2005), GFP-GROWTH (PIZZI, 2006) e CONNECTIONBLOCKQ (GARCIA, 2008).

A maioria das propostas de algoritmos de mineração de dados multirrelacional são, na verdade, extensões dos correspondentes algoritmos tradicionais. Tais extensões nem sempre são uma tarefa trivial, visto que a adequação à abordagem multirrelacional implica diversas alterações nas estruturas de dados ou no próprio funcionamento, de modo que o algoritmo estendido possa representar e, principalmente, manipular e extrair os padrões multirrelacionais. Atualmente, é possível encontrar na literatura propostas de algoritmos multirrelacionais para as principais tarefas de mineração de dados - análise de regras de associação, classificação, análise de agrupamentos (DZEROSKY; RAEDT; WROBEL, 2003), atendendo importantes áreas de aplicação, tais como mecanismos de busca na *web*, *marketing*, biologia molecular, análise de dados de negócios, bioinformática, entre outros (BLOCKEEL; DZEROSKY, 2005).

Os algoritmos de mineração de dados multirrelacional existentes, em especial os de extração de regras de associação - que constituem o foco deste trabalho -, produzem resultados satisfatórios quando submetidos ao processamento de pequenas e médias bases de dados. O grande desafio se configura com a mineração de bases de dados volumosas, que são compostas de milhões de registros. Os algoritmos existentes respaldam sua eficiência na utilização de estruturas de dados em memória que sintetizam e representam a base de dados. Tais estruturas reduzem o acesso a disco e possibilitam utilizar estratégias mais eficientes na busca de padrões, tal como a abordagem *pattern-growth*, inicialmente apresentada pelo algoritmo FP-GROWTH. Cabe ressaltar, no entanto, que a dificuldade, nesse caso, é a alocação de memória suficiente para representar toda a estrutura de dados necessária ao funcionamento desses algoritmos, dadas as dimensões das grandes bases de dados – com tamanhos variando de centenas de *megabytes* até *terabytes* de dados.

Diante do exposto, verifica-se a relevância e a necessidade do estudo voltado à mineração multirrelacional de grandes volumes de dados. Isso porque os algoritmos existentes focam somente a eficiência no processo de extração de padrões multirrelacionais, incrementando suas soluções a fim de que realizem a tarefa no menor tempo possível. Porém, faz-se interessante e relevante propor também soluções que atentem para o fator escalabilidade, de modo a permitir a verificação da viabilidade de aplicação dos algoritmos multirrelacionais em grandes bases de dados relacionais.

1.3 Objetivo

O objetivo do presente trabalho é apresentar um algoritmo multirrelacional de extração de regras de associação com o foco na aplicação em grandes bases de dados relacionais. Ao contrário dos algoritmos multirrelacionais existentes, que partem do pressuposto de que há memória suficiente para a representação e processamento dos padrões, a atual proposta leva em conta a limitação desse recurso em situações nas quais o processamento envolve grande volume de dados. Para lidar com tal restrição, utiliza-se do conceito de partição da base de dados para subdividi-la em unidades cujo tamanho possa ser alocado em memória, viabilizando o seu processamento.

Além disso, o algoritmo de mineração multirrelacional deve apresentar desempenho satisfatório que viabilize a análise das bases de dados em tempo hábil. Para isso, a proposta terá como base o algoritmo PATRICIA-MINE (PIETRACAPRINA; ZANDOLIN, 2003), que realiza a tarefa de extração de padrões associativos com base na estratégia *pattern-growth,* introduzida pelo algoritmo FP-GROWTH (HAN; PEI; YIN, 2000). Nessa proposta, será utilizada uma estrutura de dados mais compacta e eficiente denominada *Patricia-trie* - acrônimo de *Practical Algorithm To Retrieve Information Coded In Alphanumeric* - ou *Radix-tree*, que comprime substancialmente a representação dos dados por meio da redução no número de nós da árvore, permitindo a otimização no uso do espaço em memória.

1.4 Metodologia

O trabalho iniciou-se com o levantamento dos algoritmos tradicionais e multirrelacionais destinados à mineração de regras de associação. Em relação a esses últimos, foram considerados no estudo os algoritmos que extraem padrões de múltiplas tabelas de uma base de dados relacional. Na literatura é possível encontrar diversos trabalhos que efetuam a extração de padrões multirrelacionais por meio de programas lógicos ou de grafos, porém essas vertentes não foram analisadas nesse instante. O estudo focou basicamente os algoritmos multirrelacionais derivados da abordagem tradicional, ou seja, aqueles que estendem o funcionamento dos algoritmos de mineração de uma única tabela para o contexto multirrelacional.

A partir da leitura e do estudo desses algoritmos, foi possível notar que as propostas existentes apresentavam uma limitação na mineração de grandes bases de dados, fato esse que motivou a elaboração do presente trabalho. Para apresentar uma solução razoável para tal limitação, realizou-se, como primeiro passo, um levantamento dos algoritmos tradicionais existentes, com o objetivo de identificar propostas que apresentaram bons resultados quando aplicadas em grandes volumes de dados.

Uma vez escolhido o algoritmo tradicional base, iniciaram-se as fases de projeto e de implementação do algoritmo multirrelacional. Nessas etapas, foram elaboradas as estratégias de representação dos itens frequentes, bem como o processamento e o relacionamento dos padrões provenientes das diversas tabelas.

O algoritmo elaborado pelo presente trabalho foi escrito em linguagem de programação Java, plataforma J2SE, versão 1.6. O ambiente de desenvolvimento integrado utilizado para a codificação e confecção da interface gráfica foi o Netbeans[1], versão 6.7.1. Dentre as razões para a escolha dessa linguagem de programação, destaca-se:

- Linguagem padrão para o desenvolvimento dos trabalhos do grupo de pesquisa;
- Ferramenta de suporte à mineração de dados multirrelacional concebida em trabalho anterior implementado nesta linguagem (OYAMA, 2006);
- Linguagem e ferramentas de desenvolvimento gratuitas;
- Portabilidade e independência de plataforma e de sistema gerenciador de banco de dados (SGBD);
- Existência de boa documentação oficial e não-oficial, tais como fóruns de discussão, tutoriais etc.

Especificamente em relação à ferramenta, denominada fDMMR (OYAMA, 2006), cabe fazer o adendo de que se trata de um recurso que oferece suporte às principais etapas do processo de Descoberta de Conhecimento em Bases de Dados (KDD), na qual foram implementados tanto o algoritmo proposto quanto aqueles destinados aos testes, que serão descritos futuramente. Tal ferramenta foi utilizada como base para a coleta das métricas de desempenho dos algoritmos, caracterizando-se como uma interface para a preparação dos dados, para a aplicação do algoritmo e para a visualização dos resultados. A descrição pormenorizada da ferramenta encontra-se no Apêndice D deste trabalho.

Para a finalização do desenvolvimento e implementação do objetivo desta dissertação, foi realizado um estudo comparativo entre o algoritmo proposto e alguns trabalhos correlatos. Tal comparação foi feita com o intuito de verificar como a solução apresentada comporta-se frente a outros algoritmos tradicionais e multirrelacionais em relação ao tempo de execução, ao uso de memória, bem como ao grau de escalabilidade.

[1] Disponível em http://www.netbeans.org

1.5 Organização da dissertação

A seguir, estão relacionados e brevemente descritos os capítulos que compõem o presente trabalho.

No capítulo 2 são relacionados os principais conceitos envolvendo a mineração de dados multirrelacional. Além disso, descreve-se o levantamento dos principais trabalhos correlatos.

No capítulo 3, é detalhado o desenvolvimento do trabalho propriamente dito, descrevendo os conceitos, as estruturas de dados e as demais informações referentes ao algoritmo proposto.

O capítulo 4, por sua vez, foi utilizado para a apresentação dos testes comparativos, bem como para uma discussão acerca dos resultados obtidos.

Por fim, o capítulo 5 apresenta as conclusões obtidas e algumas propostas para continuidade do trabalho.

Capítulo 2
Mineração multirrelacional de regras de associação

2.1 Considerações iniciais

O armazenamento e a recuperação de grandes volumes de dados tornou-se viável com a disponibilidade de ferramentas, técnicas e meios de armazenamento, principalmente com a consolidação do modelo relacional na década de 80. O surgimento de sistemas gerenciadores de banco de dados (SGBDs) possibilitou a manipulação eficiente dos dados. Apesar disso, à medida que as bases de dados aumentavam em tamanho, diminuía a possibilidade de se obter informações. A mineração de dados surgiu, então, como uma evolução natural desse processo (HAN; KAMBER, 2006), possibilitando que os dados, uma vez devidamente armazenados, pudessem ser transformados em informações úteis. A mineração multirrelacional é uma abordagem recente que possibilita extrair padrões envolvendo múltiplas tabelas, configurando-se como a maneira mais direta e eficiente de mineração em bases de dados relacionais.

Neste capítulo serão apresentados os conceitos que fundamentam o presente trabalho, com foco na mineração de regras de associação. Os principais algoritmos de mineração da abordagem tradicional – que trabalham sobre uma única tabela – serão inicialmente destacados, pois alguns deles servem de base para os algoritmos multirrelacionais tratados no trabalho. O cerne deste capítulo é a discussão acerca de

mineração de dados multirrelacional, levantando um panorama das abordagens e dos algoritmos existentes.

2.2 Mineração de dados

A mineração de dados ou *data mining* pode ser definida como a aplicação de técnicas computacionais para extração de conhecimento a partir de um conjunto de dados que, geralmente, é grande o suficiente para inviabilizar a análise humana. De um modo geral, pode ser definida como sendo a aplicação de algoritmos específicos para a extração de padrões dos dados (FAYYAD; PIATESTKY-SHAPIRO; PADHRAIC, 1996). Complementando as definições acima, a mineração de dados ainda pode ser definida como técnicas de extração de informações não-triviais, implícitas, previamente desconhecidas e potencialmente úteis dos dados armazenados (HAN; CHEN; YU, 1996).

A mineração de dados é considerada a etapa mais importante do Processo de Descoberta de Conhecimento em Bases de Dados ou *Knowledge Discovery in Databases* (FAYYAD; PIATETSKY-SHAPIRO; PADHRAIC, 1996). Além da mineração de dados propriamente dita, o KDD apresenta as etapas de pré-processamento – nas quais os dados são preparados para mineração – e de pós-processamento, em que os padrões obtidos são transformados a fim de facilitar a visualização e a compreensão. No apêndice A, é feito maior detalhamento acerca das definições de KDD e mineração de dados.

2.3 Mineração de regras de associação

A mineração de regras de associação pode ser vista como uma das mais importantes tarefas de mineração de dados. O desafio de encontrar regras de associação no contexto de bases de dados foi inicialmente exposto em Agrawal, Imielinski e Swami (1993) e consiste na busca por padrões associativos que indiquem o relacionamento entre conjuntos de itens. O exemplo clássico que ilustra a mineração de regras de associação é a chamada “análise de cesta de compras” (*market basket analysis*), que consiste na identificação das associações entre itens tal

que a presença de alguns itens na cesta implique frequentemente a presença de outros. A solução desse enunciado é possível com a aplicação de algoritmos de mineração de regras de associação.

Uma regra de associação pode ser definida na forma de uma implicação X→Y, sendo X e Y conjuntos de itens. Tais conjuntos são chamados de *itemsets*, sendo comum também referenciá-los por k-*itemsets*, em que k é o número de itens que o referido conjunto possui. Esse padrão indica uma associação entre o conjunto antecedente (X) e o conjunto consequente (Y), de modo que a ocorrência de X implica a ocorrência de Y. Para obter e mensurar as regras de associação são utilizadas duas medidas de interesse denominadas *suporte* e *confiança*.

O *suporte* representa a frequência da regra de associação, ou seja, indica a porcentagem de ocorrência concomitante dos conjuntos X e Y na base de dados.

$$suporte(X \rightarrow Y) = \frac{\text{nº de registros contendo X e Y}}{\text{total de registros}} \tag{2.1}$$

A medida confiança, por sua vez, indica a frequência em que a ocorrência do conjunto de itens X implica na ocorrência do conjunto Y. Tal medida expressa a validade da regra de associação X→Y.

$$confiança(X \rightarrow Y) = P(Y \mid X) = \frac{\text{nº de registros contendo X e Y}}{\text{nº de registros contendo X}} \tag{2.2}$$

Para melhor ilustrar o uso dessas medidas de interesse, considere um exemplo típico de “análise de cesta de compras”, cujos dados estão representados na tabela 2.1. Cada linha da tabela pode ser vista como uma cesta de compras, a qual possui um identificador (ID) e os respectivos itens comprados.

Tabela 2.1 Exemplo de cesta de compras

ID	ITENS
1	Pão, Manteiga, Leite
2	Pão, Café
3	Pão, Leite
4	Café, Frutas, Iogurte
5	Pão, Iogurte, Leite

Considere que os *itemsets* X e Y sejam, respectivamente, {Pão} e {Leite}. Para o cálculo do suporte da regra {Pão}→{Leite}, por exemplo, é computado o número de linhas nas quais ocorrem esses dois *itemsets*. Nesse caso, é possível encontrar ambos em 3 (três) cestas de compras. Portanto, o valor de suporte é igual a 3 / 5 = 0,6.

O cálculo da confiança é equivalente ao conceito de probabilidade condicional P(Y | X), ou seja, indica a frequência em que o *itemset* consequente Y ocorre dado o *itemset* antecedente X. Em termos práticos, para a regra de associação {Pão}→{Leite}, verifica-se que das 4 (quatro) cestas de compras em que ocorre o *itemset* {Pão} em 3 (três) há também a ocorrência do i*temset* {Leite}, isto é, a referida regra de associação possui confiança igual a 3 / 4 = 0,75.

A tabela 2.2 relaciona essas duas medidas de interesse, descrevendo a influência de cada uma na interpretação das regras de associação.

Tabela 2.2 Relação entre suporte e confiança (BERSON; SMITH; THEARLING, 1999)

	CONFIANÇA ALTA	**CONFIANÇA BAIXA**
SUPORTE ALTO	A regra é frequentemente correta e pode ser frequentemente utilizada	A regra é raramente correta, mas pode ser frequentemente utilizada
SUPORTE BAIXO	A regra é frequentemente correta, mas pode ser raramente utilizada	A regra é raramente correta e pode ser raramente utilizada

Os algoritmos de mineração das regras de associação necessitam conhecer os valores mínimos de suporte e de confiança, que são definidos a critério do analista de dados. Esses valores são utilizados como limitantes para estabelecer quais as regras de associação são interessantes ao contexto da análise. As regras que satisfazem, concomitantemente, o suporte mínimo (*min_sup*) e a confiança mínima (*min_conf*) são chamadas de regras de associação fortes e são aquelas que são resultantes do processo de mineração (HAN; KAMBER, 2006).

A extração de regras de associação pode ser, de um modo geral, dividida em duas etapas. A primeira consiste em localizar todos os *itemsets* frequentes, ou seja, aqueles cujo suporte seja igual ou superior ao valor *min_sup* previamente

estabelecido. A segunda etapa consiste na geração das regras de associação fortes a partir dos *itemsets* frequentes.

A etapa que realmente determina o desempenho do processo de extração de regras de associação é a busca pelos *itemsets* frequentes. Os diversos algoritmos propostos objetivam justamente otimizar essa etapa, tentando realizá-la da maneira mais eficiente possível.

A geração de regras de associação fortes, por sua vez, é uma etapa mais simples que pode ser resolvida de maneira direta, dada a prévia obtenção de todos os *itemsets* frequentes (KANTARDZIC, 2003).

2.4 Algoritmos de mineração de regras de associação

Dentre os algoritmos de mineração de regras de associação existentes, destacam-se o APRIORI (AGRAWAL; SRIKANT, 1994), baseado no paradigma de "geração-e-teste de candidatos"[2] (CGT) e o FP-GROWTH (HAN; PEI; YIN, 2000), baseado no paradigma *pattern-growth*[3] (PG).

Nas subseções seguintes, serão discutidas brevemente as principais características desses dois algoritmos de grande relevância para a área de mineração de regras de associação. Em seguida, será apresentado um panorama dos algoritmos tradicionais existentes, destacando quais foram as contribuições trazidas pelas propostas.

2.4.1 Algoritmo APRIORI

O algoritmo APRIORI foi idealizado e formalizado por Agrawal e Srikant (1994) a partir dos modelos matemáticos para a extração de regras de associação booleanas. O seu funcionamento é baseado no conceito de "geração-e-teste de candidatos", que divide cada iteração do algoritmo em duas fases, como ilustrado na Figura 2.1. A primeira fase visa à obtenção dos k-*itemsets* candidatos. Na primeira iteração, os 1-itemsets candidatos são obtidos por meio da varredura no conjunto de

[2] Em inglês, *Candidate Generate-and-Test*
[3] Pattern-growth – Preferiu-se manter o termo em inglês por não haver tradução que representa tão bem o termo original

dados. A partir da segunda iteração, o conjunto de k-*itemsets* candidatos é gerado por meio de combinações dos (k-1)-*itemsets* frequentes.

A segunda fase consiste no teste dos *itemsets* candidatos a fim de filtrar aqueles que são de interesse. Ou seja, faz a verificação daqueles que atendem à frequência mínima pré-estabelecida (*minsup*).

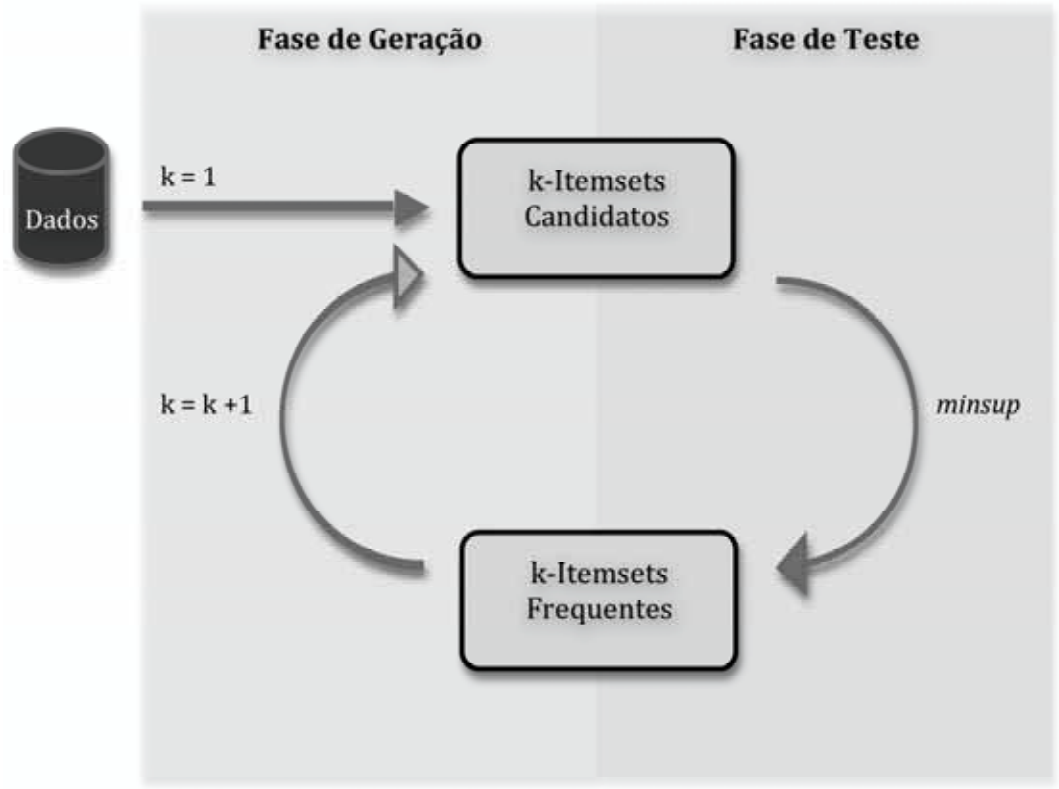

Figura 2.1 Esquema de funcionamento do algoritmo Apriori

Uma característica marcante do APRIORI é o processo de busca por largura, de modo que na i-ésima iteração são encontrados os i-*itemsets* frequentes. Além disso, como já foi indicado anteriormente, a geração dos k-*itemsets* candidatos depende do conjunto dos (k-1)-*itemsets* frequentes, ou seja, para obter padrões em uma dada iteração, é necessário consultar conhecimentos prévios - *prior knowledge* -, fato este que deu origem ao nome do algoritmo (HAN; KAMBER, 2006).

A eficiência desse algoritmo está ligada à possibilidade de redução do espaço de busca a cada iteração, por meio de uma operação denominada poda (*prune*). Para isso, o algoritmo leva em consideração a propriedade APRIORI, a qual diz que "todos os subconjuntos não-vazios de um *itemset* frequente devem ser também frequentes" (AGRAWAL; SRIKANT, 1994). Em outras palavras, se I é um i*temset* infrequente,

qualquer outro *itemset* J que contenha I também deve ser infrequente, pois a frequência de J não será maior que a frequência de I. Essa propriedade permite que o algoritmo APRIORI apresente um considerável ganho no desempenho, uma vez que, por meio da etapa de poda, são descartados da análise os *itemset*s candidatos que possuam algum subconjunto infrequente.

Assim, o algoritmo APRIORI apresenta boa eficiência na mineração de pequenos volumes de dados e para um alto valor de suporte mínimo (GYORODI *et al*., 2004). À medida que a quantidade de dados a ser minerada aumenta e/ou o valor do suporte mínimo diminui, o algoritmo APRIORI acaba gerando um número muito elevado de *itemset*s candidatos a cada iteração, o que torna o processamento computacionalmente custoso e, muitas vezes, inviável. Além disso, levando-se em conta que a cada iteração o algoritmo varre completamente o conjunto de dados para contabilizar as ocorrências de cada *itemset*, pode-se verificar que, quanto mais *itemset*s candidatos houver, mais iterações serão necessárias para a conclusão da mineração e, consequentemente, maior será o número de operações de acesso a disco (HAN; KAMBER, 2006).

2.4.2 Algoritmo FP-GROWTH

O algoritmo FP-GROWTH (HAN; PEI; YIN, 2000) introduziu a abordagem *pattern-growth* (PG) para a extração de regras de associação, apresentando uma alternativa às limitações existentes nos algoritmos de "geração-e-teste de candidatos" (CGT). A proposta da abordagem PG é utilizar-se de estruturas de dados - na maioria das vezes, tipos especializados de árvores - para a compressão e representação da base de dados. Tal utilização possibilita que o cerne do processamento seja realizado primariamente em memória, reduzindo as custosas operações que envolvem a leitura da base de dados. Além disso, os padrões frequentes são extraídos à medida que o algoritmo de mineração percorre tais estruturas, não sendo necessária a onerosa etapa de geração de candidatos, tal como ocorre com os algoritmos de CGT. Outra consequência do uso dessas estruturas é que, uma vez localizado um *itemset* frequente, o mesmo não necessita ser armazenado para consultas em iterações posteriores, ou seja, não há a ideia de conhecimentos prévios.

Particularmente, o algoritmo FP-GROWTH utiliza uma estrutura denominada *frequent-pattern tree* ou simplesmente *FP-tree* para representar comprimidamente o

conteúdo da base de dados. A Figura 2.2 apresenta um exemplo de uma *FP-tree*, no qual o nó *null{}* é o seu nó raiz e os demais nós contêm o identificador do item e o valor do suporte, separados por ":" (dois pontos), respectivamente.

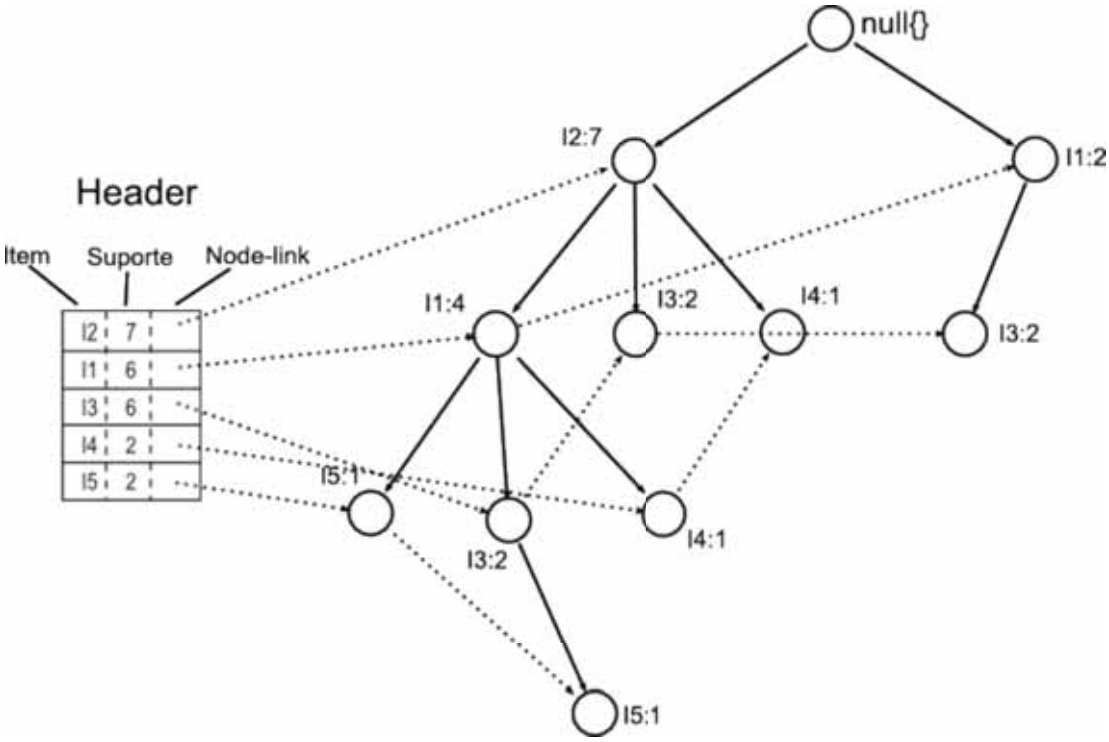

Figura 2.2 Exemplo de uma FP-tree (HAN; KAMBER, 2006)

Na *FP-tree*, cada sub-ramo da árvore representa a ocorrência dos itens desse sub-ramo na base de dados. Por exemplo, tome o ramo mais à direita da *FP-tree* da Figura 2.2, o qual é formado pelos itens I1 e I3. O *itemset* associado a esse ramo é, portanto, o {I1, I3} e possui o suporte em valor absoluto igual a 2 (dois), que é o valor do suporte do último nó visitado nesse ramo, no caso, I3. De modo análogo, pode-se obter todas as combinações de *itemsets* frequentes desse conjunto de dados, bastando o percurso por todos os ramos e sub-ramos da FP-tree.

Ainda com base na Figura 2.2 é possível observar a existência de uma estrutura auxiliar denominada tabela *Header*. Cada linha dessa tabela armazena o valor total do suporte correspondente a cada item, bem como mantém uma lista encadeada conectando todos os nós do mesmo item. Essa indexação apresenta-se como boa solução para a melhoria do desempenho quando a *FP-tree* for percorrida para extração dos *itemsets* frequentes.

Cabe acrescentar também que o algoritmo FP-GROWTH necessita de apenas duas passagens pela base de dados para a construção da árvore. Uma vez construída, todas as demais operações referentes à extração de *itemsets* são feitas em memória por meio da *FP-tree*. Assim, o algoritmo apresenta bom desempenho na maioria dos casos, devido à capacidade de compressão da base de dados pela *FP-tree*. A exceção ocorre quando a base de dados a ser minerada é esparsa, ou seja, cada registro da base de dados apresenta diferenças significativas em relação aos demais. Com isso, a *FP-tree* não consegue comprimir a base eficientemente, resultando em uma árvore com muitos nós, o que causa um impacto negativo no desempenho. Nesse caso, o algoritmo Apriori beneficia-se da esparsidade dos dados, pois o número de *itemsets* candidatos é reduzido rapidamente a cada iteração (PEI *et al*., 2001).

2.4.3 Outras propostas para a mineração de regras de associação

As principais propostas de melhorias no algoritmo APRIORI focam a redução do número de passagens pela base de dados e a geração de um número menor de *itemsets* candidatos a cada iteração (GANTI; GEHRKE; RAMAKRISHNAN, 1999).

Uma dessas propostas consiste no uso de estruturas *hash* para melhorar o desempenho da tarefa de extração de regras de associação, abordada no algoritmo HASH-MINE (WOJCIECHOWSKI; ZAKRZEWICZ, 2000) e no algoritmo DHP (PARK; CHEN YU, 1995). Especificamente, o algoritmo DHP propõe uma estrutura *hash* para favorecer a geração dos *itemsets* candidatos no algoritmo APRIORI, principalmente, nas duas primeiras iterações, as quais apresentam o maior custo computacional. Além disso, essa técnica emprega métodos de *prune* mais eficientes, diminuindo consideravelmente o número de transações utilizadas a cada iteração.

Outra proposta de otimização do algoritmo APRIORI envolveu a utilização de uma *trie* como estrutura central do algoritmo. Tal estrutura apresentou um ganho de desempenho superior quando comparado com as propostas de otimização do algoritmo Apriori baseadas em estruturas *hash* (BODON, 2003).

O algoritmo PARTITION (SAVASERE; OMIECINSKI; NAVATHE, 1995), por sua vez, apresenta o conceito de particionamento como uma estratégia para mineração de grandes bases de dados. Para esses casos, o algoritmo propõe a divisão da base de dados em unidades lógicas menores, denominadas partições, tal que cada partição possua um tamanho conveniente, de forma que isoladamente possa ser

processada em memória. Além do particionamento, o algoritmo PARTITION apresenta a vantagem de realizar a mineração em 2 (duas) passagens pela base de dados, diferentemente do algoritmo APRIORI, no qual não é possível prever a quantidade de passagens, pois sua aplicação depende do número de iteração que o algoritmo necessita efetuar.

O algoritmo SAMPLING (TOIVONEN, 1996) é uma proposta que realiza a mineração de padrões por meio de amostragem da base de dados. Isso permite que padrões possam ser extraídos de grandes bases de dados, bastando a geração de uma amostra que possa ser representada e processada em memória. O algoritmo SAMPLING efetua a mineração em, no máximo, duas passagens pela base de dados.

Há ainda estudos de integração de algoritmos de mineração de regras de associação com bases de dados relacionais, com a elaboração de sentenças SQL para geração de *itemsets* candidatos, com a contagem dos suportes dos *itemsets* e com as demais rotinas do algoritmo APRIORI (IMIELINSKI; VIRMANI, 1999; SARAWAGI; THOMAS; AGRAWAL, 2000).

A abordagem *pattern-growth* (PG) e seu principal representante, o algoritmo FP-GROWTH, também tem sido alvo de inúmeros estudos voltados à melhoria do algoritmo original ou a propostas de novos métodos eficientes de extração de *itemsets* sem a geração de candidatos. Os estudos são, na maioria das vezes, voltados à elaboração de novas estruturas de dados mais eficientes do que a *FP-tree* original ou, ainda, visam a apresentar soluções que sejam eficazes tanto em bases de dados densas quanto em bases de dados esparsas.

O algoritmo H-MINE (PEI *et al*., 2001) apresenta uma nova estrutura denominada *H-Struct* que, diferentemente da *FP-tree*, favorece a mineração de *itemsets* frequentes em dados esparsos. Tal estrutura mantém para cada registro da base de dados um vetor ou *array* de itens, além de um conjunto de ponteiros que referenciam outros registros. O algoritmo H-MINE também realiza duas passagens pela base de dados para a construção da *H-Struct*. Uma vez construída, essa estrutura segue constante durante o processamento e não há a criação de novas estruturas, apenas são realizadas manipulações envolvendo os ponteiros. O mesmo não ocorre com o algoritmo FP-GROWTH, que cria uma subárvore a cada chamada recursiva. Embora o H-MINE apresente melhorias no desempenho, principalmente, para bases de dados esparsas, o FP-GROWTH ainda é mais eficiente em base de dados densas.

Por esse motivo, o H-MINE utiliza uma heurística para identificar quando é mais vantajoso representar os dados por meio de uma *H-Struct* ou uma *FP-tree*. Para mineração de grandes bases de dados, o algoritmo adota uma estratégia similar ao algoritmo PARTITION, subdividindo a base de dados em partições.

Já o OPPORTUNEPROJECT é um algoritmo híbrido que estende as funcionalidades do FP-GROWTH e do H-MINE, processando os padrões utilizando uma *FP-tree* ou *H-Struct,* dependendo da densidade do conjunto de dados (LIU et al., 2002).

O algoritmo ITL-MINE (GOPALAN; SUCAHYO, 2002), por sua vez, otimiza o H-MINE necessitando de apenas uma passagem pela base de dados para a construção da estrutura central. Ademais, o algoritmo CTIL-MINE (SUCAHYO; GOPALAN, 2003) apresenta ainda a vantagem de utilizar uma estrutura compacta que reduz o espaço necessário para armazenamento e também diminui o tempo gasto de percurso na estrutura (CEGLAR; RODDICK, 2006). Outra proposta nessa linha é o algoritmo CT-PRO (GOPALAN; SUGAHYO, 2004) que apresenta desempenho superior ao CTIL-MINE, além de apresentar uma estratégia de particionamento para possibilitar a mineração de grandes bases de dados.

Em outro estudo, o algoritmo FP-GROWTH foi investigado quanto à sua estratégia de exploração. No algoritmo original, a estratégia utilizada é a *bottom-up*, de modo que o processamento dos padrões inicia-se dos nós-folhas em direção ao nó raiz. O algoritmo proposto TD-FP-GROWTH adota a estratégia *top-down* e demonstra-se mais eficiente em relação ao tempo e ao espaço de memória utilizado. Assim como o H-MINE, o algoritmo TD-FP-GROWTH utiliza a estrutura principal durante todo o processamento, não sendo necessária a criação de sub-estruturas durante as chamadas recursivas (WANG *et al*., 2002).

O algoritmo PATRICIAMINE (PIETRACAPRINA; ZANDOLIN, 2003) apresenta uma nova estrutura denominada *Patricia-trie*, que possui uma alta capacidade de compressão dos nós, reduzindo o espaço necessário em memória para o armazenamento da árvore. Com essa proposta, a estrutura proporciona ao algoritmo um bom desempenho em bases de dados esparsas e densas, sendo mais eficiente que os algoritmos H-MINE e OPPORTUNEPROJECT.

Além desses, destacam-se o AFOPT (LIU et al., 2003), o FP-GROWTH* (GRAHNE; ZHU, 2003), o NONORDFP (RACZ, 2004) e o FP-GROWTH-TINY

(OZKURAL; AYKANAT, 2004), que apresentam melhorias no desempenho em relação ao FP-GROWTH original, além de reduzirem o espaço de memória requerido para o armazenamento das estruturas de dados dos algoritmos (SAID; DOMINIC; ABDULLAH, 2009).

Os trabalhos mais recentes têm focado, particularmente, nas propostas de melhorias e incrementos dos algoritmos do estado da arte. Para citar, há um estudo envolvendo o algoritmo FP-GROWTH, com o intuito de otimizá-lo para que possa tomar proveito do ganho de desempenho oferecido pelos modernos processadores com arquitetura de vários núcleos, também conhecidos como *multi-core* (LIU *et al*., 2007).

Outro trabalho recente apresenta o algoritmo PTCLOSE, que é uma variação do PATRICIAMINE, modificado para possibilitar a mineração de um tipo diferenciado de *itemset* denominado *closed itemset*. A mineração de *closed itemsets* retorna um número menor de padrões, visto que remove *itemsets* que sejam subconjuntos de outros. O PTCLOSE faz uso da estrutura *Patricia-trie* e apresenta eficiência superior aos dos algoritmos correlatos (NEZHAD; SADREDDINI, 2007).

Uma estrutura de indexação denominada *IMine* foi proposta para uso juntamente com algoritmos de mineração de regras de associação, em especial, o FP-GROWTH. A *IMine* visa dar suporte à mineração em bases de dados relacionais, proporcionando um ganho de desempenho ao reduzir o custo das operações de entrada e saída (BARALIS; CERQUITELLI; CHIUSANO, 2009).

Cabe ainda relacionar os estudos comparativos de algoritmos de mineração de regras de associação encontrados na literatura (HIPP; GÜNTZER; NAKHAEIZADEH, 2000; GOETHALS, 2003; DUNHAM *et al*., 2002; IVÁNCSY; KOVÁCS; VAJK, 2004; ZHAO; BHOWMICK, 2006). Há também um estudo específico que compara os algoritmos APRIORI, PARTITION e SAMPLING (GŸORODI, 2003). O trabalho de Dexters, Purdom e Gucht (2006) apresenta uma análise probabilística dos principais algoritmos de mineração de regras de associação. Acrescenta-se à lista ainda o trabalho de Wang (2004), o qual apresenta uma proposta de categorização dos diversos algoritmos de extração de regras de associação.

2.5 Mineração de dados multirrelacional

Os algoritmos tradicionais de mineração de dados, tais como o APRIORI e o FP-GROWTH, buscam por padrões em dados que estejam dispostos em uma estrutura única, por exemplo, uma tabela ou um arquivo. Para a aplicação desses algoritmos em bases de dados relacionais, os dados provenientes de um conjunto de relações devem passar por uma etapa de pré-processamento, na qual os mesmos são devidamente reunidos em uma única tabela por meio de operações de junção ou agregação[4].

Embora essa proposta seja possível e suficiente para algumas aplicações, a utilização dos algoritmos tradicionais de mineração em múltiplas tabelas pode produzir resultados insatisfatórios. Quando da extração em grandes bases de dados, a operação de junção pode gerar uma tabela com muitos registros, comprometendo o desempenho dos algoritmos. Além disso, durante o pré-processamento pode ocorrer aparecimento de inconsistências nos dados ou perda de informações (TSECHANSKY *et al*., 1999; DZEROSKI; RAEDT; WROBEL, 2003), como será ilustrado no exemplo a seguir.

[4] Agregação: operação que toma uma coleção de valores como entrada, retornando um valor simples que sintetiza o conjunto. Por exemplo, média e contagem são operações de agregação. (SILBERSCHATZ *et al*., 1999).

Paciente

ID Pac	Nome	Sexo	Data Nascimento
011	João	M	01/08/1976
012	Maria	F	10/05/1984
013	César	M	14/11/1960
...	...	...	...

Internação

ID Intern	Paciente	Motivo	Dias
0201	011	Infecção	20
0202	012	Bronquite	2
0203	012	Pneumonia	14
0204	013	Bronquite	4
...	...	...	...

(a)

ID Paciente	Nome	Sexo	Data Nascimento	ID Internacao	Motivo	Dias
011	João	M	01/08/1976	0201	Infecção	20
012	Maria	F	10/05/1984	0202	Bronquite	2
012	Maria	F	10/05/1984	0203	Pneumonia	14
013	César	M	14/11/1960	0204	Bronquite	4
...	...	...	...	...	...	...

(b)

ID Paciente	Nome	Sexo	Data Nascimento	ID Internacao	Num Internacoes	Média Dias
011	João	M	01/08/1976	0201	1	20
012	Maria	F	10/05/1984	0203	2	8
013	César	M	14/11/1960	0204	1	4
...	...	...	...	...	...	...

(c)

Figura 2.3 a) Relações *Paciente* e *Internação* em uma base de dados relacional; b) Resultado de uma junção natural de *Paciente* e *Internação*; c) Utilização de funções agregadas gerando os atributos *Num_Internacoes* e *Média_Dias.*

O exemplo ilustra uma base de dados simplificada de um hospital, na qual são armazenados os dados referentes aos pacientes e suas respectivas internações. A Figura 2.3(a) apresenta a tabela *Paciente* que se associa à tabela *Internação*, por meio de um relacionamento do tipo ’um para muitos’.

Para tornar possível a mineração dessa base de dados pelos algoritmos tradicionais, é necessário realizar um pré-processamento para reunir os dados em uma única tabela. A Figura 2.3(b) exibe o resultado da junção natural entre *Paciente* e *Internação*, que origina uma tabela contendo os dados provenientes de ambas, por meio do relacionamento de chave estrangeira existente (atributo *Paciente* da tabela *Internação*). Como mencionado anteriormente, esse pré-processamento pode introduzir inconsistências nos dados de origem. No exemplo, a tabela resultante da junção apresenta redundância nos dados da paciente *Maria*, o que levaria a imprecisões nos resultados obtidos.

A Figura 2.3(c) exemplifica o uso de funções de agregação para a generalização dos dados, que objetiva eliminar as redundâncias existentes na tabela (b). No exemplo, foram utilizadas as funções *soma* e *média*. A primeira função foi

utilizada para contabilizar o número de internações de cada paciente – atributo *Num_Internacoes*; a segunda função, por sua vez, foi aplicada para calcular a média de dias de internação – atributo *Média_Dias.* Não obstante, o uso de funções de agregação pode levar a perda de informações, tal como ocorre quando se utiliza o valor médio (*Média_Dias*) para representar o conjunto de dias de internação. A média de dias para a paciente Maria é de 8 (oito) dias; entretanto, ao verificar a tabela *Internação*, pode-se notar que as duas internações desta paciente variam bastante entre si, que são 2 (dois) e 14 (catorze) dias, respectivamente. Essa diferença pode indicar um padrão associativo com o motivo da internação, de modo que a análise do atributo *Dias* seria, nesse caso, mais interessante que *Média_Dias*. O uso de funções agregadas ainda causa outro problema: a eliminação de atributos não-agregáveis, como é o caso de *Motivo*. O atributo não pode ser sumarizado e, por esse motivo, é removido da tabela resultante.

A mineração multirrelacional é uma abordagem mais recente que visa contornar as dificuldades encontradas na aplicação dos algoritmos tradicionais, possibilitando a extração de padrões diretamente em múltiplas relações, sem a necessidade da transferência dos dados para uma única relação (DZEROSKY; RAEDT; WROBEL, 2003; DOMINGOS, 2003). Com isso, evitam-se as custosas operações de junção e as perdas semânticas ocasionadas pela limitação da representação em uma única tabela.

Assim, de um modo geral, a mineração multirrelacional pode ser descrita como um campo multi-disciplinar que engloba programação lógica indutiva, aprendizagem de máquina, KDD, bases de dados relacionais, entre outros (BLOCKEEL; DZEROSKY, 2005). A mineração de bases de dados relacionais é uma das principais aplicações da abordagem multirrelacional (PIZZI; RIBEIRO; VIEIRA, 2005; TEREDESAI *et al*., 2005; GARCIA; VIEIRA, 2008), porém o leque de aplicações é ainda mais amplo. Tal abordagem possibilita a mineração de objetos complexos ou estruturados, tais como proteínas e subestruturas moleculares e outras aplicações cujas informações estruturais sejam relevantes à análise (PAGE; CRAVEN, 2003).

Apesar de introduzir vantagens frente à abordagem tradicional, a mineração de dados multirrelacional não apresentou um rápido crescimento inicial, o que foi ocasionado devido a três fatores principais: i) a escalabilidade limitada apresentada

pelos algoritmos multirrelacionais; ii) a inabilidade para tratar ruídos; e iii) incertezas e a carência de aplicações que pudessem fazer uso efetivo desses algoritmos (DOMINGOS, 2003).

As primeiras propostas viáveis de mineração multirrelacional remetem à Programação Lógica Indutiva (*Inductive Logic Programming* - ILP), com técnicas baseadas na representação dos padrões em forma de lógica de primeira ordem (KNOBBE *et al*., 1999), que tem como principal representante o algoritmo WARMR (DEHASPE; RAEDT, 1997). Atualmente, é possível encontrar uma quantidade razoável de técnicas multirrelacionais, sendo que as principais tarefas de mineração de dados – análise de regras de associação, classificação, análise de agrupamentos, entre outros – já podem ser realizadas seguindo a abordagem multirrelacional. (DZEROSKY; RAEDT; WROBEL, 2003).

2.6 Algoritmos multirrelacionais de mineração de regras de associação

Os algoritmos multirrelacionais de extração de regras de associação utilizam diferentes abordagens para representar e extrair os padrões. Uma primeira abordagem engloba os algoritmos baseados em lógica ou também conhecidos como algoritmos de programação lógica indutiva (ILP). A característica principal dessa abordagem é a representação dos dados e padrões em *datalogs*, que são escritos na forma de lógica de primeira ordem. O algoritmo de ILP mais difundido para mineração de regras de associação é o WARMR (DEHASPE; RAEDT, 1997), que apresenta um funcionamento baseado no algoritmo Apriori. O WARMR tem sido utilizado para mineração de padrões em dados estruturados, tal como encontrados em bases de dados químicas (KING; SRINIVASAN; DEHASPE, 2001).

Outro algoritmo de ILP encontrado na literatura é o FARMER (NIJSSEN; KOK, 2001), que possui similaridades com o WARMR, porém apresenta uma maior eficiência com o uso de árvores ordenadas para organização dos itens frequentes. O algoritmo RADAR (CLARE; WILLIAMS; LESTER, 2004), por sua vez, apresenta uma proposta para extração de regras de associação em grandes bases de dados relacionais por meio do uso de técnicas de indexação que reduzem a quantidade de memória principal necessária para a busca dos padrões frequentes.

Os algoritmos baseados em grafos são outro tipo de abordagem para a extração de regras de associação multirrelacionais. Nesta abordagem, a representação dos padrões multirrelacionais é feita por meio de grafos e o processo de mineração é embasado em teorias matemáticas para identificação do conjunto de subgrafos que satisfazem uma determinada frequência de ocorrência (KETKAR; HOLDER; COOK, 2005). O algoritmo AGM (INOKUCHI; WASHIO; MOTODA, 2000), baseado também no APRIORI, foi uma das primeiras soluções a utilizar a teoria matemática de grafos para extração de padrões frequentes. A partir daí, surgiram outras técnicas - como o FSG (KURAMOCHI; KARYPIS, 2004) - visando melhorias no desempenho do processo. Por outro lado, algoritmos como o GBI (MATSUDA *et al*., 2000) propõem técnicas heurísticas que apresentam soluções aproximadas - porém com melhor eficiência - para a tarefa de encontrar padrões frequentes.

Por fim, há os algoritmos multirrelacionais baseados no funcionamento e nas estruturas de dados de técnicas tradicionais, principalmente o APRIORI e o FP-GROWTH. O intuito dessa abordagem é estender os algoritmos de mineração tradicional, que funcionam adequadamente para extração de padrões em uma única tabela, adaptando-os ao contexto multirrelacional de modo a possibilitar o processamento de padrões estruturalmente mais complexos. O escopo deste trabalho recai justamente sobre os algoritmos dessa abordagem.

O algoritmo multirrelacional MRFP-GROWTH (TEREDESAI *et al*., 2005) é uma extensão do algoritmo tradicional FP-GROWTH e utiliza-se de uma estrutura de dados denominada *MRFP-tree* para representar os padrões relacionais. O algoritmo inicialmente localiza os padrões frequentes locais para cada uma das tabelas envolvidas. Na etapa seguinte, esses padrões locais são avaliados a fim de se obter os padrões multirrelacionais de fato.

O APRIORI-GROUP (RIBEIRO; VIEIRA; TRAINA, 2005) estende o funcionamento do algoritmo Apriori, permitindo a mineração de padrões envolvendo mais de uma tabela. Para isso, o algoritmo utiliza-se do conceito de agrupamentos, corrigindo possíveis imprecisões provenientes da redundância de dados introduzidas durante a etapa de pré-processamento, em que as múltiplas tabelas são agrupadas em uma única por meio de operações de junção.

O algoritmo CONNECTION (PIZZI; RIBEIRO; VIEIRA, 2005), por sua vez, baseia-se no algoritmo FP-GROWTH e foi inicialmente idealizado para uso em data warehouses. No entanto, é possível utilizá-lo na abordagem multirrelacional, desde que as múltiplas tabelas compartilhem ao menos um atributo. O algoritmo efetua inicialmente a mineração dos padrões para cada tabela. Então, os padrões locais são combinados para a extração dos padrões multirrelacionais, de modo análogo ao algoritmo MRFP-GROWTH.

O algoritmo APRIORIMR (OYAMA, 2006) caracteriza-se como uma extensão do algoritmo Apriori para mineração de regras de associação multirrelacional. O APRIORIMR modifica algumas etapas do algoritmo original, adequando-os para ser utilizado em bases de dados relacionais. Para o cálculo correto do suporte dos *itemsets* multirrelacionais, é utilizado o conceito de agrupamento (RIBEIRO; VIEIRA; TRAINA, 2005), de modo que as medidas de interesse – suporte e confiança – levem em conta a possível duplicidade de dados devido à junção de tabelas.

Outro representante dessa categoria é o algoritmo GFP-GROWTH (PIZZI, 2006), que realiza a mineração de padrões multirrelacionais baseando-se no funcionamento do FP-GROWTH e no conceito de agrupamentos, introduzido pelo algoritmo APRIORI-GROUP (RIBEIRO; VIEIRA; TRAINA, 2005). Segundo resultados obtidos em um teste comparativo, o algoritmo GFP-GROWTH apresenta melhor desempenho em relação ao algoritmo CONNECTION.

O CONNECTIONBLOCK (GARCIA, 2008) é uma modificação do algoritmo CONNECTION (PIZZI; RIBEIRO; VIEIRA, 2005) para contemplar o conceito de bloco, que consiste no conjunto de registros das tabelas envolvidas que compartilham um mesmo valor de um determinado identificador. Além disso, foi proposto outro algoritmo denominado CONNECTIONBLOCKQ (GARCIA, 2008), que possui o diferencial de possibilitar a mineração de regras de associação contendo campos quantitativos. O CONNECTIONBLOCKQ automatiza a tarefa de discretização dos dados quantitativos, que é, na maioria dos casos, realizada na etapa de pré-processamento, transformando os dados contínuos em classes ou faixas de valores.

Outros estudos recentes em mineração multirrelacional de regras de associação focam em algoritmos ILP, os quais não fazem parte do escopo deste trabalho, uma vez que não operam diretamente nas bases de dados relacionais. Pode-

se destacar o novo método denominado C^2D (SENKUL; TOROSLU, 2009) que visa tornar viável a descoberta de regras por usuários não-experientes. Para tal, a proposta remove tarefas complexas, tal como a especificação de predicados, que são comuns na ILP. Há ainda outro algoritmo ILP recente, KDB-KRIMP (KOOPMAN; SIEBES, 2009), que visa a extração de padrões em bases de dados por meio de uma nova forma de representação denominada *code tables*. Tal representação apresenta a vantagem de mesclar descrições globais e locais, permitindo representar a base de dados completa e, ao mesmo, preservar os detalhes de cada padrão.

A literatura de mineração multirrelacional de regras de associação em bases de dados, diferentemente da abordagem tradicional, carece de estudos comparativos ou elucidativos acerca do desempenho dos algoritmos existentes. Pelas conclusões provenientes da abordagem tradicional, sabe-se que os algoritmos baseados no paradigma PG, tais como o MRFP-GROWTH e o GFP-GROWTH, apresentam maior eficiência do que baseados no paradigma CGT, como é o caso do APRIORI-GROUP e APRIORIMR, para a maioria das situações e configurações de bases de dados.

Nas seções subsequentes, será realizado um estudo das principais propostas de mineração de regras de associação em bases de dados relacionais, descrevendo o funcionamento, as estruturas de dados, bem como as principais contribuições e dificuldades de cada algoritmo.

2.6.1 Algoritmo MRFP-GROWTH

O algoritmo MRFP-GROWTH (TEREDESAI *et al*., 2005), caracteriza-se como uma extensão do algoritmo tradicional FP-GROWTH (HAN; PEI; YIN, 2000), contemplando a extração de regras de associação envolvendo múltiplas relações. Assim como no FP-GROWTH, a principal característica do algoritmo MRFP-GROWTH é a ausência da custosa etapa de geração de candidatos existente no algoritmo APRIORI e seus derivados. Para que isso seja possível, é utilizada uma estrutura em memória que representa os itens *frequentes* e as informações das associações dos *itemsets*, denominada *MRFP-tree*.

O MRFP-GROWTH realiza o processo de extração de padrões frequentes em múltiplas tabelas. Para isso, classifica as tabelas presentes na base de dados como sendo primária ou secundária. A tabela primária é aquela que possui uma determinada chave primária ID, enquanto que as tabelas secundárias possuem uma

referência à chave P da tabela primária, geralmente por meio de uma chave estrangeira. Na Figura 2.4, apresenta-se a tabela 1 como primária com seu atributo TID como identificador e as tabelas 2 e 3 como secundárias, com chaves estrangeiras TID_2 e TID_3, respectivamente. As tabelas secundárias possuem um relacionamento do tipo "um-para-muitos" com a tabela primária.

Tabela 1: primária

TID	Itens
1	A,B,C,D
2	E,G,H
3	A,B
4	C,D,A
5	G,E,C
6	F,H,G
7	D,E,B

Tabela 2: secundária-A

TID_2	Itens
1	α,β,γ
1	β,δ
2	δ,θ,η
3	α,θ
4	β,η,γ
5	α,δ
5	α,η,γ
7	β,η

Tabela 3: secundária-B

TID_3	Itens
1	Aa Cc
2	Bb Dd Aa
2	Bb Ee
3	Cc Ff
4	Hh Dd Gg
6	Gg Ff
6	Aa Cc
7	Ee

Figura 2.4 Tabela primária e tabelas secundárias (KANODIA, 2005)

Um importante conceito do algoritmo MRFP-GROWTH é a propagação de *ID set*. Um *ID set* pode ser definido como o conjunto de IDs em que um determinado padrão ocorre (KANODIA, 2005). Tome como exemplo o padrão "A→B" da tabela 1, o qual pode ser encontrado nas tuplas com TID 1 e 3. Assim, define-se como o *ID set* deste padrão como {1,3}. A ideia da propagação de *ID set* é relacionar implicitamente padrões de tabelas diferentes, evitando o uso de junções, que é um processo bem mais custoso computacionalmente.

Algoritmo: MRFP-Growth
Entradas: Base de dados D; minsup
Saídas: Conjunto de padrões multirrelacionais frequentes

1. Para cada tabela secundária, faça:
   a) Construa a MRFP-tree para os itens da tabela. Mantenha uma lista de IDs associados a cada item
   b) Extraia os itens frequentes da MRFP-tree e os respectivos *ID sets*
2. Construa uma MRFP-tree para os IDs nos ID sets de todos os padrões frequentes gerados por todos as relações secundárias
3. Minere a MRFP-tree final e encontre os padrões frequentes.

Figura 2.5 Algoritmo MRFP-Growth (KANODIA, 2005)

O funcionamento básico do algoritmo está descrito na Figura 2.5. O mesmo inicia-se com a construção de uma *MRFP-tree* para cada tabela secundária da base de dados, com o intuito de obter seus respectivos padrões frequentes locais.

A construção da *MRFP-tree* é similar ao processo que ocorre no algoritmo FP-GROWTH, com exceção de que cada nó da árvore tem que manter uma lista de IDs indicando as tuplas em que o item ocorre. Considere o suporte mínimo *minsup*=2. A Figura 2.6 apresenta a *MRFP-tree* construída a partir da tabela primária.

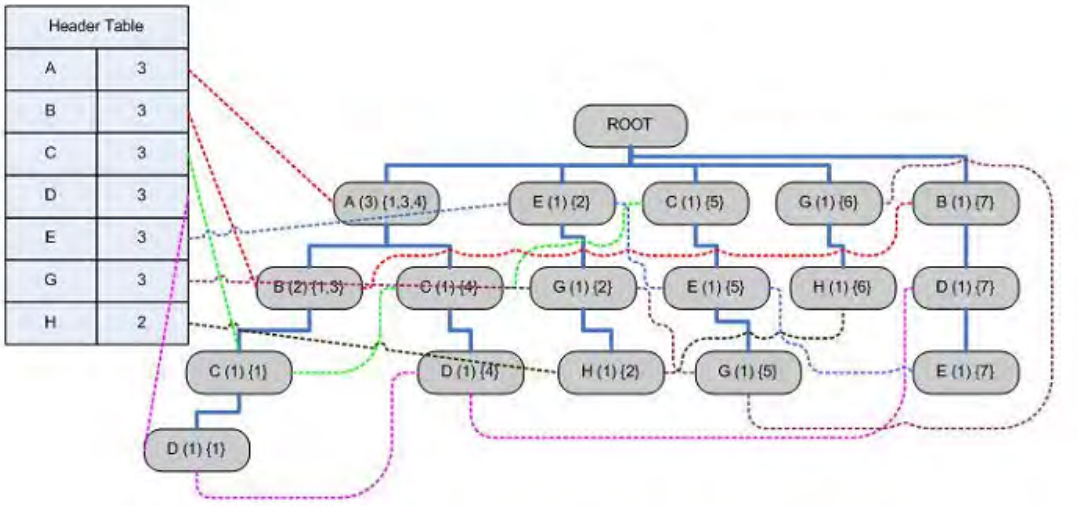

Figura 2.6 MRFP-tree para a tabela primária (KANODIA, 2005)

Como mostra a figura, cada nó da *MRFP-tree* contém, respectivamente, o item, o valor do contador de suporte - que é indicado entre parênteses - e uma lista de IDs, que é indicada entre chaves. De modo análogo, repete-se a construção das *MRFP-trees* para as tabelas secundária-A e secundária-B, as quais são exibidas, respectivamente, pela Figura 2.7 e pela Figura 2.8.

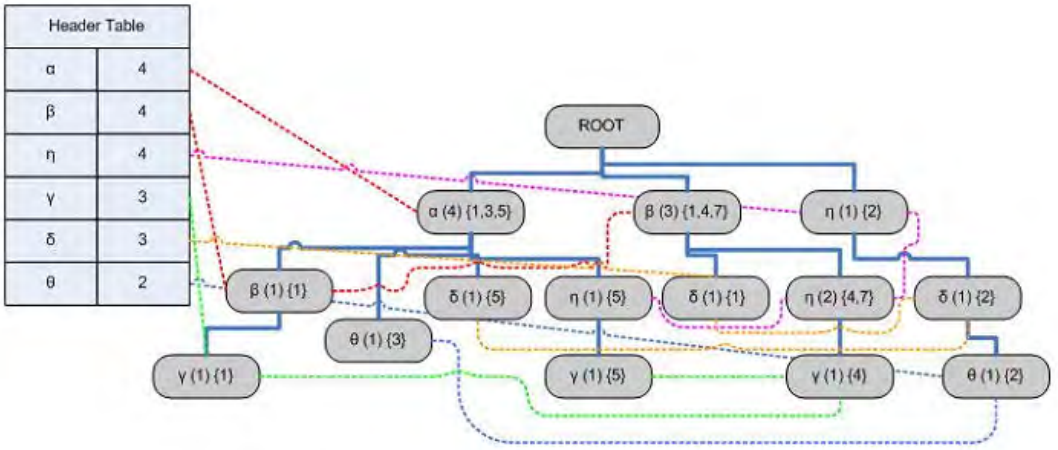

Figura 2.7 MRFP-tree para a tabela secundária-A (KANODIA, 2005)

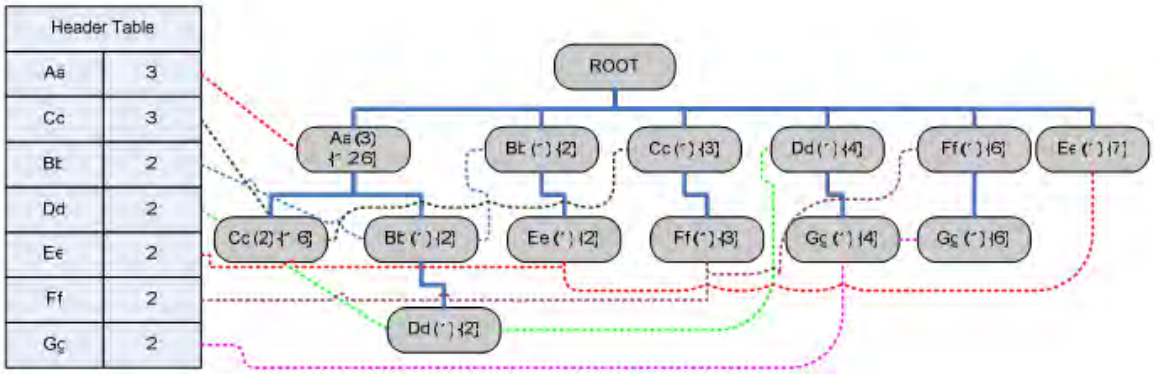

Figura 2.8 MRFP-tree para a tabela secundária-B (KANODIA, 2005)

Cada uma das árvores geradas são mineradas por um algoritmo similar ao FP-GROWTH, com a diferença de considerar a existência das *ID set*s. À medida que os padrões frequentes locais vão sendo obtidos, os mesmos devem ser inseridos em uma tabela, tal como mostrada na Tabela 2.3. As linhas 1 a 8 são os padrões frequentes locais resultantes da tabela primária, as linhas 9 a 12 referem-se à tabela secundária-A e a linha 13 refere-se à tabela secundária-B.

Tabela 2.3 Padrões frequentes extraídos isoladamente (KANODIA, 2005)

ID	ID set	Padrões frequentes	Suporte
1	2 6	H G	2
2	2 5	G E	2
3	1 4	D C	2
4	1 4	D C A	2
5	1 7	D B	2
6	1 4	D A	2
7	1 4	C A	2

8	1 3	B A	2
9	4 5	γ η	2
10	1 4	γ β	2
11	1 5	γ α	2
12	4 7	η β	2
13	1 6	Cc Aa	2

Cada linha da tabela recebe um ID único e relaciona cada padrão *frequente* obtido às linhas da tabela em que ocorre por meio do atributo *ID set*. Por exemplo, o padrão frequente "H G" ocorre nas tuplas com ID 2 e 6 da tabela primária.

Uma vez que todas as *MRFP-tree* forem processadas e a tabela de padrões frequentes esteja disponível, inicia-se a construção da *MRFP-tree* final, que permitirá extrair de fato os padrões multirrelacionais. O processo de construção considera o atributo *ID set* da tabela de padrões frequentes como o item dos nós da *MRFP-tree* final, tal como exibida na Figura 2.9.

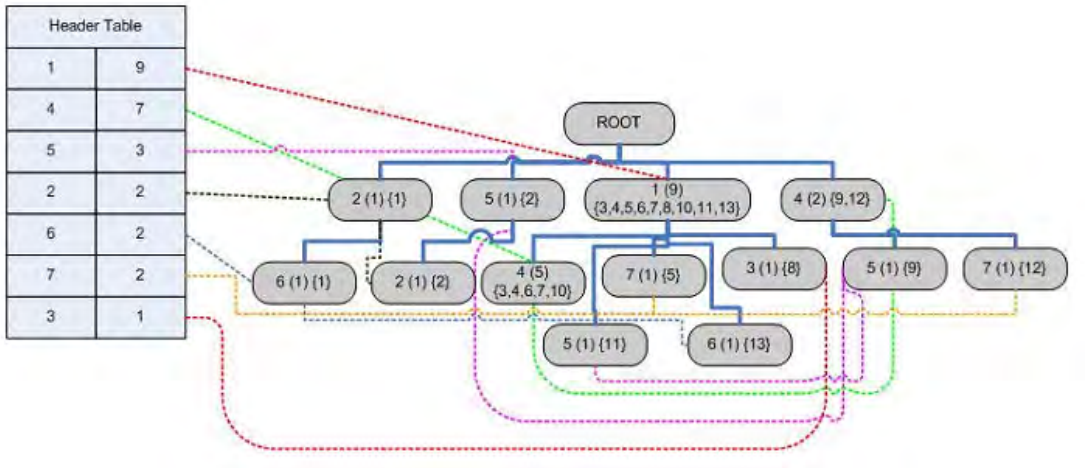

Figura 2.9 MRFP-tree final (KANODIA, 2005)

A mineração da *MRFP-tree* final gerará padrões frequentes que consistem em IDs e não em itens reais. A Tabela 2.4 apresenta esses padrões obtidos pela mineração da *MRFP-tree* final. O primeiro campo da tabela é o *Cross Support*, que é definido como a frequência de ocorrência de um padrão na base de dados, considerando as múltiplas tabelas. O campo *ID set* apresenta os padrões frequentes obtidos pela *MRFP-tree* final, enquanto que o campo ID apresenta as tuplas da tabela de padrões frequentes (Tabela 2.3) em que há a ocorrência do *ID set*. Por fim, a última coluna apresentada é o padrão frequente já mapeado para os itens reais. Esse

mapeamento é feito armazenando os padrões frequentes das linhas da Tabela 2.3 indicadas pelo(s) valor(es) do campo ID. O padrão frequente final será o conjunto formado pela união desses padrões frequentes.

Na Tabela 2.4, por exemplo, observe-se o ID set "1 4" que apresenta, no campo ID, o conjunto {3, 4, 6, 7, 10}. Verificando na Tabela 2.3 as linhas correspondentes aos valores desse conjunto, encontram-se os seguintes padrões frequentes "D C", "D C A", "D A", "C A", "γ β", respectivamente. Assim, o padrão frequente final é o conjunto gerado pela união desses padrões.

Tabela 2.4 Padrões frequentes finais (KANODIA, 2005)

Cross Support	ID set	ID	Padrões frequentes
2	1 3	8	A B
2	4 7	12	β η
2	1 7	5	D B
2	2 6	1	H G
2	1 6	13	Cc Aa
2	2 5	2	E G
2	4 5	9	η γ
2	1 5	11	α γ
2	1 4	3 4 6 7 10	D β C A γ

O algoritmo MRFP-GROWTH preserva as principais características do FP-GROWTH, no qual foi baseado. Entre elas, destaca-se a utilização de estruturas de dados em memória para efetuar o processo de mineração. Isso reduz substancialmente o acesso a disco, porém, como ocorre com o próprio FP-GROWTH, a memória principal pode não ser suficiente para o processamento de grandes volumes de dados. O MRFP-GROWTH, no entanto, tenta minimizar o problema de limitação de memória por meio da estratégia de análise das tabelas individualmente, de modo que somente os dados de uma única tabela fiquem na memória em dado momento. Essa estratégia também permite extrair padrões locais, ou seja, que ocorrem dentro de uma única tabela. Com isso, são encontrados tanto padrões frequentes multirrelacionais quanto aqueles envolvendo tabelas individuais.

Outra característica positiva do MRFP-GROWTH é a ideia de propagação de ID, que implicitamente relaciona as tuplas de diferentes tabelas. Isso evita o uso de junções, as quais, como dantes mencionado, configuram-se como operações computacionalmente custosas. Assim, tal ideia acaba por resultar em um ganho de desempenho.

Por outro lado, um ponto fraco do algoritmo MRFP-GROWTH é o grande número de estruturas de dados (*MRFP-tree*) criadas durante o processo de mineração de dados. Se N for o número de tabelas da base de dados, serão necessárias N+1 *MRFP-trees*. Além do espaço de memória considerável, a geração de muitas *MRFP-trees* acarreta também a necessidade de repetidas execuções dos métodos de construção e mineração da árvore.

Porém, cabe acrescentar que, assim como o FP-GROWTH, o algoritmo MRFP-GROWTH também apresenta baixo desempenho quando a base de dados é esparsa. Isso ocorre porque, uma vez que a árvore apresenta muitos ramos, o processo recursivo de mineração acaba por tornar-se oneroso.

2.6.2 Algoritmo APRIORI-GROUP

O algoritmo APRIORI-GROUP (RIBEIRO; VIEIRA; TRAINA, 2005) modifica o algoritmo APRIORI (AGRAWAL; SRIKANT, 1994) para possibilitar a mineração confiável em bases de dados considerando a ocorrência de agrupamentos. Agrupamentos são definidos como conjuntos de transações ou tuplas relacionadas que trazem consigo informações sobre uma mesma entidade. A não observância desse fato acarreta padrões incoerentes, uma vez que as medidas de interesse possam estar incorretas.

Em caráter de exemplo, considere as tabelas Clientes (Tabela 2.5) e Compras (Tabela 2.6) de uma base de dados comercial.

Tabela 2.5 Tabela Clientes

IDCliente	Sexo	Renda
1	M	Média
2	F	Alta
3	M	Média
4	M	Baixa

Tabela 2.6 Tabela Compras

IDCompra	Produto	IDCliente
1	Pão	1
2	Carne	2
3	Leite	2
4	Pão	2
5	Leite	3
6	Pão	4

Para utilizar o algoritmo APRIORI-GROUP, faz-se necessária uma etapa de pré-processamento de dados, a fim de efetuar a junção das tabelas Clientes e Compras. O resultado da junção, a tabela Cli-Compra, é mostrada na Tabela 2.7 .

Tabela 2.7 Tabela Cli-Compra resultante da junção das tabelas Clientes e Compras

IDCliente	Sexo	Renda	IDCompra	Produto
1	M	Média	1	Pão
2	F	Alta	2	Carne
2	F	Alta	3	Leite
2	F	Alta	4	Pão
3	M	Média	5	Leite
4	M	Baixa	6	Pão

Como pode ser observado nas linhas destacadas da Tabela 2.7, há valores redundantes para os atributos relativos aos clientes. Neste caso, as informações sobre o sexo e a renda do cliente cujo IDCliente é 2 aparecem três vezes na tabela Cli-Compra. A aplicação de um algoritmo tradicional nesta base de dados levaria à extração de padrões incorretos ou imprecisos. Para ilustrar melhor esse problema, tome-se o padrão {Sexo=F, Renda=Alta}. Nos algoritmos tradicionais, tal padrão apresentaria suporte=50%, desde que 3 (três) das 6 (seis) tuplas da tabela satisfaçam o padrão. Porém, o valor semanticamente correto para o suporte deste padrão é 25%, uma vez que somente um dos quatro clientes apresenta esse padrão.

Para evitar esse tipo de erro, o algoritmo APRIORI-GROUP implementa as medidas de interesse tomando por base a ocorrência de agrupamentos na tabela alvo. No caso do exemplo da Tabela 2.7, pode-se assumir que o IDCliente é o identificador dos agrupamentos, de modo que múltiplas ocorrências de um padrão em um mesmo agrupamento passam a ser contabilizadas como uma única ocorrência. Assim, agrupando as tuplas com base no atributo IDCliente, as múltiplas ocorrências do padrão “Sexo=F, Renda=Alta” para o cliente 2 são reduzidas a uma única, de modo que agora o valor de suporte é 25%, que é a frequência de ocorrência semanticamente correta.

Na Figura 2.10 tem-se o pseudocódigo do APRIORI-GROUP, que apresenta muita semelhança com o algoritmo Apriori. As diferenças existentes decorrem da adaptação do algoritmo tradicional para manipular corretamente os padrões levando em consideração a ocorrência de agrupamentos na base de dados.

Algoritmo: Apriori-Group
Entradas: Base de dados D; minsup
Saída: Itemsets frequentes L

```
1)  G = Ø                                          // conjunto de agrupamentos
2)  para cada transação t
3)      se existir g ∈ G tal que ID(g) = ID(t)
4)      adicione a transação t ao agrupamento g
5)      senão
6)      crie um novo agrupamento g tal que ID(g)=ID(t)
7)      adicione g a G
8)  para cada agrupamento g ∈ G
9)      para cada item c que ocorre em g
10)     c.contador++;
11) L1={c | c.contador/|G| >= minsup}      // conjunto de 1-itemsets frequentes
12) para (k = 2; Lk-1 ≠ Ø; k++)
13)     Ck =Apriori-gen(Lk-1)              // conjunto de k-itemsets candidatos
14)     para cada agrupamento g ∈ G
15)         para cada candidato c ∈ Ck contido em g faça
16)             c.contador++
17)         Lk  = { c ∈ Ck | c.contador/|G|  >= minsup }      // conjunto de
                                                              k-itemsets candidatos
18) retorne L = Uk Lk;
```

Figura 2.10 Algoritmo Apriori-Group (RIBEIRO; VIEIRA; TRAINA, 2005)

O procedimento inicia-se com a criação do conjunto de agrupamentos G, no qual cada transação da base de dados é avaliada a fim de que registros com mesmo identificador ID formem um mesmo grupo (linhas 1 a 7). Em seguida, verifica-se o número de ocorrências dos itens de cada agrupamento na base de dados. Os itens que apresentam uma frequência superior ao suporte mínimo *minsup* são considerados 1-*itemsets* frequentes, dando origem ao conjunto L_1 (linhas 8 a 11). A partir desse ponto, o algoritmo APRIORI-GROUP comporta-se similarmente ao algoritmo APRIORI, efetuando alternadamente as etapas de geração e teste de candidatos até que todos os k-*itemsets* sejam extraídos (linhas 12 a 18).

O algoritmo APRIORI-GROUP apresenta-se como uma alternativa para a mineração de dados envolvendo múltiplas tabelas, uma vez que possibilita a extração de padrões coerentes por meio da análise da ocorrência de agrupamentos no conjunto de dados. Porém, a sua aplicação se torna relativamente comprometida, tendo em vista a necessidade da junção de tabelas. Como já visto anteriormente, essa operação é computacionalmente custosa e pode acarretar perdas semânticas. Além disso, a proposta tem por base o algoritmo APRIORI e, portanto, apresenta os mesmos

problemas da abordagem CGT na mineração de grandes volumes de dados, ou seja, a geração de elevado número de *itemsets* candidatos e a quantidade de passagens pela base de dados.

2.6.3 Algoritmo GFP-GROWTH

O algoritmo GFP-GROWTH (PIZZI, 2006) aplica o conceito de agrupamentos no algoritmo FP-GROWTH para possibilitar a extração de padrões multirrelacionais. O seu funcionamento é similar ao do APRIORI-GROUP, ou seja, é efetivada inicialmente a junção das tabelas que farão parte da análise e, a partir desta tabela alvo, são extraídos os padrões levando em consideração a existência de agrupamentos. Assim como no APRIORI-GROUP, o cerne do GFP-GROWTH consiste em identificar os agrupamentos existentes na base de dados e efetuar a mineração dos padrões com base nesses agrupamentos. O pseudocódigo do GFP-GROWTH é mostrado na Figura 2.11.

```
Algoritmo: GFP-Growth
Entradas: Base de dados D; suporte mínimo minsup; confiança mínima minconf
Saída: Itemsets frequentes L

// Definições de variáveis
// G: conjunto de agrupamentos
// ftp: estrutura GFP-tree
// FP: conjunto de padrões frequentes

1)   G = findGroups(D)
2)   ftp = geraGFPTree(null)
3)   FP = GFP-Growth( ftp, null )
4)   para cada elemento de FP
5)        gere as regras envolvendo seus itens tal que conf_a >= minconf
```

Figura 2.11 Algoritmo GFP-Growth (PIZZI, 2006)

A etapa de geração dos agrupamentos (linha 1) consiste em organizar as transações da base de dados de acordo com o valor de seus IDs. Se já houver algum agrupamento com o ID da transação em questão, a mesma é incluída neste agrupamento. Caso contrário, um novo agrupamento é criado e a transação é inserida neste agrupamento.

A segunda etapa do algoritmo é a construção da *GFP-tree* (linha 2), que ocorre de maneira similar ao da *FP-tree*, com exceção da necessidade de representação dos agrupamentos nos nós da árvore e na tabela *header*. Na Figura 2.12 é apresentado um exemplo de *GFP-tree*, no qual é possível observar a existência de uma lista de agrupamentos para cada nó da árvore, relacionando o conjunto de agrupamentos em que determinado item ocorreu.

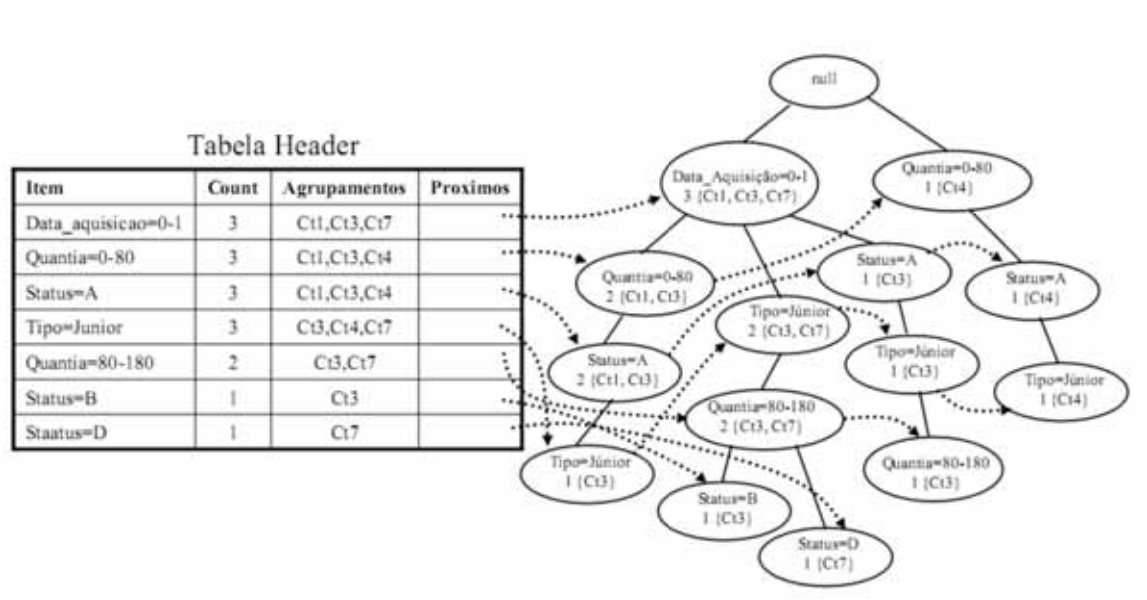

Figura 2.12 Exemplo de GFP-tree (PIZZI, 2006)

A lista de agrupamentos presente na tabela *header* indica quais agrupamentos estão relacionados com todas as ocorrências (nós) de um dado item. Ou seja, essa lista consiste na união das listas de agrupamentos de todos os nós correspondentes ao item. Por exemplo, o item "Quantia=80-180" ocorre em dois nós da GFP-tree da Figura 2.12, sendo que o primeiro apresenta a lista de agrupamentos {Ct3, Ct7} e o segundo apresenta a lista {Ct3}. Embora esse item ocorra em três transações, o que se considera é sua ocorrência nos agrupamentos, de modo que múltiplas ocorrências dentro de um mesmo agrupamento são contabilizadas como uma única ocorrência, tal como ocorre no APRIORI-GROUP. Dessa forma, é realizada uma união das duas listas de agrupamentos, resultando na lista {Ct3, Ct7}, que é armazenada na tabela *header* para consultas posteriores. A frequência de ocorrência do item é, na verdade, expressa pela quantidade de agrupamentos presente nesta lista. No item "Quantia=80-180", por exemplo, o contador é igual a 2, que é de fato o número correto de ocorrências deste item na base de dados.

Após a construção da *GFP-tree*, a mesma é passada como parâmetro para que se possa dar início ao procedimento de extração dos padrões frequentes (linha 3). Essa etapa é semelhante a do algoritmo FP-GROWTH, na qual os padrões vão sendo obtidos por meio das *GFP-tree* condicionais, que são construídas e mineradas recursivamente. Uma vez obtidos todos os *itemsets* frequentes, o algoritmo então gera as regras de associação cujo valor de confiança seja maior do que o limitante previamente estabelecido (linha 4 e 5).

O algoritmo GFP-GROWTH, assim como o APRIORI-GROUP, necessita de uma etapa de pré-processamento para reunir as informações em uma única tabela, incorrendo nas limitações já mencionadas como o alto custo computacional e a possibilidade de perdas semânticas. Além disso, o algoritmo GFP-GROWTH realiza a tarefa de mineração de padrões utilizando uma estrutura (*GFP-tree*) armazenada em memória. Se por um lado isso reduz a necessidade de acessos a disco, por outro torna o algoritmo limitado à quantidade de memória disponível, o que se torna um grande problema à medida que se aumenta o tamanho da base de dados a ser analisada.

2.6.4 Algoritmo APRIORIMR

O algoritmo APRIORIMR foi idealizado para efetuar a mineração de padrões diretamente em bases de dados relacionais. Esse algoritmo foi concebido para ser utilizado juntamente com uma ferramenta de mineração de dados multirrelacional (OYAMA, 2006). O APRIORIMR é uma adaptação do algoritmo APRIORI ao contexto multirrelacional, possibilitando a extração de padrões sem a necessidade de um pré-processamento para a junção das múltiplas tabelas.

De forma análoga aos algoritmos APRIORI-GROUP (RIBEIRO; VIEIRA; TRAINA, 2005) e GFP-GROWTH (PIZZI, 2006), o APRIORIMR utiliza o conceito de agrupamentos para a extração de padrões multirrelacionais, ajustando os valores das medidas de interesse para que contemplem os grupos e, assim, obtenham padrões que sejam precisos e reflitam o conteúdo armazenado na base de dados.

Para isso, define-se uma das tabelas da base de dados como a tabela de agrupamento – geralmente é aquela que permite individualizar os registros, como por exemplo, tabela de clientes, de pacientes etc. – a qual é utilizada como referência na etapa de cálculo do suporte e teste dos *itemsets* frequentes. É justamente na etapa de

geração do conjunto de *itemsets* frequentes que se concentram as principais modificações introduzidas pelo algoritmo APRIORIMR.

O processo de obtenção dos *itemsets* frequentes inicia-se com o recebimento de um conjunto de *itemsets* candidatos. O valor de suporte para cada candidato é, então, calculado por meio da seguinte consulta SQL que retorna o número de ocorrências do *itemset* especificado.

```
SELECT COUNT( DISTINCT pkTAg1, pkTAg2, ..., pkTAgN )
FROM listaTabelas WHERE listaPk_Fk AND listaCondições          (2.3)
```

Os termos pkTAg1, pkTAg1, ..., pkTAgN são as chaves primárias da tabela de agrupamento. O uso da sentença "COUNT(DISTINCT ...)" nesses termos permite que a consulta efetue a contagem do suporte levando-se em conta a ocorrência de agrupamento. Ou seja, múltiplas ocorrências encontradas para o mesmo identificador de grupo - no caso a chave primária da tabela de agrupamento - são ignoradas, sendo computada apenas uma única ocorrência.

Na cláusula FROM há a listagem das tabelas com as quais o *itemset* em questão se relaciona. O termo "listaCondições", por sua vez, representa o conjunto de condições da forma "tabela.atributo=valor", sendo uma condição para cada item do *itemset*. E, por fim, a "listaPk_Fk enumera as expressões que relacionam duas tabelas por meio da relação entre chaves estrangeiras e chaves primárias.

O algoritmo APRIORIMR baseia-se no algoritmo APRIORI e, por isso, herda suas principais características. A principal limitação decorre justamente do fato de seu funcionamento basear-se na abordagem de "geração-e-teste" de candidatos, o que compromete o desempenho, tendo em vista a geração de um número elevado de *itemsets*. Analogamente ao algoritmo APRIORI-GROUP, o algoritmo realiza junção das tabelas para a obtenção dos valores de suporte, incorrendo nos problemas já mencionados de custo computacional e perdas semânticas.

2.7 Considerações finais

Nesse capítulo, foram abordados os princípios da extração de regras de associação, que é uma das tarefas mais importantes da mineração de dados. Foram

apresentados os dois mais difundidos algoritmos de mineração de regras de associação: APRIORI e FP-GROWTH. Em seguida, foi apresentado um panorama da mineração de regras de associação, relacionando os principais algoritmos tradicionais existentes.

Na segunda parte do capítulo, abordou-se a mineração de dados multirrelacional, que apresenta a característica de possibilitar a extração de padrões provenientes de múltiplas tabelas. Essa abordagem possui ampla aplicação, principalmente, em contextos em que a disposição estrutural da base de dados possui considerável valor semântico. Para essa abordagem, foram discutidos os principais algoritmos existentes na literatura, apresentando-se as vantagens e as limitações de cada proposta. Com isso, foram fundamentados os principais conceitos utilizados no trabalho, bem como foram apresentados os principais algoritmos do estado da arte, tanto os tradicionais quanto os multirrelacionais.

Capítulo 3
Mineração de dados multirrelacional em grandes bases de dados

3.1 Considerações iniciais

A maioria dos dados oriundos das mais diversas áreas de conhecimento fica armazenada em bases de dados relacionais (YIN, 2007). Especificamente, as grandes aplicações – tais como redes de varejo, instituições financeiras, indústrias e o próprio meio acadêmico – geram e armazenam imensos volumes de dados. A análise dessas bases é humanamente improvável e, muitas vezes, mesmo com o uso das técnicas de mineração existentes ainda não é possível finalizar a busca de padrões em bases de dados dessa dimensão, quer seja pela ineficiência em termos de tempo computacional, quer seja pela inabilidade inata da técnica - a qual pode apresentar limitações que impossibilitam a manipulação de grandes quantidades de dados e padrões.

Neste capítulo, é apresentado o algoritmo multirrelacional proposto para mineração de regras de associação em grandes bases de dados relacionais - denominado MR-RADIX -, bem como são expostos os conceitos e as propostas correlatas e relevantes para o desenvolvimento do trabalho.

3.2 Definição do problema

Com o surgimento das propostas de algoritmos multirrelacionais, tornou-se possível a extração eficiente de regras de associação em SGBDs relacionais. Esse fato é de suma importância, visto que as bases de dados relacionais caracterizam-se como importantes e difundidas fontes de armazenamento de dados, sendo utilizadas por aplicações dos mais variados tipos e dimensões, desde ferramentas computacionais simples até complexos sistemas que geram e manipulam grandes volumes de dados.

A mineração de grandes bases de dados apresenta dificuldades particulares, uma vez que um algoritmo que apresenta um excelente desempenho pode não ser capaz de completar o processamento dessa quantidade de dados. Basta verificar que as bases de dados de tamanho elevado possuem milhões de registros e *gigabytes* ou até mesmo *terabytes* de dados armazenados. Com isso, o número de padrões que podem ser gerados por um algoritmo de mineração de regras de associação excede a capacidade de memória disponível.

A título de exemplo, considere que o algoritmo APRIORI seja utilizado para a mineração de *itemsets* frequentes de tamanho 40, ou seja, o *itemset* é formado por 40 itens. Para completar essa tarefa, faz-se necessário gerar e testar $2^{40} \approx 10^{12}$ *itemsets* candidatos. Considerando que cada *itemset* ocupe o espaço de memória correspondente a um inteiro de 4 (quatro) bytes, obtém-se que o espaço total de memória necessária para o armazenamento de todos os *itemsets* candidatos é $4x10^{12}$ bytes ou 4 terabytes. Evidentemente que essa quantidade de memória necessária ao processamento excede a quantidade atual de memória disponível nos equipamentos convencionais.

Mas cabe salientar que, para possibilitar a mineração de regras de associação, há algumas propostas da abordagem tradicional que focam o fator escalabilidade, de modo que o tempo de execução de um algoritmo seja proporcional à dimensão do conjunto de dados (GANTI; GEHRKE; RAMAKRISHNAN, 1999). Essas propostas fazem uso de novas estruturas de dados para otimizar o uso do espaço de memória e apresentar algoritmos mais eficientes para a busca de padrões frequentes.

O algoritmo CT-PRO, por exemplo, apresenta uma estrutura de dados denominada *Compressed FP-Tree* ou *CFP-Tree*, que reduz em até 50% o espaço em

memória ocupado quando comparada com a *FP-tree* convencional. Outro algoritmo que propõe uma otimização nesse sentido é o FP-GROWTH-TINY, que chega a reduzir 2,4 vezes o espaço de memória necessário em relação ao FP-GROWTH. O algoritmo PATRICIAMINE, por sua vez, faz uso da estrutura Patricia-trie, a qual consiste em uma modificação da *trie* convencional que contempla estratégias de compressão dos nós. Essa estrutura provou-se muito eficiente quanto ao requerimento de espaço de memória, sobretudo em grandes volumes de dados. Em um teste comparativo, foi mostrado que, para uma tabela contendo aproximadamente 400.000 registros, a estrutura *Patricia-trie* ocupa cerca de 41 MB, enquanto que uma árvore *trie* convencional ocupa mais de 163 MB (PIETRACAPRINA; ZANDOLIN, 2003).

Tais propostas reduzem substancialmente o espaço de memória requerido, possibilitando que uma quantidade de dados maior possa ser processada com os mesmos recursos computacionais. Porém, para bases de dados significativamente grandes, a otimização do uso da memória por si só não garante a integralidade do processamento, tendo em vista que ainda haverá a questão da memória limitada. Considere-se, por exemplo, que o algoritmo possua escalabilidade linear, de modo que a relação entre o tamanho do conjunto de dados e o tamanho da estrutura mantenha-se sempre proporcional. Imagine que a tabela citada anteriormente, passe de 400.000 (quatrocentos mil) para 400.000.000 (quatrocentos milhões) de registros. Seguindo a proporcionalidade, o espaço ocupado pela Patricia-trie seria também multiplicado por 1000 (mil), ou seja, necessitaria de aproximadamente 40 GB de memória para sua representação.

Para possibilitar uma mineração de dados dessa dimensão, as propostas existentes subdividem os dados iniciais em porções cujo tamanho permita o processamento em memória. Tal estratégia denomina-se particionamento e foi inicialmente proposta pelo algoritmo PARTITION (SAVASERE; OMIECINSKI; NAVATHE, 1995), porém já foram realizados estudos para adaptá-la a outros algoritmos de mineração em uma única tabela, tais como o H-MINE (PEI et al., 2001, 2007) e o FP-GROWTH* (GRAHNE; ZHU, 2004).

As propostas citadas de mineração de grandes volumes de dados pertencem à abordagem tradicional, de modo que consideram apenas a análise de padrões envolvendo uma única tabela ou arquivo. No contexto multirrelacional, por outro lado, é limitado o estudo de algoritmos eficientes e escaláveis para a mineração de

grandes bases de dados relacionais. A maioria dos algoritmos multirrelacionais existentes focam essencialmente no desempenho, não levando em conta o problema do processamento de grandes volumes de dados em limitados espaços de memória.

3.3 Visão geral da proposta

O algoritmo proposto visa à mineração de grandes bases de dados relacionais. Para isso, o mesmo baseia-se no algoritmo tradicional PATRICIAMINE (PIETRACAPRINA, ZANDOLIN, 2003), estendendo-o ao contexto multirrelacional.

A escolha desse algoritmo é justificada pelo fato de que sua proposta faz uso de uma estrutura de dados mais eficiente, favorecendo ganhos no desempenho e a otimização do uso do espaço em memória. Cabe salientar que essas se tornam características cada vez mais importantes à medida que o processamento envolve bases de dados de dimensões substancialmente grandes.

O algoritmo PATRICIAMINE foi selecionado após um levantamento prévio – de leitura e estudo - de diversas propostas presentes na literatura atual. O referido algoritmo, em vários experimentos, apresentou resultados mais favoráveis e melhor desempenho, ainda quando comparado com o algoritmo OPPORTUNEPROJECT (LIU *et al*., 2002), considerado um dos mais avançados algoritmos da abordagem *pattern-growth* (PIETRACAPRINA, ZANDOLIN, 2003; GOPALAN; SUCAHYO, 2004; SHANG, 2005). A título de comparação, observe o gráfico da Figura 3.1, que apresenta um comparativo dos tempos de execução desses dois algoritmos para uma base de dados denominada *Pumsb*. Tal base é composta de aproximadamente 50 mil registros. Para uma melhor visualização dos gráficos, utiliza-se a escala logarítmica para o eixo Y, correspondente ao tempo de execução.

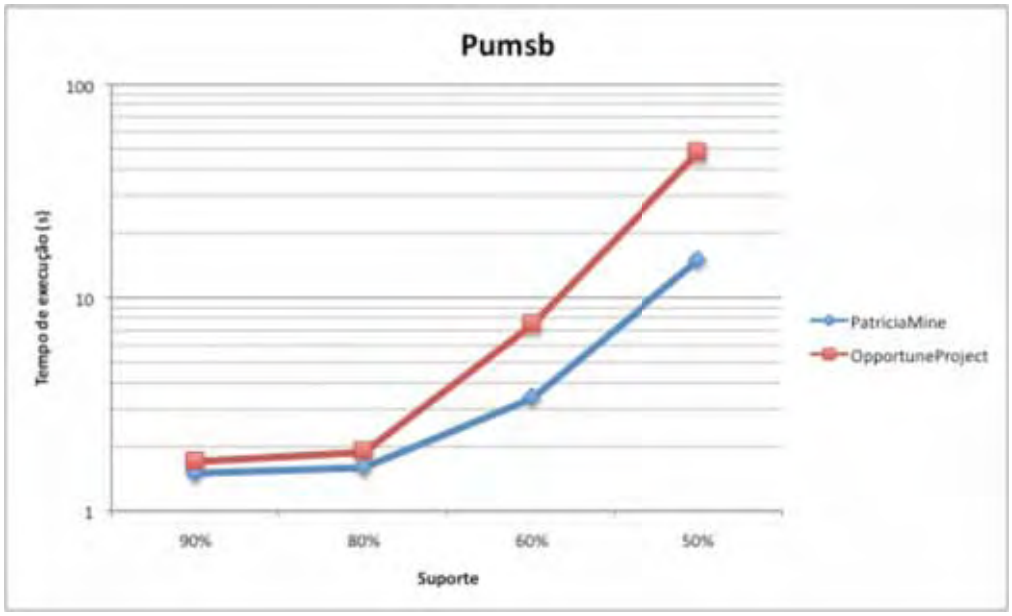

Figura 3.1 Gráfico do tempo de execução dos algoritmos PATRICIAMINE e OPPORTUNEPROJECT (PIETRACAPRINA; ZANDOLIN, 2003)

Salienta-se que à medida que se reduz o valor do suporte, mais *itemsets* tornam-se frequentes e devem ser analisados, o que acarreta o aumento do tempo de execução. Dessa forma, é possível visualizar no gráfico acima que o algoritmo PATRICIAMINE apresenta, no geral, melhor desempenho que o OPPORTUNEPROJECT. A diferença no tempo de execução vai tornando-se ainda mais expressiva à medida que o suporte é reduzido.

Na Figura 3.2 é apresentado o gráfico comparativo de tempo de execução do algoritmo PATRICIAMINE com relação aos dois principais representantes de algoritmos de mineração de regras de associação, a saber, APRIORI e FP-GROWTH. Também consta no gráfico o tempo de execução referente ao algoritmo FP-GROWTH* (GRAHNE; ZHU, 2005), o qual se caracteriza como uma implementação mais otimizada do algoritmo FP-GROWTH. A base de dados, nesse caso, é a *Connect*, a qual contém mais de 67 mil registros.

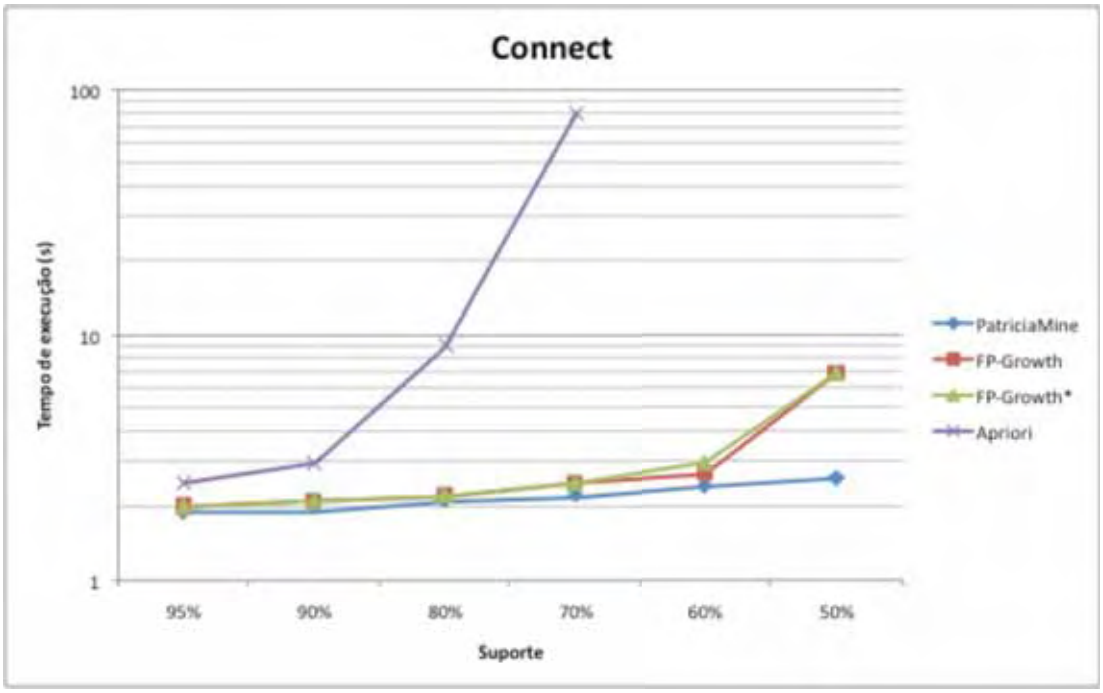

Figura 3.2 Gráfico comparativo entre PATRICIAMINE, FP-GROWTH, FP-GROWTH* e APRIORI (GRAHNE; ZHU, 2005)

Nota-se que, para valores mais altos de suporte, maiores que 60%, as duas propostas baseadas no FP-GROWTH apresentam um desempenho ligeiramente inferior ao do algoritmo PATRICIAMINE. Para valores menores de suporte, ou seja, inferiores a 50%, a diferença no desempenho acentua-se. Enquanto que o tempo de execução do PATRICIAMINE mantém-se com um aumento linear, nos demais algoritmos o tempo apresenta crescimento exponencial.

O algoritmo APRIORI, como pode ser verificado no mesmo gráfico, apresentou tempos de execução elevados em todas as faixas de valores de suporte. Para valores de suporte superior a 70%, os seus tempos de execução foram tão altos que sequer se encontram relacionados.

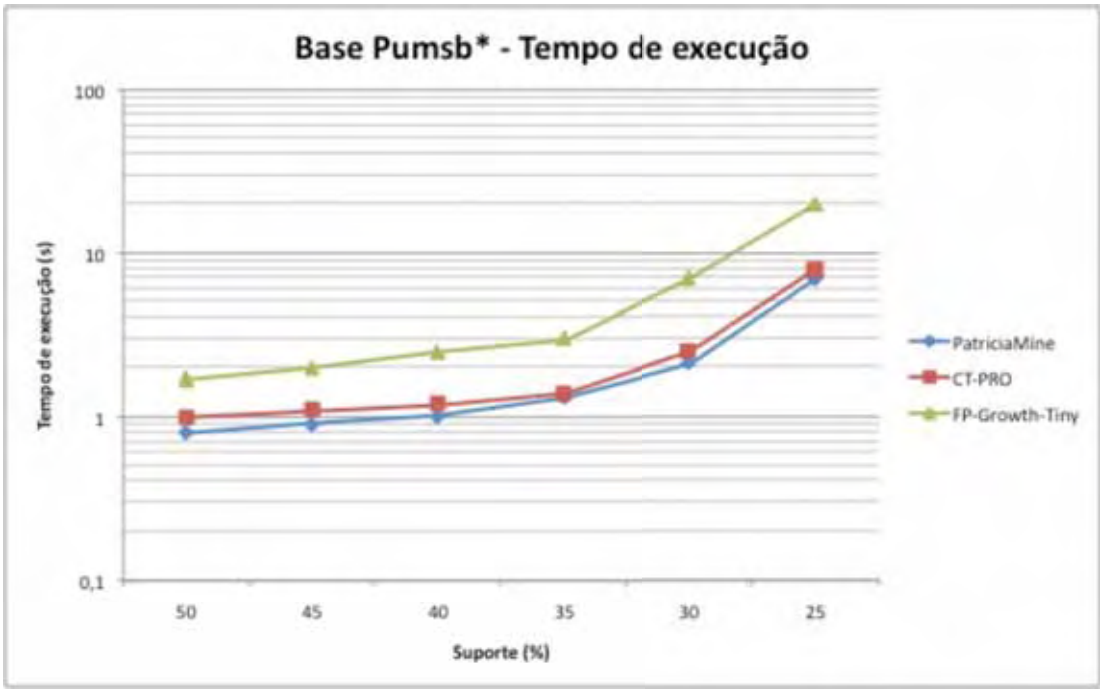

Figura 3.3 Gráfico comparativo do tempo de execução dos algoritmos PATRICIAMINE, CT-PRO e FP-GROWTH-TINY

O gráfico da Figura 3.3, por sua vez, apresenta os tempos de execução, comparando o PATRICIAMINE com algoritmos recentes de desempenho similar. Tais resultados comparativos foram obtidos por meio da consulta ao *website* do *IEEE Workshop on Frequent Itemset Mining Implementations (FIMI'04)*[5].

Cabe acrescentar que a base de dados referente ao teste é a *Pumsb** - uma modificação da *Pumsb* - na qual foram removidos os itens com alto número de ocorrências. Embora os algoritmos apresentem tempos de execução bem próximos, o algoritmo PATRICIAMINE se sobressai em relação aos demais quando observado o quesito espaço de memória principal necessária para a efetivação da mineração, tal como ilustrado na Figura 3.4. Essa característica positiva possui relação direta com a estrutura de dados *Patricia-trie*, a qual representa com mais eficiência o conjunto de itens frequentes na memória.

[5] Disponível em http://fimi.cs.helsinki.fi/. Acesso em 30 de agosto de 2009.

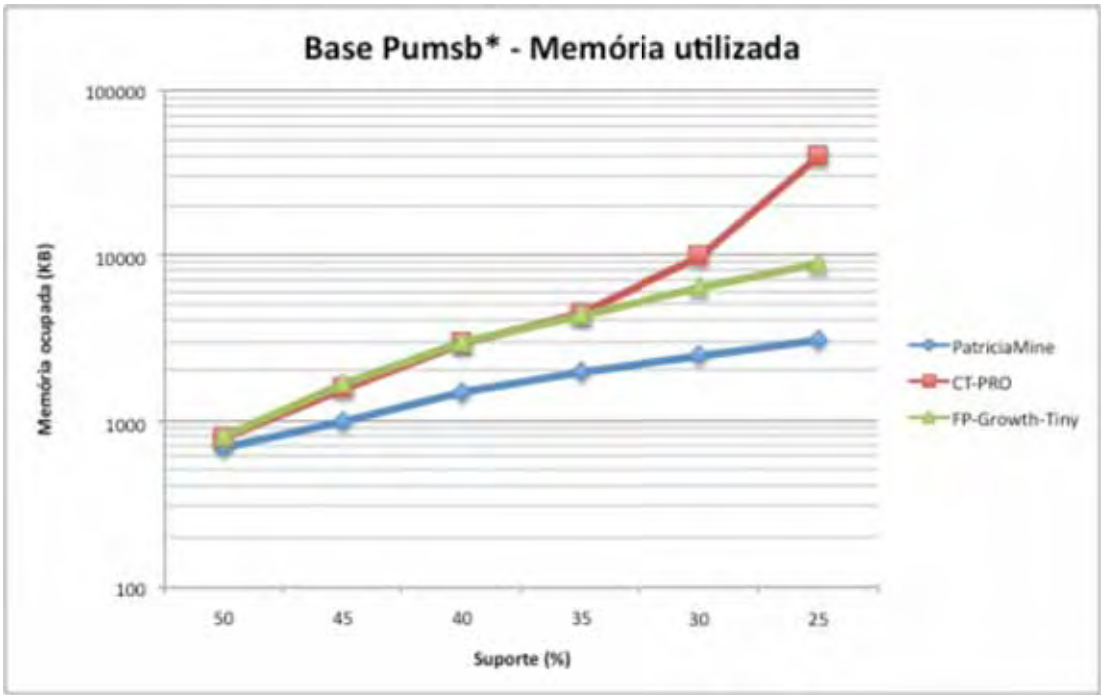

Figura 3.4 Gráfico comparativo do espaço de memória ocupado dos algoritmos PATRICIAMINE, CT-PRO e FP-GROWTH-TINY

Com isso, verificou-se que o algoritmo tradicional PATRICIAMINE apresenta bons resultados de eficiência e consumo de memória para diferentes configurações de bases de dados e valores de suporte, apresentando, inclusive, melhor desempenho que o algoritmo OPPORTUNEPROJECT, considerado pela literatura como uma das mais eficientes propostas pertencentes ao estado da arte.

Justifica-se, então, a escolha do algoritmo PATRICIAMINE como algoritmo base deste trabalho, uma vez que, comprovadamente, mostrou-se mais eficiente que algoritmos correlatos. Esse algoritmo, dentre o pesquisados, apresentou o melhor panorama em termos de tempo de execução e consumo de memória.

Com base no PATRICIAMINE, efetuou-se, então, o estudo de uma proposta de algoritmo multirrelacional denominado MR-RADIX. A mineração de padrões multirrelacionais por meio do algoritmo proposto foi possível com a utilização de uma nova estratégia denominada “fusão de itens secundários”. A estratégia possibilita a extração de padrões multirrelacionais sem a necessidade da operação de junção física entre tabelas, que é uma tarefa computacionalmente custosa. Além disso, o número de estruturas de dados não aumenta pelo fato de estar lidando com múltiplas tabelas. Todas as operações de busca de padrões são realizadas em uma

mesma estrutura *Radix-tree*, diferentemente do algoritmo MRFP-GROWTH, por exemplo, que cria uma estrutura independente para cada tabela da base de dados.

Complementando o desenvolvimento do algoritmo multirrelacional, foi elaborado um modelo de representação para permitir o manuseio dos itens provenientes de bases de dados relacionais. Ademais, foi proposta uma nova estrutura de dados denominada *ItemMap*, que tem como objetivo mapear os itens relacionais em identificadores, os quais ocupam menos espaço em memória e otimizam o armazenamento e a manipulação de nós na *Radix-tree*. A proposta também contempla a estratégia de particionamento dos dados, tal como apresentado no algoritmo PARTITION, possibilitando a mineração de bases de dados de grandes dimensões.

3.4 Estratégia para mineração de grandes bases de dados

A mineração de grandes bases de dados necessita de atenção específica, pois um algoritmo que funciona adequadamente para pequenas e médias bases pode não ser uma solução viável para tratar grandes volumes de dados. Isso porque a maioria dos algoritmos multirrelacionais existentes, tal como o MRFP-GROWTH, fundamentam seu bom desempenho no uso de estruturas de dados que representem compactamente a base de dados em memória.

O particionamento é uma estratégia que permite a mineração de grandes bases de dados quando o tamanho das estruturas de dados (conjuntos de candidatos ou *FP-tree*, por exemplo) excede a quantidade de memória disponível. Tal estratégia consiste na divisão da base de dados em unidades menores, que sejam disjuntas entre si. O tamanho dessas unidades é definido de forma que a extração de todos os padrões possa ser realizada dentro da disponibilidade de memória. De forma geral, podemos resumir o processo de mineração de padrões utilizando o particionamento em duas fases, tal como mostrada na Figura 3.5.

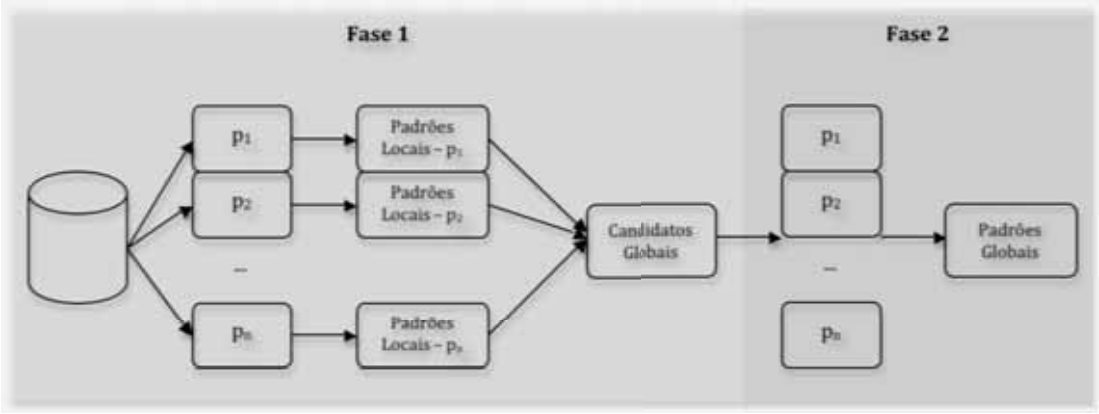

Figura 3.5 Fases do particionamento

Na primeira fase, cada partição é minerada para a obtenção dos *itemsets* frequentes locais. A segunda fase consiste no cálculo dos valores de suporte dos *itemsets* locais encontrados anteriormente agora referente a toda a base de dados, obtendo, assim, os *itemsets* frequentes globais.

3.5 Algoritmo MR-RADIX

O algoritmo multirrelacional proposto denomina-se MR-RADIX e caracteriza-se como uma proposta para mineração de regras de associação em grandes bases de dados relacionais. O MR-RADIX pertence à abordagem *pattern-growth* (PG), cujo principal representante é o algoritmo FP-GROWTH, efetuando a tarefa de mineração sem as etapas de geração e teste de candidatos.

A proposta utiliza a estrutura de dados *Radix-tree* ou também referenciada como *Patricia-trie* para a representação da base de dados. Tal estrutura proporciona eficiência tanto no desempenho do algoritmo de mineração quanto na utilização do espaço em memória. A *Radix-tree* apresenta alta capacidade de compressão, reduzindo em até 75% o espaço necessário para a sua alocação em memória quando comparada com a *FP-tree*, que é a estrutura *trie* padrão utilizada em diversos algoritmos da abordagem PG (PIETRACAPRINA; ZANDOLIN, 2003).

Alguns algoritmos, tais como o H-MINE e o OPPORTUNEPROJECT, fazem uso de dois tipos de estruturas para a mineração de bases de dados densas e esparsas. Para as bases densas são utilizadas as *tries*, uma vez que esse tipo de base gera poucos ramos e intensifica a eficiência da mineração. Por outro lado, as bases de dados esparsas geram muitos ramos na árvore, aumentando significativamente o

tamanho dessas estruturas e, consequentemente, o tempo necessário para o seu processamento. Para essas bases, o uso de estruturas baseadas em *arrays* apresentam melhores resultados. Por esse motivo, tais algoritmos utilizam-se de heurísticas que avaliam a densidade da base de dados e efetuam a escolha da melhor estratégia.

A estrutura *Radix-tree*, como já foi mencionado, comprime a representação da base de dados e, dessa forma, apresenta-se como uma alternativa para a mineração de bases de dados relacionais, sejam densas ou esparsas. O problema da esparsidade da base de dados, que tende a aumentar o tamanho da estrutura, é contornado com a eficiente compressão dos nós.

O algoritmo MR-RADIX segue a mesma linha do algoritmo TD-FP-GROWTH, efetuando a mineração utilizando uma estratégia *Top-Down*, ou seja, os padrões frequentes são extraídos percorrendo-se a *Radix-tree* a partir do nó raiz em direção aos nós-folhas. O FP-GROWTH e outros algoritmos derivados utilizam a estratégia *Bottom-Up*, partindo dos nós-folhas em direção ao topo da estrutura. Com a estratégia *Top-Down*, obtém-se um considerável ganho no desempenho, uma vez que há uma única *Radix-tree* e todo o processamento é feito com base nesta estrutura global. Com isso, elimina-se a carga extra de processamento que seria dispensada à construção de estruturas intermediárias e temporárias.

Além disso, o algoritmo MR-RADIX , assim como o PATRICIAMINE, apresenta uma estratégia iterativa para a exploração da árvore, o que o torna diferente em relação aos que apresentam propostas recursivas – característica da maioria dos algoritmos relativos ao tema, tais como FP-GROWTH e seus derivados. Com isso, gera-se um ganho no desempenho, uma vez que se elimina o custo computacional com as sucessivas chamadas recursivas.

3.5.1 Representação de *itemsets* relacionais

A mineração de regras de associação na abordagem tradicional considera que os dados a serem minerados estão dispostos em apenas uma tabela. Esta estrutura não precisa necessariamente estar no formato de uma tabela relacional, em que os registros possuem o mesmo número de colunas ou atributos. Geralmente, a mesma baseia-se no formato transacional, tal como exemplificada na Tabela 3.1.

Tabela 3.1 Exemplo de uma base de dados transacional

TID	Itens
1	Pão, Leite, Manteiga
2	Pão
3	Leite, Ovos
4	Manteiga, Frutas
5	Leite, Manteiga, Cereal
6	Leite
7	Pão, Leite, Manteiga, Frutas
8	Leite, Cereal

No caso da mineração envolvendo uma única tabela, a representação dos *itemsets* é trivial, pois a única informação relevante e necessária para a extração de padrões é o próprio identificador de cada item. Na Tabela 3.1, por exemplo, os *itemsets* seriam tratados pelo algoritmo de mineração na forma de conjuntos de identificadores de itens, tais como {Pão, Leite, Manteiga}, {Manteiga, Frutas}, e assim por diante.

Para a mineração de dados provenientes de múltiplas tabelas relacionais, a representação dos *itemsets* torna-se um pouco mais complexa, pois há a necessidade de armazenamento de um número maior de informações devido à multiplicidade de fontes de dados. Por esse motivo, foi proposto um modelo de representação denominado *itemset* relacional, que estende o conceito de *itemset* tradicional, incluindo informações para permitir a identificação de padrões provenientes de bases de dados relacionais.

Na Figura 3.6 pode ser visto o esquema de um *itemset* relacional contendo *k* itens, cujos índices variam de 0 a k-1. Cada item relacional é composto de três campos identificadores. O primeiro campo *Tabela* indica o nome da tabela de origem desse item. O segundo campo *Nome do atributo* armazena a identificação da coluna associada a esse item. Por fim, o campo *Valor do atributo* representa o conteúdo na coluna *Nome do atributo* para um dado registro da tabela de origem.

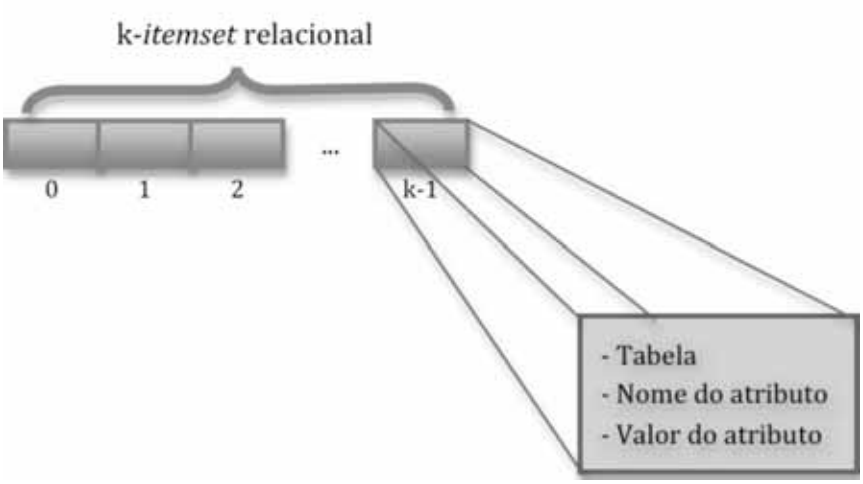

Figura 3.6 *Itemset* relacional

Essa representação de um item relacional, por meio da utilização dos três campos citados, possibilita identificar unicamente um item na base de dados. Para facilitar a visualização de um item relacional, sempre que conveniente, será utilizada a notação **Tabela.NomedoAtributo=ValordoAtributo**.

Desse modo, a remoção de qualquer um dos campos pode tornar o item ambíguo ou incompleto. Para demonstrar esse fato, considere a Figura 3.7, que apresenta uma base de dados contendo as relações *Paciente* e *Internação*. Tome como exemplo o item "Internação.Hemorragia=Sim". A remoção do nome ou do valor do atributo acarretaria um item incompleto, sem significado para a mineração de regras de associação. Por outro lado, se a informação da tabela de origem for eliminada da representação do item, o mesmo ainda continua apresentando algum significado.

Paciente

ID_Pac	Nome	Sexo
011	João	M
012	Maria	F
013	César	M
...	...	...

1 — n

Internação

ID_Intern	ID_Pac	Hemorragia	Inconsciente
0201	011	Sim	Sim
0202	012	Não	Sim
0203	012	Não	Não
0204	013	Sim	Sim
...	...	...	...

Figura 3.7 Exemplo de base de dados relacional

Os itens "Hemorragia=Sim" e "Inconsciente=Não", por exemplo, representam corretamente itens existentes na tabela Internação. A necessidade do campo *Tabela*, então, se justifica pelo fato de a mineração ser multirrelacional, ou

seja, diferentes tabelas podem possuir nomes e valores de atributos similares. Veja o caso do atributo *ID_Pac* que existe tanto na tabela *Paciente* quanto na tabela *Internação*. O item "ID_Pac=012" é ambíguo e só pode ser devidamente representado com a especificação da sua tabela de origem, tal como em "Paciente.ID_Pac=012".

3.5.2 Estrutura *Radix-tree*

Primeiramente, cabe ressaltar que a estrutura *Radix-tree* – ou *Patricia-trie* – foi desenvolvida inicialmente por Morrison (1968). O objetivo inicial em sua proposta consistia em melhorar as características da *trie* padrão por meio do adicionamento, em sua estrutura, da capacidade de compressão dos nós.

Essa proposta já foi utilizada em vários trabalhos na área de mineração de regras de associação (PIETRACAPRINA; ZANDOLIN, 2003; HAMIDA, 2005; HAMIDA; SLIMANI, 2005). A compressão nessa estrutura é realizada por meio do agrupamento dos nós que compartilham o mesmo ramo, e, concomitantemente, apresentam o mesmo valor para o contador de suporte. A Figura 3.8 ilustra um exemplo de uma *trie* padrão enquanto que a Figura 3.9 apresenta a *Radix-tree* correspondente. Ao observar a primeira delas, é possível constatar que os nós são representados de forma individual. Com isso, há a necessidade de que haja 12 (doze) nós para que a representação da base de dados seja construída.

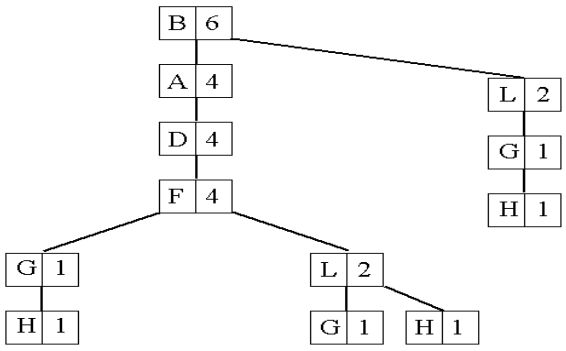

Figura 3.8 Exemplo de *Trie* Padrão

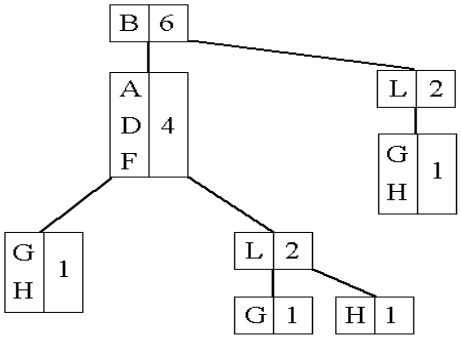

Figura 3.9 Exemplo de *Radix Tree*

Por sua vez, a Figura 3.9, que ilustra a *Radix-tree*, mostra que essa estrutura necessita de apenas 8 (oito) nós para a representação da mesma base de dados. Desse modo, gera-se um ganho de espaço em memória de 33% (trinta e três por cento). Assim, para efetuar essa compressão, a estrutura agrupa, como já foi mencionado, os nós que compartilham o mesmo ramo e possuem o mesmo valor do contador de suporte. Esse é o caso, por exemplo, dos nós A, D e F, os quais possuem o mesmo valor 4 (quatro).

Além da árvore propriamente dita, a *Radix-tree* apresenta uma estrutura auxiliar denominada *ItemList* (IL) que tem como objetivo indexar os itens frequentes e conectar todos os nós de um mesmo item, de modo análogo às tabelas *header* existentes no algoritmo FP-GROWTH.

A *ItemList*, portanto, possui três campos: i) o identificador do item; ii) o contador de suporte total; e iii) um ponteiro para a lista encadeada de nós. A Figura 3.10 exibe a *ItemList* para a *Radix-tree* da Figura 3.9.

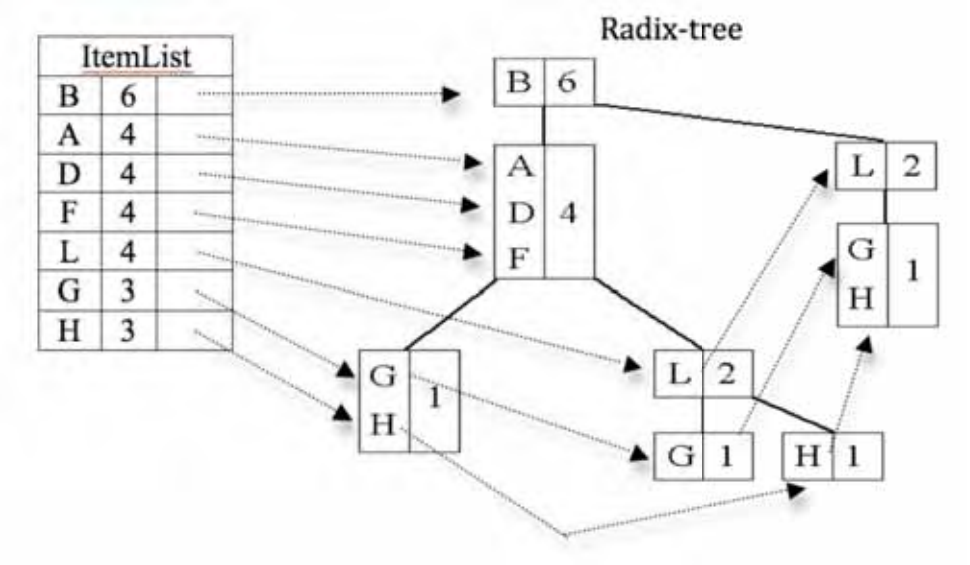

Figura 3.10 Exemplo das estruturas *ItemList* e *Radix-tree*

A etapa mais complexa da *Radix-tree* consiste no processo de sua construção, uma vez que a montagem da estrutura gera a necessidade de que hajam possíveis subdivisões de nós. Assim, há três casos distintos de inserção de um novo padrão na estrutura em que ocorrem tais divisões:

- **Caso 1: O novo padrão está contido no conjunto de itens de um dado nó**. Nesse caso, a porção equivalente entre o padrão e os itens do nó é mantida no nó em questão e a porção restante dos seus itens forma um novo nó filho;
- **Caso 2**: **O novo padrão apresenta correspondência parcial com os itens do nó**. Novamente, a porção em que há equivalência é mantida no nó em questão e as porções divergentes formam dois nós filhos sendo um para a porção restante do padrão e um para a porção restante do nó;
- **Caso 3**: **O novo padrão contém o nó**. De modo análogo ao primeiro caso, é mantida, no nó em questão, a porção em que há correspondência. A porção restante do nó, por sua vez, acaba por gerar um nó filho.

Considere o conjunto de dados da Tabela 3.2, que apresenta exemplos de registros contendo itens relacionais, os quais são ilustrados por letras com a finalidade de facilitar o entendimento da leitura dos dados. Os registros serão

submetidos à representação na *Radix-tree*, a fim de exemplificar o procedimento de construção e subdivisão de nós.

Tabela 3.2 Exemplo de Conjunto de Dados

TID	Itens
1	B A D F G H
2	B L
3	B A D F L H
4	B A D F L G
5	B L G H
6	B A D F

O primeiro passo da construção consiste na criação de um novo nó contendo todos os itens de um registro. Considerando-se, por exemplo, o primeiro registro elencado na tabela, tem-se que o nó é composto pelos itens {B, A, D, F, G, H}, o qual é inserido na estrutura *Radix-tree* a partir do nó raiz. Como esse é o primeiro nó inserido na estrutura, não houve necessidade de divisão de nós.

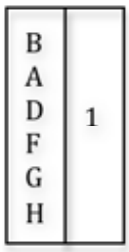

Figura 3.11 Situação da estrutura após inserção do primeiro nó

Passando-se para o segundo registro, cria-se um nó formado pelos seus itens {B, L}, de modo análogo ao realizado no processo descrito anteriormente. No entanto, ao tentar inserir esse nó na estrutura, observa-se que o nó atual e o nó existente compartilham apenas o primeiro item, ‘B’. Dessa forma, há uma correspondência parcial entre os nós envolvidos, tornando possível a aplicação do caso 2, anteriormente descrito. O resultado da subdivisão pode ser visto na Figura 3.12.

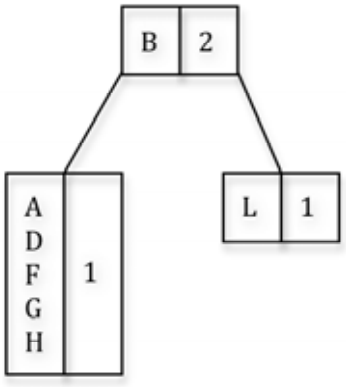

Figura 3.12 Situação da estrutura após inserção do segundo nó

Prosseguindo com a construção, gera-se, agora, um nó referente ao terceiro registro {B, A, D, F, L, H}. Nesse caso, há, inicialmente, a correspondência parcial envolvendo o item 'B'. Portanto, há a utilização do caso 3 para a subdivisão do nó. A execução segue o ramo da estrutura da árvore, de modo que a porção {A, D, F, L, H} é, então, comparada com o nó contendo os itens {A, D, F, G, H}. Assim como nos exemplos anteriormente descritos, é utilizado o segundo caso de subdivisão, permanecendo os itens correlatos {A, D, F} e as porções distintas entre si acabam por gerar dois nós filhos, a saber {G, H} e {L, H}. A Figura 3.13 apresenta a árvore ao final da inserção do referido nó.

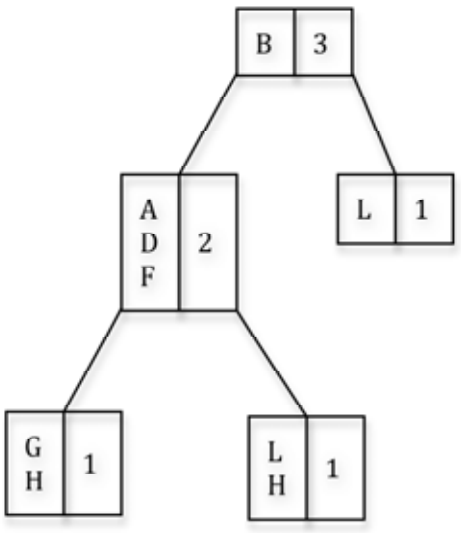

Figura 3.13 Situação da estrutura após inserção do terceiro nó

Assim, de modo análogo, o algoritmo de construção segue a execução para os demais registros, até que, ao final, obtém-se a árvore representante daquele conjunto de dados, que é a estrutura mostrada na Figura 3.9.

3.5.3 Estrutura *ItemMap*

A estrutura *ItemMap* (IM) foi proposta neste trabalho com o intuito de otimizar a representação dos itens relacionais na *Radix-tree*. Isso porque um item relacional, como antes mencionado, é identificado pelos campos ‘tabela’, ‘nome do atributo’ e ‘valor do atributo’. Devido ao tamanho que essa representação pode apresentar, sua utilização acarretaria a necessidade de substanciais espaços em memória para o seu armazenamento dentro dos nós da estrutura.

Tome-se, como exemplo, o item relacional “Internação.Hemorragia=Sim”. Considerando que cada caractere ocupa um byte em memória, o espaço necessário para o armazenamento desse item seria de 23 (vinte e três) bytes, desconsiderando-se o “.” e o “=”. Como o número de nós na estrutura geralmente é elevado, o espaço necessário somente para o armazenamento dos itens acabaria por despender uma substancial capacidade da memória disponível. A IM proposta foi desenvolvida com vistas a atenuar esse problema.

Desse modo, a estrutura efetua o mapeamento entre a representação do item relacional e a representação interna utilizada pela *Radix-tree*. A IM possui um campo que armazena a representação do item relacional e outro campo que registra o índice correspondente desse item na IL. A Figura 3.14 apresenta essa relação entre a IM e a IL.

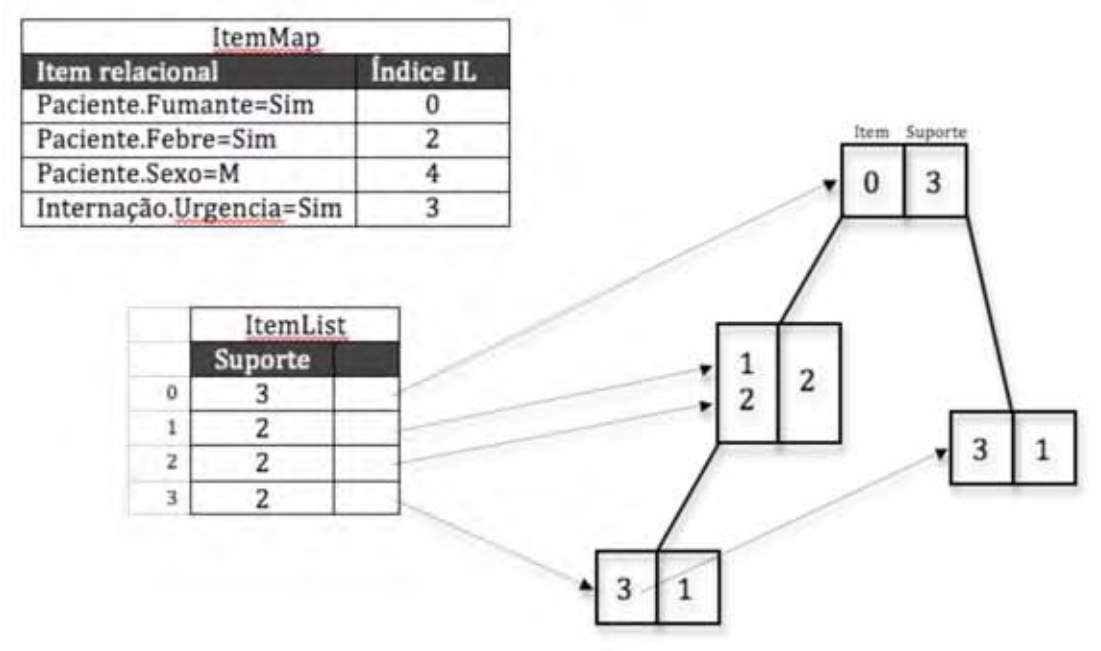

Figura 3.14 ItemMap e ItemList

Na IM, é armazenada a representação na forma de item relacional e também é ajustado o índice que tal item ocupa na IL. Por sua vez, a IL da presente proposta apresenta uma redução no número de campos, pois não é mais necessário o armazenamento do item nessa estrutura. Como afirmado anteriormente, cada item apresenta uma entrada na IM e, com isso, um valor de índice na IL. A partir dessa etapa, a identificação e representação do item é realizada por meio desse índice.

Desse modo, considerando que um índice é armazenado como um inteiro e ocupa 4 *bytes*, há um ganho substancial de espaço em memória. Após a mineração, os *itemsets* frequentes encontrados estarão representados pelos valores dos índices estabelecidos na IL. A obtenção dos padrões na forma de *itemsets* relacionais é imediata, novamente com o auxílio da IM.

3.5.4 Fusão de itens secundários

Outra proposta do presente trabalho foi a operação denominada ‘fusão de itens secundários’, que possibilita relacionar itens provenientes de diferentes tabelas sem a necessidade das custosas junções físicas. A operação consiste em reunir todos os itens frequentes das tabelas envolvidas no processo de mineração e que possuam um mesmo identificador TID. A Figura 3.15 apresenta uma base de dados cuja tabela primária é *Paciente* e a tabela secundária é a *Internação*. As linhas destacadas nas

tabelas apresentam o mesmo TID e, portanto, dizem respeito ao mesmo paciente. A operação de fusão associa os itens relacionais da tabela secundária *Internação* aos itens relacionais da tabela primária *Paciente*, realizando uma espécie de junção virtual. Assim, os itens relacionais "Internação.Hemorragia=Não", "Internação.Inconsciente=Sim" e "Internação.Inconsciente=Não" são, então reunidos juntamente com os itens relacionais da tabela *Paciente*.

Paciente

TID	Nome	Sexo
011	João	M
012	Maria	F
013	César	M
...	...	...

1 — n

Internação

ID Intern	TID	Hemorragia	Inconsciente
0201	011	Sim	Sim
0202	012	Não	Sim
0203	012	Não	Não
0204	013	Sim	Sim
...	...	...	...

Figura 3.15 Exemplo de "fusão de itens secundários"

Tais itens relacionais formam um conjunto referente ao paciente em questão e são inseridos como um nó na *Radix-tree*. Nesse caso, os padrões são contabilizados com referência à tabela primária *Paciente*, de modo que as múltiplas ocorrências dos itens relacionais provenientes das tabelas secundárias são desconsiderados, tal como ocorre com "Internação.Hemorragia=Não". Se o intuito da mineração for analisar a relação entre *Hemorragia* e *Inconsciente*, por exemplo, o foco da aplicação seria a tabela *Internação*, que seria considerada tabela primária e, nesse caso, a tabela *Paciente* assumiria o papel de tabela secundária.

3.5.5 Descrição do algoritmo MR-RADIX

O algoritmo MR-RADIX baseia-se na abordagem *pattern-growth* para efetuar a mineração multirrelacional de regras de associação. A estrutura de dados utilizada para a representação da base de dados em memória é a *Radix-tree*, cujo funcionamento e características já foram abordadas nas seções anteriores.

A execução da proposta do presente trabalho pode ser dividida em duas etapas: i) etapa de construção da *Radix-tree*; ii) etapa de mineração dos padrões frequentes. Essas etapas serão minudenciadas a seguir.

A primeira etapa, construção da *Radix-tree*, necessita de duas passagens completas pela base de dados a fim de obter seus itens frequentes para formular sua

representação na referida estrutura. Na Figura 3.16, tem-se o pseudo-código referente à construção da *Radix-tree*.

```
Algoritmo: Construção da Radix-tree
Entradas: Base de dados D; minsup
Saída: Radix-tree R, ItemList IL, ItemMap IM

1)  IM ← Ø
2)  para cada tabela T em D
3)      para cada item c em T
4)          se c ∈ C
5)              c.contador++;
6)          senão
7)              C ← {c | c.contador ← 1};

8)  IL ← {c | c.contador >= minsup};
9)  IM ← {c | c.contador >= minsup};

10) para cada tupla tp existente na tabela primária
11)     FREQ ← itens frequentes de tp
12)     para cada tabela secundaria TS
13)         para cada tupla ts em TS, tal que ts.tid  = tp.tid
14)             FREQ ← itens frequentes de ts

15) Crie um nó cujos itens sejam FREQ
16) Adicione o nó à R
```

Figura 3.16 Pseudo-código referente à etapa de construção da *Radix-tree*

A primeira parte desse pseudo-código – linhas 1 a 7 -, corresponde à primeira passagem pela base de dados e visa à contagem de todos os itens para cada uma das tabelas envolvidas, sejam primárias ou secundárias. Cada item encontrado é representado na forma de um item relacional e inserido em um conjunto que armazena os diversos itens e o número de suas respectivas ocorrências.

A segunda parte, por sua vez – linhas 8 e 9 -, consiste na construção das estruturas auxiliares *ItemList* (IL) e *ItemMap* (IM). Como já mencionado, a IL é utilizada durante todo o processamento da *Radix-tree*, indicando o valor de suporte e a lista encadeada de nós referente a cada item. A IL, assim, favorece o percurso na *Radix-tree*, tendo em vista que é possível obter todos os nós que compartilham um determinado item seguindo a respectiva lista encadeada.

Quando o algoritmo encontra um novo item na base de dados, é gerada, então, sua representação na forma de um item relacional. O algoritmo avalia o

conjunto de itens C que contenham os itens e suas respectivas ocorrências encontradas na base de dados, considerando para a construção da IL e IM apenas os itens frequentes, ou seja, cujo número de ocorrências seja igual ou maior ao valor do suporte mínimo *minsup*. Para cada um desses itens, é criada uma entrada na IM e na IL.

A terceira parte pseudo-código, descrita nas linhas 10 a 14, efetua a construção da *Radix-tree*, e corresponde à segunda leitura da base de dados, na qual são localizados os itens frequentes de cada registro. Tais passos correspondem à operação de "fusão de itens secundários", em que os itens relacionais das tabelas secundárias que compartilham um mesmo identificador (tid) são agrupados para inserção na árvore. Para cada registro da tabela primária, cria-se um novo nó contendo tais itens relacionais, o qual é acrescentado à árvore (linhas 15 e 16).

Já a exploração da *Radix-tree* para a busca dos *itemsets* frequentes, que consiste na segunda etapa do algoritmo, é realizada por uma estratégia *Top-down*, que percorre os nós iniciando do nó raiz em direção aos nós folhas. Essa estratégia apresenta a vantagem de não necessitar da construção de subestruturas temporárias durante o processamento dos padrões. Isso porque, ao processar um nó dos níveis mais elevados, gera-se todos os *itemsets* frequentes relacionados a esse nó. Dessa forma, ao processar nós de nível mais baixo, fica garantido que todos os nós ancestrais já foram processados.

Assim, pode-se efetuar uma pseudo-construção de subestruturas por meio de simples manipulação de ponteiros dos nós. Essa possibilidade torna o algoritmo muito mais eficiente do que aqueles que utilizam estratégia *Bottom-up*, nos quais há a necessidade de construção, de fato, das subestruturas.

A Figura 3.17 apresenta o pseudo-código para a busca e geração dos *itemsets* frequentes. Esse procedimento recebe a estrutura *Radix-tree* já previamente construída e com as ligações entre a mesma e a IL também efetuadas.

```
Algoritmo: Mineração da Radix-tree
Entradas: Radix-tree R, ItemList IL, minsup
Saída: Itemsets frequentes L

// Definições de variáveis
// X: Itemset representado por um vetor de itens
// h: índice atual em X
// ℓ: índice atual na IL

1)  X ← Ø;
2)  L ← Ø;
3)  ℓ ← 0; h ← 0;
4)  enquanto ( l < |IL| ) {
5)      se ( IL[ℓ].contador < minsup )
6)          ℓ ← ℓ + 1;
7)      senão
8)          se ( h > 0 && ℓ = X[h-1] ) {
9)              ℓ ← ℓ + 1;
10)             h ← h - 1;
11)         senão
12)             X[h] ← ℓ;
13)             h ← h + 1;
14)             L ← X;
15)             Ajusta IL[j].contador e IL[j].ptr, para j < ℓ
16)             ℓ ← 0;
```

Figura 3.17 Pseudo-código referente à etapa de mineração da *Radix-tree*

O algoritmo acima é uma versão iterativa para o percurso e extração de *itemsets* frequentes. A ideia por trás do seu funcionamento é a geração sucessiva de padrões que compartilham um mesmo prefixo, isto é, o algoritmo seleciona cada item da *ItemList* em ordem decrescente, de acordo com o valor do suporte e extrai todos os *itemsets* frequentes que se iniciam pelo item selecionado.

O algoritmo primeiramente inicializa as variáveis auxiliares, o conjunto de *itemsets* frequentes L e o vetor de itens X, conforme apresentado nas linhas 1 a 3. A sua execução mantém-se no laço principal (linhas 4 a 16) até que todas as linhas da IL (indicadas por ℓ) sejam processadas. O algoritmo passa a próxima linha da IL quando o contador de suporte do item atual for menor que o suporte mínimo previamente estabelecido (*minsup*), conforme descrito nas linhas 5 e 6.

A geração de *itemsets* frequentes cujo prefixo seja o item ℓ persiste até que seja satisfeita a condição de parada, indicadas nas linhas 7 a 10. O item atual ℓ torna-

se o prefixo do *itemset* X, que se transforma em um *itemset* frequente, sendo então atribuído ao conjunto de *itemsets* frequentes L (linhas 11 a 14). Nesse ponto, são ajustados os contadores e os ponteiros da IL, de forma a manipular a subárvore, que possui como nós folhas o item i (linhas 15 e 16). O laço principal é então executado para tal subárvore e assim sucessivamente.

A coleção de nós que compartilham um dado item I, os quais são conectados por meio da lista encadeada que parte da IL – denotada por *t-list*(I) – forma uma subárvore com seus ascendentes. De um modo geral, o processo de mineração consiste, primeiramente, na construção da subárvore cujos nós folhas sejam nós contendo o item I. Esse procedimento pode ser realizado eficientemente uma vez que IL mantém a *t-list*(I) que forma uma lista encadeando todos os nós que contém I.

Como já mencionado, o algoritmo utiliza a estratégia *top-down* e, portanto, a geração dessa subárvore pode ser efetuada com operações de manipulação de ponteiros. Ainda nessa etapa, é interessante salientar que a estrutura IL também é atualizada conforme o ajuste da subárvore, uma vez que os nós ancestrais já foram processados anteriormente, sendo possível a modificação de suas respectivas entradas na IL. Com isso, tem-se a geração dos *itemsets* relativos à subárvore.

Assim, conforme vai ocorrendo a geração, ocorre também a repetição dos procedimentos acima descritos, de modo que novas subárvores são construídas sucessivamente.

3.5.6 Estratégia de particionamento no algoritmo MR-RADIX

A fim de possibilitar a mineração de grandes bases de dados, foi implementada a estratégia de particionamento no algoritmo proposto. Tal estratégia consiste na subdivisão da base de dados em porções alocáveis e analisáveis em memória, o que viabiliza o processamento de grandes volumes de dados em etapas.

De início, é efetuada a primeira leitura da base de dados de modo similar ao funcionamento do MR-Radix apresentado anteriormente. Nessa fase, é obtida a contagem de todos os itens existentes na base de dados, bem como é realizada a construção das estruturas de dados auxiliares, *ItemList* e *ItemMap*. A primeira delas relaciona somente os itens frequentes, ou seja, aqueles cujo número de ocorrência seja maior ou igual ao valor absoluto do suporte mínimo previamente estabelecido. A

segunda, por sua vez – *ItemMap* - efetua o mapeamento do item relacional e a sua respectiva representação interna utilizada pelas demais estruturas, tal como a *Radix-tree*.

Em seguida, ocorre a etapa de construção da *Radix-tree*, na qual tem lugar a segunda leitura da base de dados. Nesse ponto, a base é lida até o momento em que o algoritmo identifique que não há mais disponibilidade de memória para a alocação da estrutura. Para descrever o funcionamento da estratégia de particionamento, é apresentado, na Figura 3.18, o pseudo-código.

Algoritmo: Particionamento de MR-Radix
Entradas: Base de Dados D, minsup
Saída: Itemsets frequentes L

// Definições de veriáveis
// L_{local}: *itemsets* frequentes locais
// C: conjunto de *itemsets* candidatos globais
// P_i, para i=1...k: partições da base de dados

1) L ← Ø; L_{local} ← Ø;
2) D = P_1 P_2 P_3 ... P_k , tal que P_i, para i=1...k, seja alocável em memória
3) Para i = 1 até k
4) Lê a partição P_i e constroi a *Radix-tree* local R_i;
5) L_i = MR-Radix(R_i);

6) L_{local} = $L_1 \cap L_2 \cap L_3 \cap ... \cap L_k$;
7) C = {c | c ∈ L_{local}, c.partições e c.suporte são as partições P_1...P_k onde c ocorre e o suporte acumulado, respectivamente }

8) Para cada c ∈ C
9) Para cada P_i, para i=1...k
10) Se $P_i \not\subset$ c.partições
11) c.partições ← P_i;
12) c.suporte ← c.suporte + nº de ocorrência de c em P_i;

13) L = {c | c.suporte >= minsup}

Figura 3.18 Pseudo-código referente à estratégia de particionamento

A porção lida da base de dados até o limite da quantia de memória disponível é então considerada como uma partição e tem sua mineração realizada individualmente. A base de dados é dividida em tantas partições quantas forem necessárias, de modo que cada partição P_i possa ser alocada em memória. Esse

procedimento se repete até que toda a base tenha sido devidamente processada (linhas 1 a 5).

Assim, os *itemsets* frequentes localizados nas partições são de caráter local e devem ser armazenados para posterior análise, uma vez que podem se caracterizar como sendo ou não padrões válidos para toda a base de dados. Esses *itemsets* locais frequentes provenientes das diversas partições são reunidos, formando um conjunto de *itemsets* frequentes L_{local}. Tais *itemsets* recebem campos indicando as partições já acessadas e o suporte acumulado, originando um conjunto de *itemsets* candidatos globais (linhas 6 e 7).

Uma terceira leitura da base de dados se faz necessária para a contagem dos valores de suporte, com o diferencial de, nesse momento, fazer referência a toda a base. Aqueles *itemsets* que atenderem ao suporte mínimo tornam-se, enfim, os padrões globais buscados (linhas 8 a 13).

Cabe salientar que a obtenção da estrutura IL global logo no início do algoritmo possibilitou a construção das estruturas da *Radix-tree* das partições de um modo mais otimizado. Isso porque foram considerados, localmente, apenas os itens que já se caracterizavam como itens frequentes globais.

O funcionamento da estratégia de particionamento do algoritmo MR-RADIX, ora descrito, pode ser sucintamente representado por meio do esquema exposto na Figura 3.19.

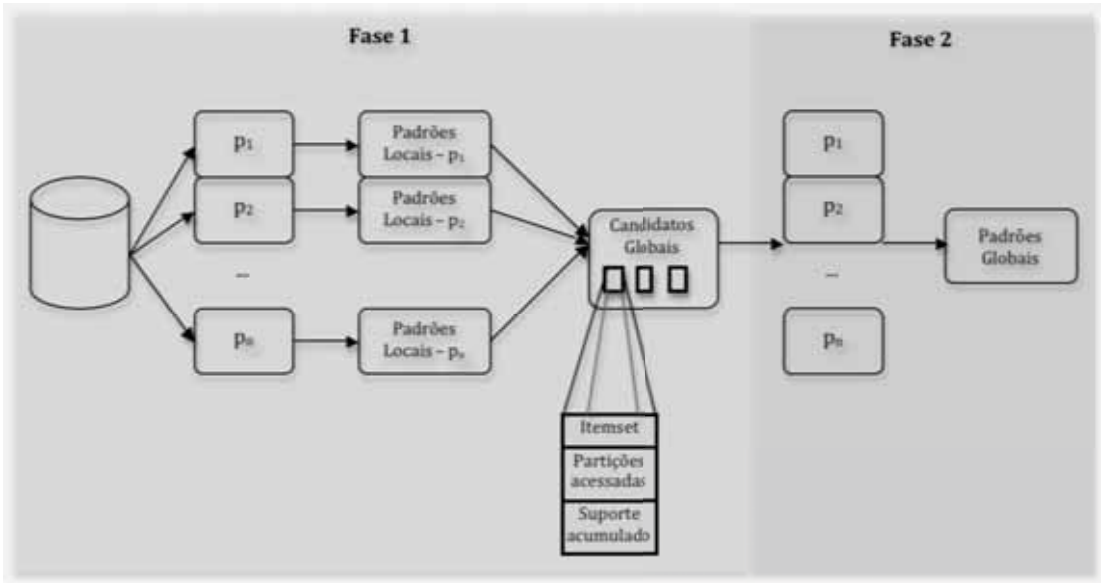

Figura 3.19 Esquema da estratégia de particionamento utilizado no MR-Radix

A primeira fase consiste na geração dos *itemsets* locais frequentes, obtidos pela mineração das partições individuais. Ainda nessa fase, esses *itemsets* locais são processados formando o conjunto de candidatos globais. Para otimizar a análise dos padrões desse conjunto, que ocorre na segunda fase, propôs-se uma representação diferenciada de tais padrões. Além do *itemset* propriamente dito, o padrão apresenta ainda dois campos: *partições acessadas* e *suporte acumulado*.

O primeiro campo armazena a relação de partições nas quais o padrão em questão era considerado localmente frequente. Por sua vez, o segundo campo representa a soma dos valores de suporte correspondente ao referido padrão nas partições em que o mesmo é frequente, ou seja, aquelas indicadas no campo *partições acessadas*.

Com o uso de tais campos, otimiza-se a segunda fase, uma vez que a contagem dos padrões não é necessária nas partições em que os mesmos já forem frequentes. Assim, o suporte de cada padrão deverá ser verificado apenas nas partições que não constarem na sua lista de *partições acessadas*. Ao concluir a contabilização do suporte nessas partições restantes, o padrão possuirá em seu campo *suporte acumulado* o valor de suporte global. Dessa forma, obtém-se os *itemsets* frequentes globais delimitando-se apenas os *itemsets* candidatos globais cujo valor de *suporte acumulado* seja superior ao suporte mínimo pré-estabelecido.

3.6 Considerações finais

Nesse capítulo, foi apresentado o algoritmo proposto neste trabalho, destacando-se os principais conceitos e as particularidades ligadas à extração de regras de associação multirrelacional.

O algoritmo deste trabalho utilizou-se da estrutura de *Radix-tree*, o qual apresenta boa capacidade de compressão de dados, resultando em uma economia de espaço de memória e melhoria no desempenho. Porém, foi necessária a proposta de algumas extensões para que o mesmo pudesse realizar a extração de padrões em bases de dados relacionais.

Assim, foram descritos o modelo de representação de *itemsets* relacionais bem como a estrutura auxiliar de mapeamento denominada *ItemMap*, a qual permitiu uma melhor representação de tais itens, de modo que a complexidade inerente aos

padrões multirrelacionais não se configure como um fator de influência negativa no que diz respeito ao uso otimizado de memória.

Para possibilitar a mineração de bases de dados relacionais, foi proposta uma técnica denominada “fusão de itens secundários”, que relaciona as múltiplas tabelas, suprimindo a necessidade das junções físicas. Tal técnica possibilita a fusão ou inserção dos itens frequentes das tabelas secundárias na estrutura *Radix-tree*, efetuando-se uma junção virtual para relacionar e produzir os padrões multirrelacionais.

Além disso, mostrou-se que foi desenvolvida uma estratégia de particionamento da base de dados, a qual ocorre para permitir que grandes bases sejam divididas em porções nas quais seja possível o seu processamento. Desse modo, quando não há mais espaço em memória, realiza-se a partição. Assim, os padrões obtidos das partições são combinados e testados em toda a base de forma otimizada, uma vez que somente as partições nas quais não foi contabilizada a frequência do padrão serão verificadas.

Capítulo 4
Testes e Resultados

4.1 Considerações iniciais

Neste capítulo são apresentados os testes efetuados com o algoritmo proposto, com o intuito de revelar o desempenho obtido pela proposta em termos de eficiência e escalabilidade. O estudo realizado compreende a análise do algoritmo MR-Radix comparando-o a algoritmos correlatos, a fim de verificar a contribuição do trabalho em termos práticos. Além disso, os testes visam à análise comparativa entre a abordagem tradicional e a abordagem multirrelacional, com o intuito de mensurar os benefícios trazidos pela utilização desta última em bases de dados relacionais.

4.2 Bases de dados e parâmetros de testes

Os testes consistiram na aplicação dos algoritmos em bases de dados relacionais e foram desenvolvidos com o intuito de obter medidas que descrevam o desempenho do algoritmo em relação ao estado da arte. Foram utilizadas como métricas: o tempo de execução e o volume de memória principal usado para o processamento dos padrões pelos algoritmos.

As bases de dados utilizadas caracterizam-se como bases relacionais contendo dados reais que foram alimentados por profissionais de seus respectivos contextos por meio de sistemas computacionais de cadastro ou similares. A Tabela 4.1

apresenta as informações descritivas – tabelas ou relações, número de registros e número de atributos – de tais bases de dados.

Tabela 4.1 Bases de dados utilizadas nos testes

Base de dados	Relação	Nº de registros	Nº de atributos
HC[6]	Amostra	13.100	31
	Form_Urologia_Rim	60	60
	Paciente	3.647	18
SIVAT[6]	Ficha	32.336	38
	Ficha_cid_10	64.873	3
	Ficha_parte_corpo	35.624	2
	Maquina_causadora	859	3
	Ocupação	1.237	3
CENSUS	Dados	2.458.359	68

A primeira base de dados, denominada HC (*Hospital do Câncer*), consiste em um conjunto de relações que armazenam informações relativas a um Banco de Tumores de um hospital do interior do estado de São Paulo. Dentre as informações armazenadas, pode-se destacar aquelas relacionadas aos pacientes e às suas respectivas amostras cancerígenas extraídas nesta instituição, bem como as informações clínicas mais detalhadas introduzidas, sob o formato de formulários, pelos profissionais de saúde. Tal base encontra-se sob a responsabilidade do Grupo de Banco de Dados do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista.

A segunda base de dados, SIVAT (*Sistema de Vigilância de Acidentes de Trabalho)*, cataloga registros de ocorrências de acidentes de trabalho de uma dada cidade do interior do estado de São Paulo. Os dados foram coletados e cadastrados pelo órgão municipal responsável pelo acompanhamento desse tipo de acidente. A base SIVAT, assim como a anterior, também é gerenciada pelo Grupo de Banco de Dados da referida instituição.

Por último, a base de dados CENSUS contém uma amostra do censo populacional realizado nos Estados Unidos no ano de 1990. Tal base de dados possui caráter público e encontra-se disponível no *UCI Machine Learning Repository*[7]. A

[6] A base HC possui 55 relações, enquanto que a base SIVAT possui 34 relações. Na Tabela 4.1 são apresentadas somente as relações que possuem informações relevantes para a mineração de dados.
[7] Disponível em http://archive.ics.uci.edu/ml/. Acesso em 27 de dezembro de 2009.

fim de facilitar a tarefa de mineração, a base de dados já se encontra pré-processada, com todos os dados discretizados e livres de valores nulos e imprecisos.

O equipamento utilizado para hospedar os algoritmos e executar os testes foi um microcomputador contendo um processador Intel Core2 Quad Q8200 (2,33 Ghz), 4 *gigabytes* de memória principal DDR2 e um disco rígido de 250 *gigabytes*, padrão SATA, taxa de transferência de 3,0 Gbit/s e 7200 rpm (rotações por minuto). Com relação aos *softwares* utilizados, o equipamento possui o sistema operacional Microsoft Windows XP Professional SP3, além do Kit de Desenvolvimento Java J2SE 1.6, do Ambiente de Desenvolvimento Netbeans 6.7.1 e do SGBD (*Sistema Gerenciador de Banco de Dados)* Mysql 5.1. Para fins de teste de utilização de memória, foi definido como parâmetro que a quantidade máxima de memória a ser utilizada pelos algoritmos é 64 MB (sessenta e quatro *megabytes*).

A base HC já se encontrava armazenada em um SGBD Mysql, portanto não foi necessária a efetivação de mudanças. Por outro lado, as bases SIVAT e CENSUS tiveram que ser transformadas ou migradas para o referido SGBD, uma vez que se encontravam armazenadas em outros formatos. A primeira utilizava o SGBD PostgreSQL para o seu armazenamento, enquanto que a segunda se encontrava no formato de *valores separados por vírgula* ou CSV[8]. Com isso, fez-se uso de um único SGBD para a manipulação dos dados, eliminando a possibilidade de geração de medições incoerentes devido à heterogeneidade dos meios de armazenamento.

Como já mencionado, a mineração de regras de associação pode ser dividida em duas etapas principais: (i) etapa de busca de *itemsets* frequentes, que são obtidos com base no valor do suporte mínimo previamente definido; (ii) etapa de geração das regras de associação, as quais são obtidas por meio dos *itemsets* frequentes localizados na etapa anterior, após os mesmos terem sido filtrados utilizando-se o valor da confiança mínima também pré-estabelecido.

De fato, a primeira etapa é definitivamente a parte do processo de mineração de regras de associação que demanda mais tempo computacional, sendo justamente o alvo de otimizações e proposições dos algoritmos. A etapa de geração das regras propriamente dita independe do algoritmo utilizado e, portanto, não será mensurada neste trabalho. Dessa forma, os algoritmos foram avaliados na tarefa de obtenção de

[8] CSV: do inglês, *Comma-separated values*. Um arquivo CSV armazena dados no formato tabular, sendo que os valores correspondentes às colunas ou campos são separados com o uso do caractere ‘,’ (vírgula).

todos os *itemsets* frequentes, sendo realizadas sucessivas execuções para diferentes valores de suporte mínimo, com o intuito de verificar o comportamento das propostas frente à variação dessa medida de interesse.

4.3 Estudo comparativo

Para fins de evidenciação dos resultados obtidos pelo algoritmo MR-RADIX, proposto no presente trabalho, esta seção expõe testes realizados em bases de dados reais de diferentes dimensões. O estudo consistiu na observação dos tempos de execução e das quantias de memórias utilizadas quando da aplicação dos algoritmos GFP-GROWTH, APRIORIMR, PATRICIAMINE em comparação ao MR-RADIX. Além dessas duas métricas principais, foram coletadas outras a fim de complementar os resultados, tais como número de nós na *Radix-tree* ou *Patricia-trie* e o tempo gasto com a efetivação das operações de junção.

No Apêndice C deste documento tem-se acesso aos valores das medidas de desempenho extraídas durante os diversos testes efetuados. Para maior confiabilidade dos valores obtidos, cada conjunto de teste teve suas métricas coletadas por 3 (três) vezes. Para a confecção dos gráficos e tabelas apresentadas nas próximas seções, efetuou-se o cálculo da média entre o resultado das três aplicações para cada caso de teste, reduzindo-se as imprecisões geradas por possíveis variações durante a coleta. O intuito foi o de garantir que os gráficos e demais recursos a serem apresentados contenham resultados mais confiáveis e representativos.

4.3.1 Estudo comparativo entre abordagem tradicional e multirrelacional

O algoritmo multirrelacional proposto neste trabalho, MR-RADIX, foi elaborado com base no algoritmo tradicional PATRICIAMINE. Com isso, faz-se relevante a comparação entre as duas propostas, possibilitando avaliar o impacto em termos de desempenho no uso de uma abordagem multirrelacional para mineração de bases de dados relacionais.

Cabe salientar que, diferentemente do MR-RADIX, que pode ser aplicado diretamente em múltiplas tabelas, o algoritmo tradicional PATRICIAMINE necessita de uma etapa de pré-processamento para a junção dessas tabelas. Com isso, o estudo

comparativo levará em conta o custo computacional necessário para a efetivação da operação de junção quando utilizado o algoritmo tradicional PATRICIAMINE. Na Figura 4.1 é apresentado esquematicamente o critério adotado neste teste comparativo.

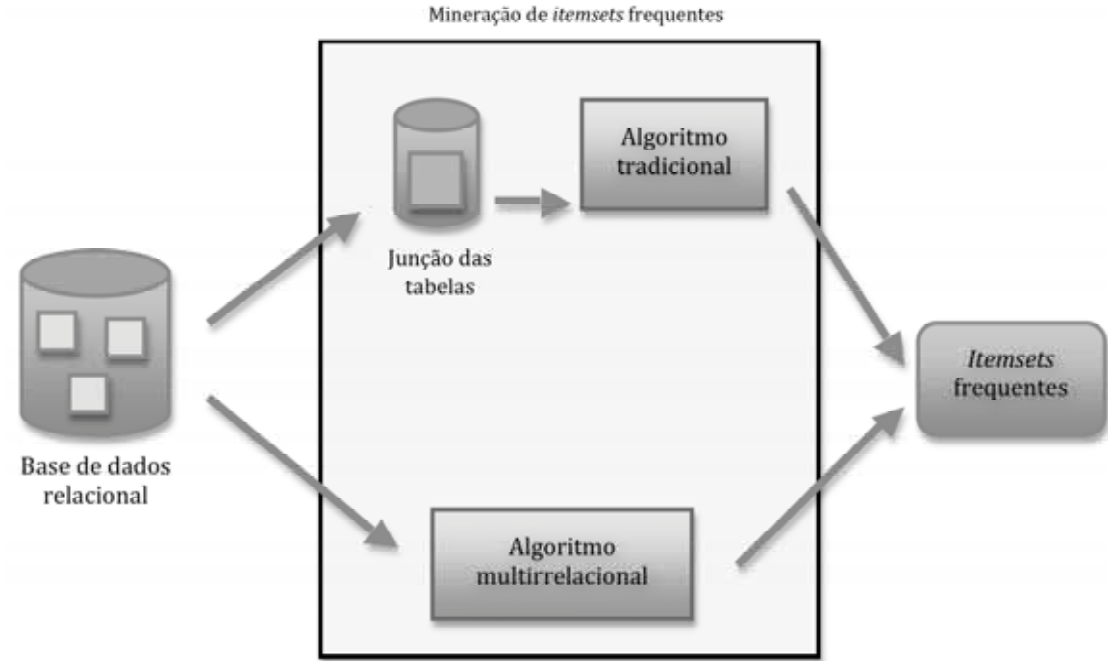

Figura 4.1 Esquema da medição para o estudo comparativo do MR-RADIX e PATRICIAMINE

Como mencionado, o desempenho do algoritmo PATRICIAMINE na mineração de *itemsets* frequentes contabiliza o custo da operação de junção das múltiplas tabelas. Para a realização dessa operação, utilizou-se um tipo de junção denominado *full outer join* para relacionar os dados de todas as tabelas envolvidas, uma vez que a relação resultante de uma junção desse tipo contém todos os registros das tabelas participantes da operação. Os valores não-coincidentes são referenciados com nulo ou *null*. Para ilustrar a operação desse tipo de junção, tome-se como exemplo as tabelas apresentadas na Figura 4.2:

Empregado

Nome	DepID
João	31
Maria	33
César	33
Hugo	34
Ana	34
Carlos	*Null*

Departamento

DepID	Nome
31	Vendas
33	Engenharia
34	Finanças
35	Marketing

Figura 4.2 Exemplo de juncão *full outer:* tabelas Empregado e Departamento

Como mencionado, a tabela resultante da aplicação dessa junção conterá todos os registros das tabelas envolvidas - no caso, *Empregado* e *Departamento*. Esse resultado pode ser visto na Tabela 4.2.

Observa-se que, nos registros em que não há correspondência entre as tabelas, os atributos não possuem valor definido. No exemplo, o empregado "Carlos" não está incluído em nenhum departamento, bem como o departamento de "Marketing" não possui nenhum empregado cadastrado; portanto, os respectivos campos são preenchidos com *null*.

Tabela 4.2 Tabela resultante da juncão *full outer*

Empregado.Nome	Empregado.DepID	Departamento.DepID	Departamento.Nome
João	31	31	Vendas
Maria	33	33	Engenharia
César	33	33	Engenharia
Hugo	34	34	Finanças
Ana	34	34	Finanças
Carlos	*null*	*null*	*null*
null	*null*	35	Marketing

Nas subseções seguintes serão apresentados e discutidos os resultados obtidos na análise comparativa entre o algoritmo multirrelacional proposto, MR-RADIX, e o algoritmo tradicional PATRICIAMINE, o qual também foi construído por este trabalho com base na versão original (PIETRACAPRINA; ZANDOLIN, 2003). O PATRICIAMINE proposto para os testes apresenta o funcionamento básico do algoritmo original, com a exceção de ter sido necessária a adaptação para o seu uso com bases de dados relacionais.

4.3.1.1 Comparativo do tempo de execução

A primeira análise realizada diz respeito ao tempo de execução dos algoritmos MR-RADIX e PATRICIAMINE visando evidenciar o desempenho na tarefa de mineração de regras de associação em bases de dados relacionais. Para isso, os respectivos tempos de execução foram medidos durante a aplicação dos algoritmos nas bases de dados HC e SIVAT.

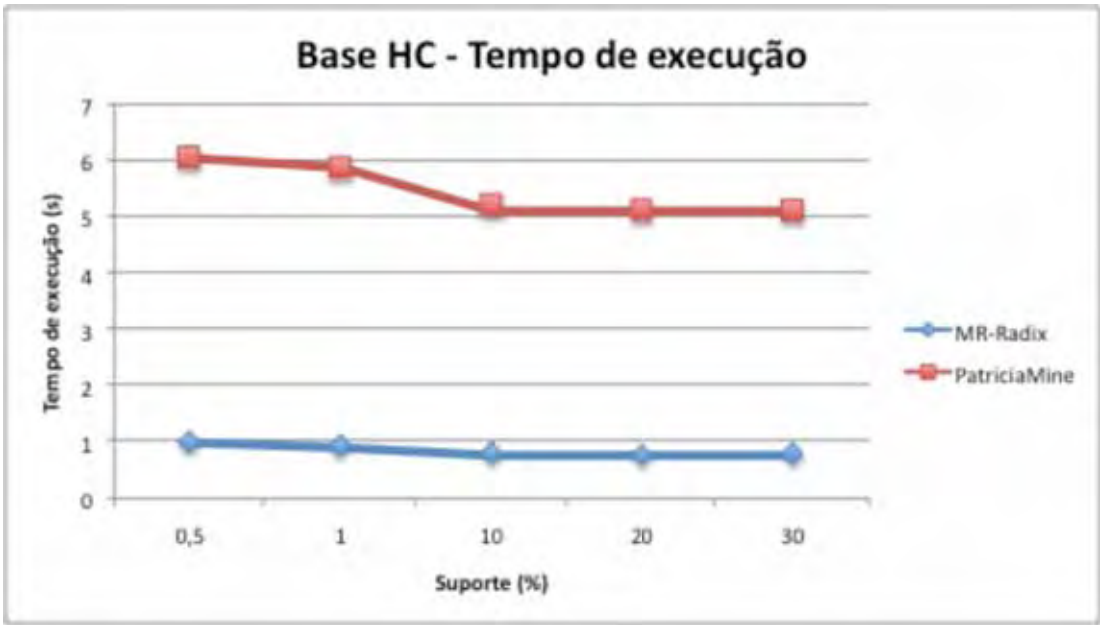

Figura 4.3 Gráfico do tempo de execução – base HC – MR-RADIX e PATRICIAMINE

Como se pode observar na Figura 4.3, a qual apresenta os valores dos tempos de execução na aplicação dos algoritmos na base HC, o desempenho proporcionado pelo algoritmo proposto, MR-RADIX, foi em torno de seis vezes mais rápido do que o seu correspondente tradicional, PATRICIAMINE. É interessante notar que o MR-RADIX não apresenta variações substanciais no tempo de execução, mesmo quando da utilização de menores valores de suporte – que tende a gerar um número maior de itens a serem processados. Em contrapartida, o PATRICIAMINE apresenta uma elevação no tempo de execução, à medida que se reduz o valor de suporte, indicando que o algoritmo é mais sensível ao aumento no número de itens a serem analisados na mineração.

O tempo de execução mais elevado do algoritmo PATRICIAMINE é explicado pelo fato da abordagem tradicional necessitar da realização de uma junção para reunir os dados provenientes de múltiplas tabelas. No gráfico da Figura 4.3, o tempo de execução referente ao PATRICIAMINE corresponde à soma dos tempos de junção e de mineração propriamente dita. Por sua vez, a Tabela 4.3 apresenta apenas os tempos relativos à operação de junção das tabelas, tanto para a base HC quanto para a base SIVAT.

Tabela 4.3 Tempos médios de junção do algoritmo PATRICIAMINE

Base de dados	Tempo de junção (s)
HC	3,70
SIVAT	1,95

Nota-se que o tempo médio gasto para a realização da junção das tabelas na base HC é de 3,70 (três virgula setenta) segundos. Observando-se o gráfico anterior, mostrado na Figura 4.3, para o valor de suporte igual a 0,5%, por exemplo, o tempo de execução total ficou em torno de 6 (seis) segundos. Com isso, conclui-se que a mineração da base HC demandou mais tempo na operação de junção das tabelas - 3,70 (três virgula setenta) segundos - do que de fato na execução do algoritmo PATRICIAMINE – cerca de 2,30 (dois vírgula trinta) segundos. Fato semelhante ocorre para todos os outros valores de suporte analisados.

Ainda que desconsiderando-se o tempo de junção, é possível observar que o algoritmo MR-RADIX foi mais de duas vezes mais eficiente em termos de tempo de execução do que o algoritmo PATRICIAMINE. Isso porque, enquanto este despendeu 2,30 (dois virgula trinta) segundos, aquele executou a mineração em um tempo próximo de 1 (um) segundo.

O segundo conjunto de testes para aferição dos tempos de execução utilizou a base SIVAT e os resultados obtidos encontram-se visualmente apresentados no gráfico da Figura 4.4.

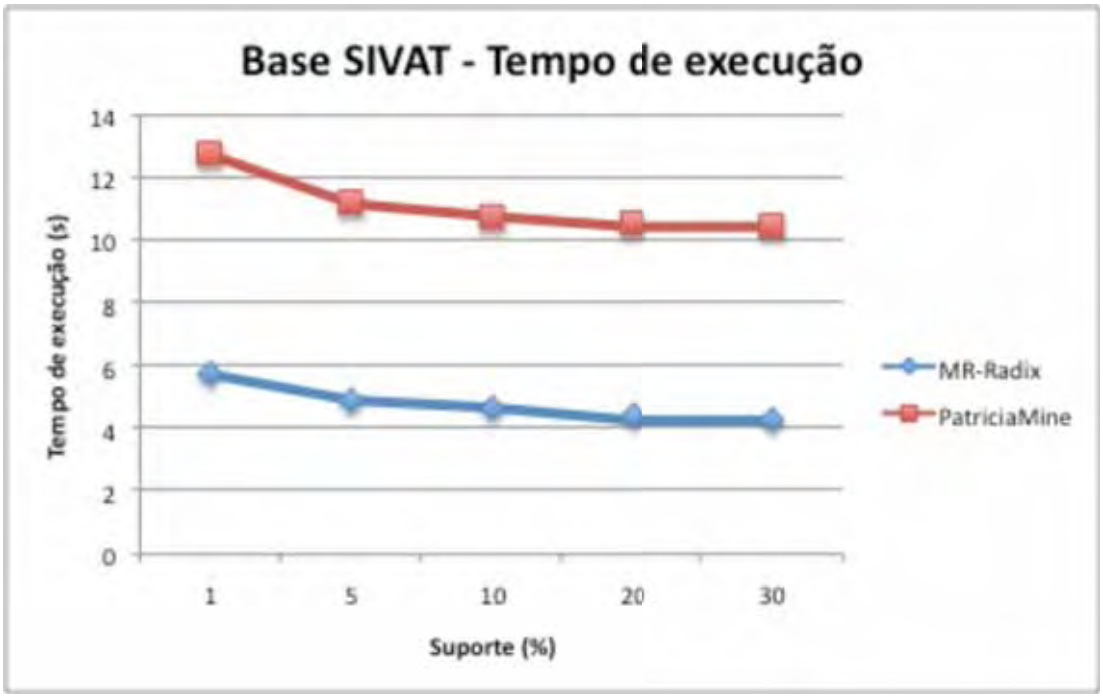

Figura 4.4 Gráfico do tempo de execução – base SIVAT – MR-RADIX e PATRICIAMINE

Com relação aos testes efetuados na base SIVAT, observa-se que, mais uma vez, o algoritmo MR-RADIX apresenta melhor desempenho frente ao PATRICIAMINE, sendo duas vezes mais rápido. Porém, o tempo de junção deixou de ser tão determinante no tempo total de execução, diferentemente do que ocorreu nos resultados anteriores com a base HC. Tome-se como exemplo o tempo de execução do PATRICIAMINE para o valor de suporte igual a 1%, que é aproximadamente 13 (treze) segundos. Observando a Tabela 4.3, verifica-se que o tempo referente à operação de junção é de 1,95 (um virgula noventa e cinco) segundos, o que permite dizer que, para esse valor de suporte, cerca de 11 (onze) segundos foram despendidos com o processamento dos padrões propriamente dito. Isso se deve ao fato de que a base SIVAT possui um volume maior de dados quando comparada à base HC e, por consequência, gera um número mais elevado de itens frequentes. Com isso, a estrutura de dados utilizada pelo algoritmo torna-se mais robusta, necessitando de mais tempo para a conclusão do processamento.

Com os resultados obtidos nesse primeiro instante, nota-se que o algoritmo MR-RADIX, apesar de lidar com padrões multirrelacionais mais complexos, apresenta maior eficiência do que seu correspondente tradicional PATRICIAMINE quando aplicado em ambas as bases de dados. Isso indica que a mineração multirrelacional mostrou-se como o meio mais vantajoso para a mineração das bases de dados

relacionais, uma vez que pode ser aplicada diretamente nas múltiplas tabelas evitando, assim, as custosas operações de junção.

4.3.1.2 Comparativo de memória utilizada

O estudo da utilização de memória objetiva investigar se a manipulação de padrões multirrelacionais acarreta a necessidade de uma quantidade maior desse recurso. Para tal, os algoritmos MR-RADIX e PATRICIAMINE foram aplicados novamente nas bases HC e SIVAT em diferentes cenários de suporte mínimo.

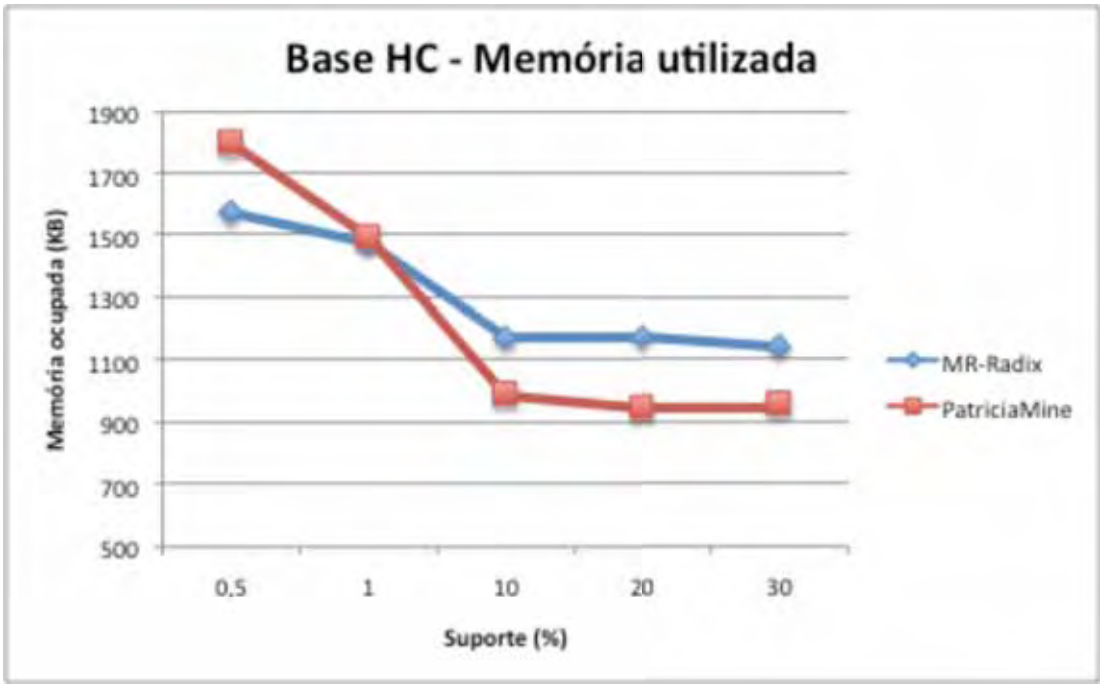

Figura 4.5 Gráfico do espaço de memória utilizado – base HC – MR-RADIX e PATRICIAMINE

Na Figura 4.5 observa-se os resultados das medições para a base de dados HC, sendo que o uso da memória está expresso em *kilobytes* (KB). Para valores maiores de suporte, nota-se que o algoritmo MR-RADIX apresenta um consumo de memória mais elevado em comparação ao PATRICIAMINE. Porém, para valores mais baixos de suporte – neste caso, inferiores a 1% - o cenário se altera e o algoritmo MR-RADIX passa a apresentar menos uso da memória principal.

Esse fato está estritamente relacionado ao número de nós gerados na estrutura de representação da base de dados – *Radix-tree*, no caso da MR-RADIX e *Patricia-trie*, no caso do PATRICIAMINE - durante o processo de mineração. Como pode ser

observado na Tabela 4.4, o número de nós gerados pelo algoritmo MR-RADIX é superior ao número gerado pelo PATRICIAMINE para valores de suporte superiores a 10%.

Tabela 4.4 Número de nós na árvore na base HC

Algoritmos	Suporte				
	0,5%	1%	10%	20%	30%
MR-RADIX	2999	2586	500	281	216
PATRICIAMINE	5132	3754	425	258	137

Em contrapartida, a abordagem tradicional origina um número maior de nós para valores menores de suporte. Isso se deve ao fato de a junção efetuada nas múltiplas tabelas acabar gerando um conjunto de dados maior. Assim, para valores de suporte menores, um número maior de itens atendem ao suporte mínimo e tornam-se frequentes e, por tal motivo, são inseridos como nós da estruturas. Com isso, o consumo de memória se torna mais elevado nessas condições, tal como inicialmente apresentado pelo gráfico da Figura 4.5.

O segundo teste efetuado para mensurar o consumo de memória dos algoritmos supracitados consistiu na mineração da base SIVAT, cujo gráfico resultante é exposto na Figura 4.6.

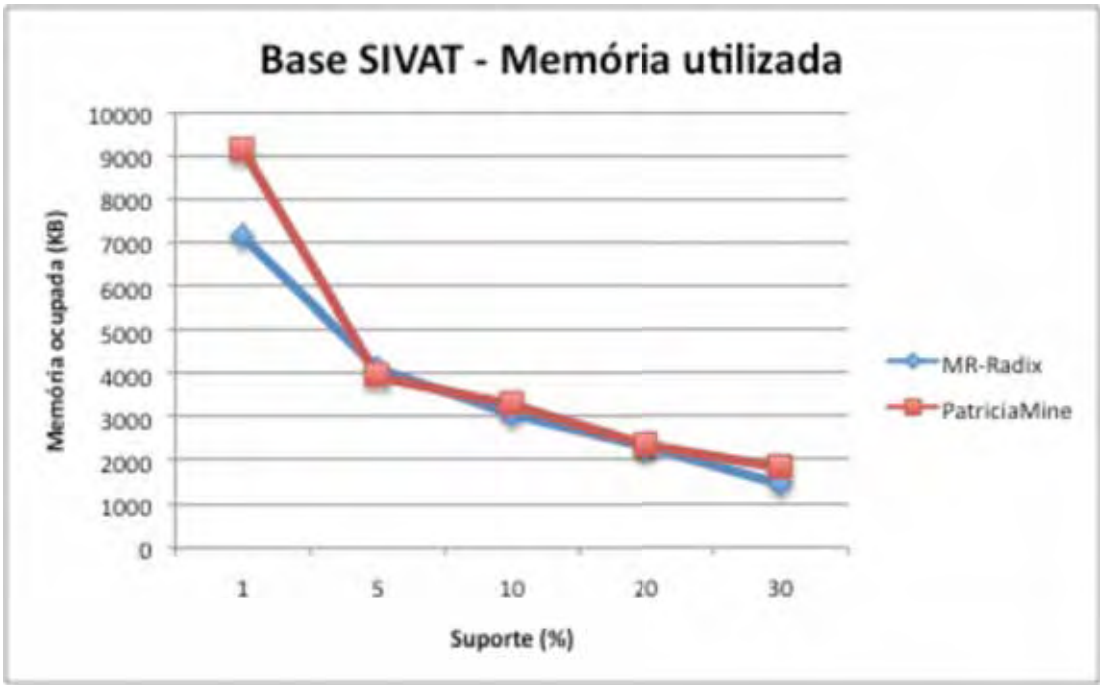

Figura 4.6 Gráfico do espaço de memória utilizado – base SIVAT – MR-RADIX e PATRICIAMINE

Como pode ser observado na Figura 4.6, o consumo de memória foi semelhante em ambos os algoritmos para a maioria dos valores de suporte. No entanto, assim como ocorreu na base HC, para menores valores de suporte – nesse caso 1% - o algoritmo PATRICIAMINE necessita de uma quantidade maior de memória. Esse comportamento está relacionado ao número de nós gerados pelos respectivos algoritmos nas diferentes configurações de suporte, como pode ser observado na Tabela 4.5.

Tabela 4.5 Número de nós na árvore na base SIVAT

Algoritmos	Suporte				
	1%	5%	10%	20%	30%
MR-RADIX	40426	24272	16266	10473	5580
PATRICIAMINE	58861	21670	14545	9287	5865

Nota-se que a diferença entre o número de nós gerados pelos algoritmos para suporte igual a 1% encontra-se em torno de 18 mil nós. Essa quantidade significativa de nós gerados devido a necessidade da junção das tabelas acaba por intensificar a necessidade de espaço em memória.

Assim, para valores maiores de suporte, o algoritmo PATRICIAMINE apresenta um consumo inferior de memória em relação ao MR-RADIX. Isso decorre do fato de o algoritmo tradicional lidar com padrões teoricamente mais simples, visto que são provenientes de um única tabela. Porém, para valores menores de suporte, o cenário se altera e o algoritmo PATRICIAMINE passa a apresentar um consumo mais elevado. Como já mencionado, esse comportamento se deve ao fato de as junções necessárias nesta abordagem acarretarem a geração de um número maior de itens frequentes e, consequentemente, um número maior de nós na estrutura *Patricia-trie*.

Essa característica deve ser levada em conta relacionando-a com os resultados relativos ao tempo de execução dos algoritmos. Ainda que o algoritmo MR-RADIX apresente um consumo um pouco mais elevado de memória para determinados valores de suporte, devido à manipulação de padrões multirrelacionais, o seu tempo de execução mantém-se substancialmente inferior em relação ao PATRICIAMINE.

Analisando ainda os gráficos do uso de memória, apresentados na Figura 4.5 e na Figura 4.6, nota-se que o algoritmo PATRICIAMINE apresenta uma curva de crescimento mais vertical, indicando que a necessidade de espaço de memória intensifica-se mais rapidamente com a redução dos valores de suporte. Por sua vez, o algoritmo MR-RADIX apresenta uma curva mais conservadora, a qual denota uma característica interessante, pois o aumento na demanda de itens frequentes não acarreta o crescimento significativo no uso de memória.

4.3.2 Estudo comparativo entre algoritmos multirrelacionais

Nesta seção, serão apresentados os resultados relativos à análise comparativa dos algoritmos multirrelacionais MR-RADIX, APRIORIMR e GFP-GROWTH. O algoritmo APRIORIMR estende o algoritmo APRIORI original para possibilitar a mineração em múltiplas tabelas de uma base de dados e, cabe acrescentar, foi implementado na ferramenta fDMMR em um trabalho anterior (OYAMA, 2006). Este algoritmo foi escolhido para possibilitar a comparação do MR-RADIX com um algoritmo representante da abordagem CGT (*Candidate Generate-and-Test)* .

O algoritmo GFP-GROWTH, por sua vez, possibilita a mineração de padrões multirrelacionais baseando-se no algoritmo tradicional FP-GROWTH e se caracteriza como uma proposta no contexto de mineração multirrelacional de regras de

associação. Com base no algoritmo original do GFP-GROWTH (PIZZI, 2006) e para possibilitar a confecção de testes, este trabalho também construiu e desenvolveu esse algoritmo acrescentando uma versão do mesmo à ferramenta *fDMMR*. Para as etapas de construção e mineração da *FP-tree*, reutilizou-se parte dos códigos-fonte do algoritmo FP-GROWTH[9], implementado por Frans Coenen.

Nas subseções seguintes serão apresentados e discutidos os resultados obtidos na análise comparativa entre os algoritmos mencionados.

4.3.2.1 Comparativo do tempo de execução

A primeira análise a ser discutida refere-se à aplicação dos algoritmos multirrelacionais nas bases de testes a fim de coletar os respectivos tempos de execução. Considere o gráfico na Figura 4.7, o qual apresenta os valores de tempos de execução em escala logarítmica referente aos testes realizados na base de dados HC.

Os tempos de execução para essa base de dados não puderam ser coletados para o algoritmo GFP-GROWTH, uma vez que, para todos os valores de suporte utilizados, houve o problema de falta de memória principal. Faz-se o adendo de que, para verificar a quantidade de memória necessária, efetuou-se um teste específico, ampliando a quantia de memória disponível na máquina. Nos resultados, foi obtido que, para o valor de suporte igual a 30%, o algoritmo GFP-GROWTH necessitaria de mais de 300 MB (trezentos megabytes) de memória. Essa característica observada se deve ao fato de que o referido algoritmo, por ser baseado no FP-GROWTH, apresenta baixo desempenho quando utilizado em bases de dados esparsas, tal como a HC. Com isso, observou-se que a *FP-tree* gerada nos testes apresentou muitos nós e ramos, o que torna oneroso o processamento dos padrões frequentes, além de exigir considerável espaço de memória para sua representação.

[9] Disponível em http://www.csc.liv.ac.uk/~frans/KDD/Software/FPgrowth/fpGrowth.html. Acesso em 26 de Dezembro de 2009.

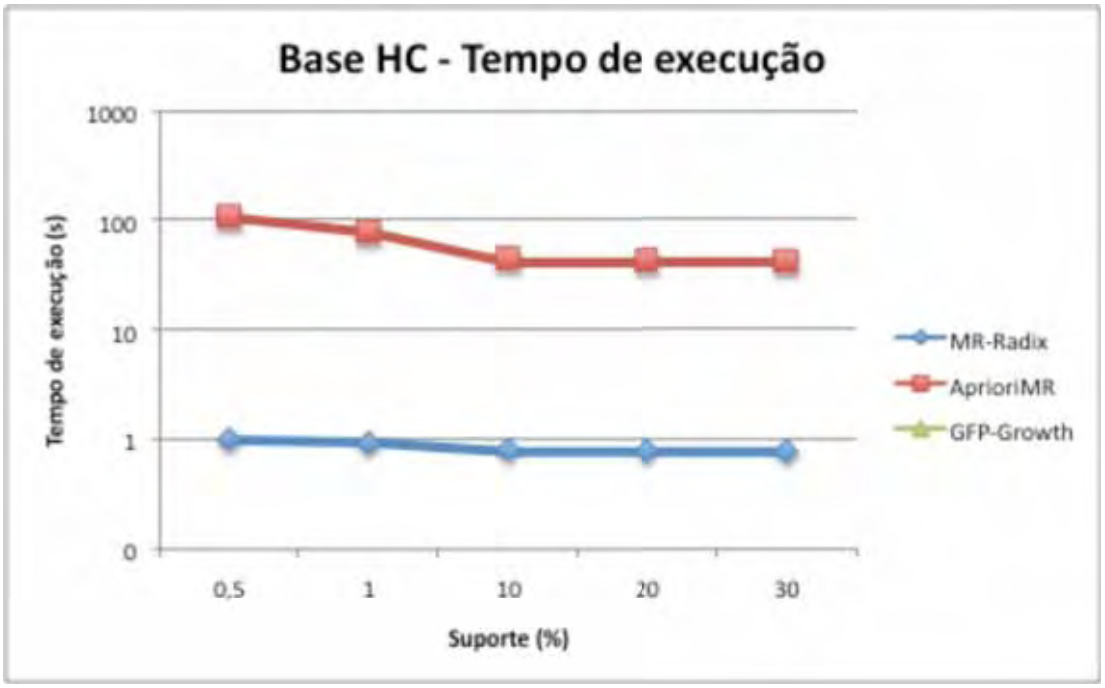

Figura 4.7 Gráfico do tempo de execução dos algoritmos multirrelacionais – base HC

O gráfico evidencia a maior eficiência apresentada pelo algoritmo MR-RADIX em comparação ao APRIORIMR. O algoritmo proposto chega a ser duas ordens de magnitude mais rápido que o algoritmo baseado no APRIORI. O desempenho inferior do APRIORIMR pode ser explicado pelo fato de que este se baseia na abordagem CGT, o que origina, durante as etapas de processamento, um número elevado de padrões a serem analisados. Cabe salientar também que o algoritmo efetua vários acessos a disco, dependendo da quantidade de etapas necessárias para a produção dos padrões frequentes. Em contrapartida, o MR-RADIX efetua apenas dois desses acessos, independentemente da quantidade de padrões a serem gerados.

Ademais, nota-se que o algoritmo MR-RADIX apresenta pouca variação do tempo de execução em relação aos diferentes valores de suporte, o que pode ser visto como uma característica benéfica, uma vez que o mesmo apresenta boa capacidade de lidar com o crescimento do número de itens frequentes. O mesmo comportamento, no entanto, não se observa no algoritmo APRIORIMR, o qual apresenta um aumento no tempo necessário para efetivar o processamento à medida que se reduz o valor de suporte. Essa característica não é própria somente do APRIORIMR, mas ocorre em outros algoritmos derivados do APRIORI (GYORODI; GYORODI; HOLBAN, 2004).

No segundo teste efetuado para a medição do tempo de execução foi utilizada a base SIVAT. Tal base de dados, como dantes relatado, apresenta um número maior de registros e, consequentemente, exige maior processamento para a obtenção dos *itemsets* frequentes. O gráfico da figura abaixo apresenta o resultado do teste comparativo, também em escala logarítmica, para o tempo de execução da base supra-mencionada.

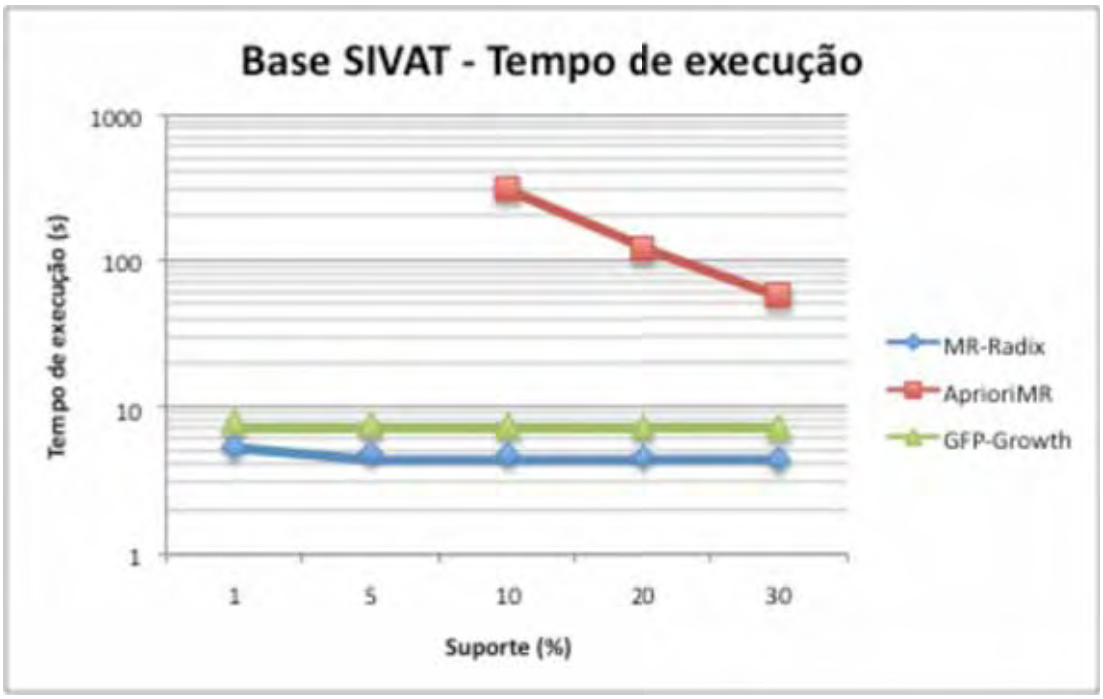

Figura 4.8 Gráfico do tempo de execução dos algoritmos multirrelacionais – base SIVAT

De modo análogo aos resultados obtidos no teste na base de dados anterior, é possível observar uma significativa diferença no desempenho do algoritmo MR-RADIX em relação ao APRIORIMR. Novamente, o algoritmo proposto apresentou baixa variação no tempo de execução em relação aos diferentes valores de suporte. Por exemplo, para o suporte igual a 5%, o algoritmo apresentou o tempo de execução próximo a 4,8 (quatro virgula oito) segundos. Reduzindo-se o valor do suporte para 1%, nota-se que o tempo de execução se eleva para cerca de 5,3 (cinco virgula três) segundos.

Tal diferença no tempo de execução é relativamente pequena se for considerado que a redução do suporte de 5% para 1% acarreta o aumento de quase 10 mil nós na *Radix-tree*, tal como pode ser observado na Tabela 4.6. Isso reforça os resultados positivos obtidos pelo algoritmo MR-RADIX, no que diz respeito à

capacidade de absorção de novos nós, de modo que o aumento na árvore não gera reduções significativas no desempenho.

Tabela 4.6 Número de nós na árvore *Radix-tree* - base SIVAT

Suporte	Número de nós
1%	31874
5%	22035
10%	18885
20%	14591
30%	8988

O algoritmo APRIORIMR, por sua vez, apresenta uma curva de crescimento do tempo de execução muito insatisfatória, denotando que a eficiência se deteriora à medida que o algoritmo é submetido a valores mais baixos de suporte. Cabe fazer o adendo de que, para valores de suportes inferiores a 10%, o APRIORIMR não foi capaz de realizar o processamento dos padrões frequentes devido à ocorrência de falta de espaço de memória suficiente para a conclusão do processo. Por esse motivo, não há informações dos tempos de execução desse algoritmo para tais valores de suporte.

O algoritmo GFP-GROWTH, por sua vez, apresentou desempenho relativamente semelhante ao apresentado pelo MR-RADIX, uma vez que mostrou tempos ligeiramente maiores. Para a maioria dos valores, os dois algoritmos mantiveram seus tempos de execução praticamente constantes. No entanto, cabe salientar que, ao reduzir o valor de suporte de 5% para 1%, ambos obtiveram um aumento no tempo de execução. À primeira vista, o gráfico parece indicar que o algoritmo MR-RADIX apresentou um queda maior de desempenho. Porém, o aumento sofrido pelos algoritmos foi relativamente similar. MR-RADIX passou de 4,8 (quatro virgula oito) segundos para 5,3 (cinco virgula três) segundos, ao passo que GFP-GROWTH passou de 7,7 (sete virgula sete) segundos para 8,0 (oito) segundos. Ou seja, o desempenho da proposta do presente trabalho se configurou como sendo o mais eficiente em relação ao tempo de execução para a base testada.

4.3.2.2 Comparativo de memória utilizada

A análise do uso de memória dos algoritmos multirrelacionais envolvidos iniciou-se com a aplicação na base de dados HC, cujos resultados se encontram apresentados em escala logarítmica na Figura 4.9.

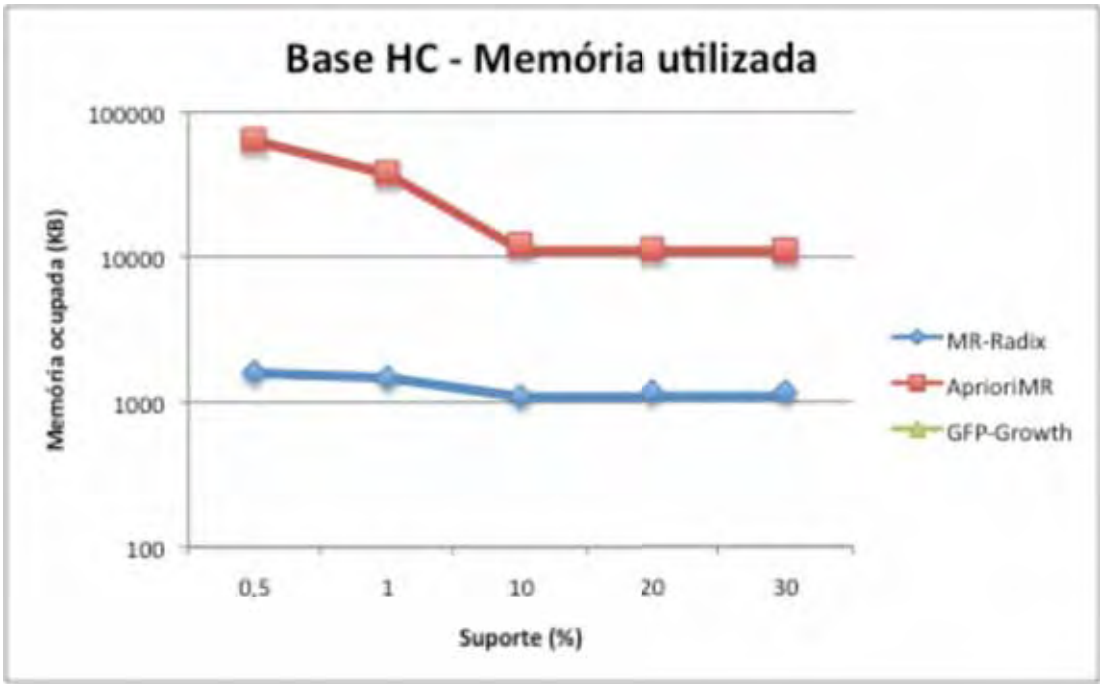

Figura 4.9 Gráfico do espaço de memória utilizado - base HC – algoritmos multirrelacionais

O algoritmo MR-RADIX, com a utilização de sua estrutura de dados *Radix-tree*, consegue comprimir substancialmente a base de dados. Por outro lado, o algoritmo APRIORIMR, mesmo quando aplicado em bases de dados de média dimensão – tal como a HC -, produz um número elevado de *itemsets* candidatos, o que gera a demanda de grande espaço de memória necessário para o armazenamento e manipulação dos padrões em processamento.

No caso extremo da presente análise, com o suporte igual a 0,5%, verifica-se que o algoritmo proposto por este trabalho ocupa cerca de 1.600 (mil e seiscentos) KB enquanto que o APRIORIMR ocupa mais de 63.000 (sessenta e três mil) KB.

Em relação ao algoritmo GFP-GROWTH, novamente, seus valores de memória utilizada não puderam ser coletados, pois o algoritmo não concluiu o processamento, uma vez que a memória disponível e limitada no início dos testes não foi suficiente para a efetivação da mineração em todos os valores de suporte analisados.

Em relação ao teste de memória utilizada aplicado na base SIVAT, exposto em gráfico de escala logarítmica na Figura 4.10, é possível notar que o consumo de

memória do algoritmo MR-RADIX também se mantém muito inferior em relação ao do APRIORIMR.

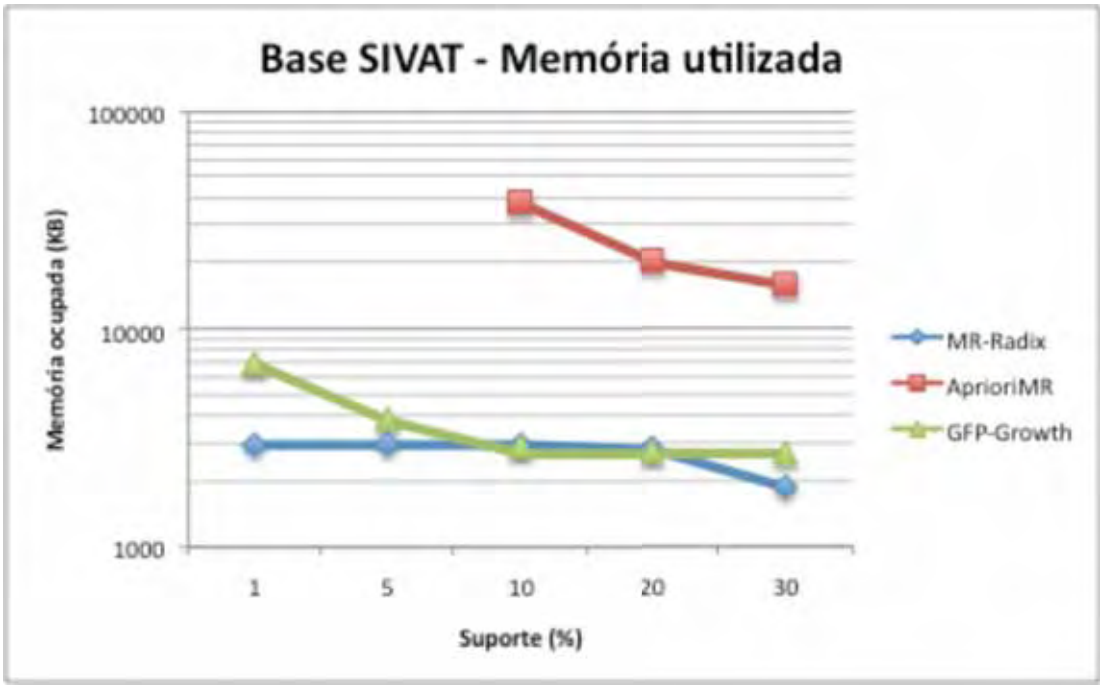

Figura 4.10 Gráfico do espaço de memória utilizado - base SIVAT – algoritmos multirrelacionais

Um ponto relevante de ser destacado é que, para o MR-RADIX, o uso de memória apresenta um aumento somente quando da redução do valor de suporte de 30% para 20%. Para valores de suporte menores que 20%, a quantidade de memória mantém-se praticamente constante ate o último valor de suporte analisado. Em contrapartida, a quantidade de memória exigida pelo algoritmo APRIORIMR aumenta de forma significativa, à medida que o valor de suporte vai sendo reduzido. Ressalta-se que, para valores de suporte inferior a 10%, não há medições do uso de memória por esse algoritmo, visto que a quantidade necessária excedeu ao limite disponível e fixado na máquina de teste (64 MB).

Em comparação com o algoritmo GFP-GROWTH, nota-se que o algoritmo MR-RADIX apresenta uma otimização no uso de memória. Para os valores de 10% e 20%, ambos utilizaram praticamente a mesma quantidade de memória. No entanto, para os demais valores de suporte, o MR-RADIX necessitou de menos memória para a conclusão da mineração. Um observação a ser feita é que o formato da curva de utilização de memória para os dois algoritmos no gráfico denota uma significativa diferença.

O algoritmo MR-RADIX apresenta uma curva horizontal que retrata uma característica positiva, uma vez que a quantidade de memória utilizada pelo algoritmo mantém-se constante, apesar da variação dos valores de suporte. Por outro lado, o algoritmo GFP-GROWTH apresenta uma curva vertical, indicando que o uso de memória cresce inversamente proporcional ao valor de suporte. Especificamente para o valor de suporte igual a 1%, nota-se a que algoritmo MR-RADIX mantém o consumo de memória próximo de 3.000 KB (três mil kilobytes) enquanto que o algoritmo GFP-GROWTH apresenta o crescimento no consumo, passando de 3.800 KB (três mil e oitocentos kilobytes) para cerca de 7.000 KB (sete mil kilobytes).

4.3.2.3 Análise de escalabilidade

Nesta subseção, será avaliado o desempenho dos algoritmos multirrelacionais em situações que envolvam a mineração de uma extensa base de dados, na qual o processamento frequentemente necessita de um espaço em memória principal superior ao disponível no equipamento, exigindo a adoção de estratégias, tal como particionamento, que permitam a concretização da mineração.

Nos testes a seguir expostos, será considerado que a memória principal estará limitada a 42 MB (quarenta e dois *megabytes*). O intuito da medição é o de verificar até quais valores de suporte os algoritmos multirrelacionais possuem a capacidade de efetuar a mineração, considerando-se essa limitação de memória.

Os algoritmos multirrelacionais utilizados nesta etapa serão os mesmos apresentados nos testes anteriores, a saber APRIORIMR, MR-RADIX e GFP-GROWTH. Para este teste, foi utilizada a notação MR-RADIX-PARTITION para representar o algoritmo proposto com a estratégia de particionamento ativa. A notação MR-RADIX, por sua vez, foi utilizada para representar a proposta sem a estratégia de particionamento ativada. Tal distinção foi feita para fins de diferenciação na leitura e análise dos testes e gráficos, bem como para comprovar a eficácia da adoção da estratégia de particionamento para a mineração de grandes bases de dados.

Cabe salientar que, para efetivar a aplicação do teste, foram utilizados diversos valores de suporte mínimo, variando-os desde 10% até 90%, o que possibilitou observar o comportamento dos algoritmos analisados por toda a faixa de valores de suporte.

Os resultados obtidos neste estudo se encontram reunidos na Figura 4.11, disposta abaixo, a qual expõe o tempo de execução dos algoritmos mencionados em relação à base CENSUS.

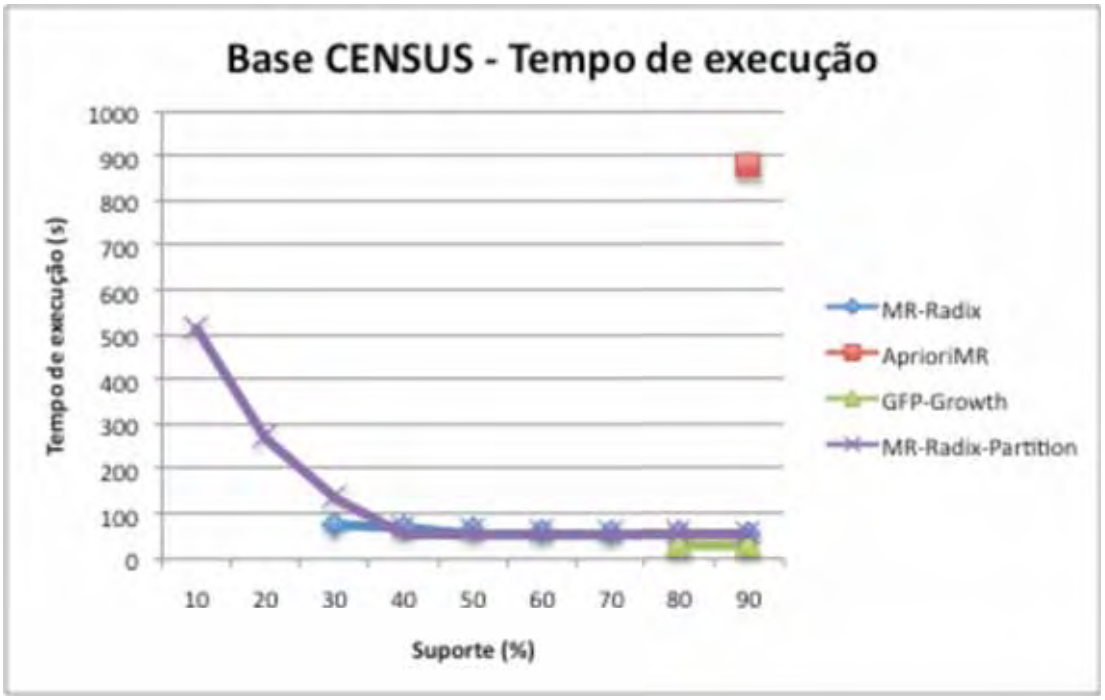

Figura 4.11 Gráfico do tempo de execução - base CENSUS

Como pode ser observado, o algoritmo MR-RADIX-PARTITION foi o único a concluir o processamento dos padrões frequentes para todos os valores de suporte aplicados. Os demais algoritmos sofreram, em algum momento, o problema de falta de memória para a conclusão da mineração. O algoritmo MR-RADIX, por não implementar a estratégia de particionamento, não teve seu tempo medido para valores de suporte inferior a 30%. As medidas de desempenho referentes ao GFP-GROWTH não puderam ser coletadas para valores inferiores a 80%. Por sua vez, o APRIORIMR concluiu o processamento em apenas um caso de teste, com suporte igual a 90%. Além disso, este último foi o algoritmo que mais tempo despendeu para a execução da tarefa de mineração.

O desempenho do MR-RADIX-PARTITION e do MR-RADIX permanecem praticamente iguais até o valor de suporte de 40%. Isso se deve ao fato de que o algoritmo MR-RADIX-PARTITION, nos valores mais altos de suporte, comporta-se de modo similar ao MR-RADIX, dado que a estratégia de particionamento não é acionada nessas situações. Tal comportamento ocorre porque a versão com a

estratégia de particionamento é capaz de identificar que há memória suficiente para o processamento de toda a base de dados e, portanto, não efetua partições.

O algoritmo GFP-GROWTH só teve seu desempenho mensurado para os dois maiores valores de suporte, 80% e 90%, uma vez que, como foi dito, houve a ocorrência de falta de memória principal. Nesses casos, seu tempo de execução foi ligeiramente inferior ao dos algoritmos MR-RADIX e MR-RADIX-PARTITION.

As divisões da base de dados CENSUS em partições foram necessárias para os valores de suporte inferiores a 30%, devido à insuficiência de memória para representar toda a base de dados. Nesses casos, tal como mostrado na Figura 4.11, somente o algoritmo MR-RADIX-PARTITION obteve êxito na execução da conclusão. Por exemplo, para suporte mínimo igual a 10%, o algoritmo subdividiu a base em 2 (duas) partições com tamanho suficiente para que sua respectiva representação pudesse ser alocada na memória principal. Para cada uma das partições, o algoritmo então efetuou a mineração de *itemsets* frequentes locais e, ao final, reuniu tais *itemsets* com o intuito de obter o conjunto de padrões frequentes globais.

O segundo passo da análise de escalabilidade consistiu no estudo do uso de memória pelos algoritmos supra-mencionados, quando aplicados na base CENSUS. Tais resultados podem ser observados na Figura 4.12, a qual expõe graficamente as métricas coletadas.

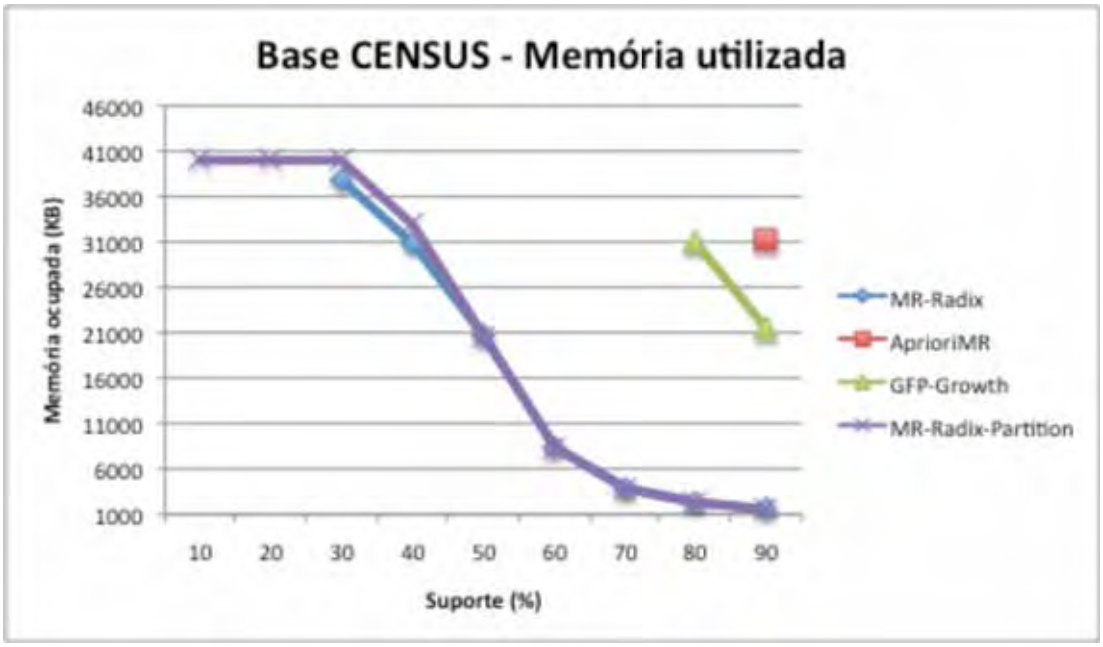

Figura 4.12 Gráfico de memória utilizada - base CENSUS

O algoritmo APRIORIMR apresentou novamente desempenho inferior em comparação aos demais algoritmos multirrelacionais. A medição para este algoritmo só foi possível para o valor mais alto de suporte, 90%, uma vez que, para todas as outras condições, o processamento não pode ser concluído devido à insuficiência de memória principal.

O algoritmo GFP-GROWTH também apresentou um consumo de memória elevado mesmo para valores mais altos de suporte quando comparado com os algoritmos MR-RADIX e MR-RADIX-PARTITION. Enquanto que o MR-RADIX, por exemplo, para o valor de suporte igual a 80%, necessitou de 2.250 KB (dois mil duzentos e cinquenta kilobytes), o GFP-GROWTH necessitou de 31.000 KB (trinta e um mil kilobytes). Ou seja, o MR-RADIX apresentou um consumo de quase quatorze vezes menor do que o GFP-GROWTH.

O comportamento, em termos de ocupação de memória dos algoritmos MR-RADIX e MR-RADIX-PARTITION, foi semelhante em todos os níveis de suporte analisados. Como já mencionado, isso ocorre porque o algoritmo MR-RADIX-PARTITION possui a capacidade de identificação do espaço de memória disponível, efetuando o particionamento da base de dados somente quando necessário.

Observa-se que, para os valores de suporte mais baixos, nos quais somente o MR-RADIX-PARTITION concluiu o processamento, a quantia de memória utilizada mantém-se relativamente constante. Isso porque o algoritmo cria as partições de forma que as mesmas ocupem todo o espaço disponível em memória. Dessa forma, cada partição é otimizada para conter o maior número de registro possível, o que, consequentemente, acarreta um número menor de partições e também um número menor de acessos a disco.

Em comparação com o GFP-GROWTH, os algoritmos MR-RADIX-PARTITION e MR-RADIX se sobressaíram, principalmente, no que se refere ao uso otimizado da memória. Nos testes comparativos do tempo de execução, notou-se um desempenho ligeiramente superior do GFP-GROWTH, porém o consumo de memória verificado foi elevado quando comparado com o algoritmo proposto neste trabalho, o qual apresentou resultados aliando baixo tempo de execução e uma melhor utilização no uso da memória.

4.4 Considerações finais

Este capítulo apresentou o estudo comparativo realizado para verificar a eficiência do algoritmo proposto MR-RADIX em relação a algoritmos correlatos. Para isso, foi feita a separação entre dois conjuntos de testes. O primeiro deles consistiu na comparação entre o algoritmo multirrelacional MR-RADIX e o algoritmo tradicional que lhe serviu de base, PATRICIAMINE. Nessa fase, foi possível confirmar a eficácia e eficiência do algoritmo multirrelacional na mineração de base de dados relacionais, uma vez que o tempo de execução e a memória utilizadas por este apresentaram melhores índices em comparação ao algoritmo tradicional, o qual possui como limitação a necessidade da execução de junções que reúnem os dados provenientes de múltiplas tabelas.

O segundo conjunto de testes, por sua vez, avaliou as métricas de desempenho, comparando o algoritmo proposto neste trabalho com algoritmos existentes na literatura, tais como APRIORIMR e GFP-GROWTH. Também nessa fase foi possível observar um ganho no desempenho do algoritmo MR-RADIX. Isso porque observou-se, para a maioria dos casos, um melhor tempo de execução e um uso mais otimizado da memória. Faz-se o adendo de que foi exposta, também nessa fase, a estratégia de particionamento da base de dados. Essa é implementada pelo algoritmo proposto para lidar com a limitação de espaço em memória, subdividindo o conjunto de dados inicial em porções que possam ser integralmente processadas. Foi constatado que o uso de tal estratégia apresentou o resultado esperado para o algoritmo MR-RADIX, possibilitando que o mesmo seja utilizado em grandes bases de dados, nas quais o processamento não pode ser realizado pelos algoritmos convencionais, uma vez que não há disponibilidade de memória suficiente para a mineração.

Capítulo 5
Conclusões

5.1 Conclusões finais da proposta da dissertação

O presente trabalho objetivou a construção de um algoritmo multirrelacional de mineração de regras de associação para grandes bases de dados. Assim, de modo a fundamentar o trabalho, primeiramente foi descrito um panorama geral dos algoritmos tradicionais e multirrelacionais, o que possibilitou a identificação de limitações nas propostas existentes. Os algoritmos multirrelacionais atualmente presentes na literatura viabilizam a extração de padrões em bases de dados relacionais, porém não permitem a análise de grandes volumes de dados em múltiplas tabelas. Em geral, as propostas existentes baseiam seus processamentos de padrões em estruturas de dados, as quais devem estar integralmente disponíveis em memória. Para bases de dados de tamanho elevado, essa estrutura pode ser suficientemente grande, a ponto de não ser alocável em memória e, assim, acaba impossibilitando o uso desses algoritmos.

Para lidar com essa dificuldade, foi elaborado um algoritmo que otimiza o uso da memória por meio de uma estrutura de dados, denominada *Radix-tree*, a qual comprime eficientemente a representação da base de dados. Além disso, foram apresentadas modificações no algoritmo, as quais permitiram que o mesmo seja aplicado em base de dados relacionais compostas de múltiplas tabelas. Uma primeira modificação foi a extensão da representação de padrões, o que permitiu a representação dos padrões multirrelacionais. Com isso, possibilitou-se o armazenamento das informações específicas dos padrões provenientes de múltiplas

tabelas, as quais, cabe salientar, possuem um grau de complexidade superior em relação aos padrões tradicionais, isto é, de uma única tabela.

Cabe ressaltar, ademais, que foram efetuadas modificações no algoritmo original (PIETRACAPRINA; ZANDOLIN, 2003), adaptando-o à aplicação no contexto multirrelacional. Propôs-se uma estrutura auxiliar – denominada *ItemMap* – que otimiza o uso do espaço em memória. Tal estrutura realiza a substituição da representação dos itens relacionais por uma notação mais comprimida, reduzindo o tamanho ocupado pelos nós da estrutura principal. Como os padrões multirrelacionais são formados por um número maior de informações, o uso da *ItemMap* possibilita que tal complexidade não seja transmitida e armazenada na *Radix-tree*, favorecendo sobretudo a própria eficiência da mineração.

O presente trabalho também apresentou a proposta de uma nova estratégia para a busca de padrões multirrelacionais, a qual foi denominada "fusão de itens frequentes". Tal estratégia possibilita a mineração de *itemsets* frequentes provenientes de múltiplas tabelas sem a necessidade de efetuar a custosa operação de junção. Os itens provenientes das tabelas secundárias são processados e inseridos como nós na *Radix-tree* da tabela principal. Além disso, ressalta-se que o algoritmo gera e utiliza apenas uma única estrutura durante todo o processamento, diferentemente de outras propostas, nas quais são geradas uma estrutura por tabela (TEREDESAI et al., 2005).

Além de possibilitar a utilização mais eficiente da memória, a proposta apresentada também faz uso da estratégia de particionamento, o que permite que grandes bases de dados relacionais possam ser processadas e terem seus padrões frequentes extraídos. Para isso, o algoritmo divide a base de dados em partições, de modo que essas possam ser alocadas em memória e processadas, obtendo assim os padrões locais dessas unidades.

Essa proposta, cabe salientar, apresenta uma forma mais otimizada para a obtenção dos padrões globais a partir dos padrões locais por meio do armazenamento das informações que indicam em quais partições um determinado padrão local foi encontrado. Com isso, pode-se ignorar a contagem nessas partições, reduzindo, assim, o número de partições a serem verificadas para a obtenção do suporte global.

A funcionalidade da proposta do presente trabalho foi exposta por meio da comparação de sua aplicação com algoritmos similares existentes na literatura da

área. Os algoritmos utilizados no estudo comparativo foram codificados neste trabalho com base na versão descrita pelos respectivos autores. Com isso, ressalta-se que um subproduto do trabalho foi justamente o incremento no número e na diversidade de algoritmos disponíveis na ferramenta *fDMMR*.

Com relação ao estudo comparativo propriamente dito, verificou-se, primeiramente, a vantagem da utilização do algoritmo multirrelacional MR-RADIX para uso em base de dados relacionais. Para tanto, esse algoritmo foi analisado em comparação ao algoritmo tradicional PATRICIAMINE. Constatou-se que a mineração multirrelacional, de fato, apresenta melhor eficiência, uma vez que evita-se as custosas operações de junção de múltiplas tabelas. Além disso, a abordagem multirrelacional não apresenta perdas semânticas, ao contrário dos algoritmos tradicionais, que podem introduzir erros ou imprecisões durante a junção dos dados.

Em seguida, foi constatado, após testes em diferentes bases de dados, que o algoritmo MR-RADIX apresenta um desempenho superior em relação ao tempo de execução e à memória utilizada quando comparado a outras propostas multirrelacionais. Por fim, utilizou-se uma base com tamanho elevado para explicitar a eficácia do algoritmo MR-RADIX no processamento de grande volumes de dados. Tal algoritmo conseguiu prosseguir, por meio da estratégia de particionamento, o processamento em valores de suporte baixos, os quais acarretaram falha na execução dos demais algoritmos, devido à insuficiência de memória para a conclusão da mineração.

5.2 Sugestões de trabalhos futuros

A partir do trabalho desenvolvido, é possível elencar sugestões de pesquisas e desenvolvimentos para trabalhos futuros.

a) *Extensão do algoritmo MR-RADIX para contemplar maximal e closed itemsets*: tal extensão possibilitaria a extração desses dois tipos especiais de *itemsets* que possuem a característica de ignorarem subconjuntos frequentes. Com isso, uma quantia significativa de *itemsets* seria descartada – sem perdas semânticas significativas –, reduzindo-se o volume de padrões a serem submetidos à verificação pelo analista;

b) *Melhoria da estratégia de particionamento*: um trabalho que objetivasse o aprimoramento ou mesmo a eliminação da necessidade da terceira leitura da base de dados poderia ser desenvolvido. Desse modo, certamente haveria ganhos na eficiência, visto que a etapa de geração de *itemsets* frequentes globais consome substancial quantidade de tempo da estratégia de particionamento;
c) *Estudo e implementação de novas medidas de interesse*: outras medidas de interesse, tais como a peculiaridade e utilidade (GENG; HAMILTON, 2006), poderiam ser o eixo condutor de trabalhos investigativos. Isso permitiria a geração de padrões baseados em outras métricas, o que possibilitaria a extração de informações de interesse que não são possíveis com o uso das métricas tradicionais – suporte e confiança - atualmente implementadas;
d) *Viabilização de estudo comparativo entre algoritmos multirrelacionais de diferentes abordagens*: seria interessante um estudo que possibilitasse a análise comparativa entre os algoritmos multirrelacionais voltados à mineração direta de bases de dados relacionais, tal como o MR-RADIX proposto, e os algoritmos ILP ou baseados em grafos.

Como subproduto deste trabalho, foram efetuados alguns incrementos na ferramenta *fDMMR*. No entanto, identificou-se possíveis alvos de investigação para trabalhos futuros relacionados à melhoria dessa ferramenta, a saber:

a) *Elaboração de técnicas de pós-processamento*: isso permitiria uma melhor visualização e compreensão dos padrões obtidos na mineração. Como o número de padrões gerados pode ser muito elevado, o uso de tais técnicas tornaria a etapa de análise dos resultados mais eficiente, auxiliando o analista a interpretar os conhecimentos implícitos;
b) *Implementação de novos tipos de algoritmos de mineração*: poderia ser realizada a inclusão de novos algoritmos na *fDMMR* voltados à mineração de outros tipos de padrões, tais como agrupamentos, árvores de classificação, entre outros. Isso permitiria ampliar o leque de aplicação da ferramenta, dando base para estudos e desenvolvimentos de técnicas de prospecção de diferentes tipos de conhecimento.

Apesar das contribuições do presente trabalho, nota-se que ainda há várias investigações a serem desenvolvidas, principalmente em grandes bases de dados, área na qual as técnicas apresentadas ainda se mostram muito aquém da eficiência esperada. Porém, o objetivo deste trabalho foi alcançado no sentido de que foi desenvolvida e apresentada uma proposta de algoritmo multirrelacional para mineração de regras de associação em grandes bases de dados, o que consistia uma lacuna no atual estado da arte.

REFERÊNCIAS BIBLIOGRÁFICAS

AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. Mining Association Rules between Sets of Items in Large Databases. In: **Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data**, p.207–216, Mai. 1993.

AGRAWAL, R.; SRIKANT, R. Fast Algorithms for Mining Association Rules in Large Databases. In: **Proceedings of the 20th International Conference on Very Large Data Bases**, p.487-499, Set. 1994.

APPICE, A.; CECI, M.; MALERBA, D. Mining model trees: A multi-relational approach. **Lecture notes in computer science**, p. 4–21, 2003.

ARTHUR, D.; VASSILVITSKII, S. How Slow is the k-means Method? In: **Proceedings of the twenty-second annual symposium on Computational geometry**, p.144–153. ACM New York, NY, USA. 2006.

BARALIS, E.; CERQUITELLI, T.; CHIUSANO, S. IMine: Index Support for Item Set Mining. **IEEE Transactions on Knowledge and Data Engineering**, v. 21, n. 4, p. 493-506, 2009.

BERSON, A.; SMITH, S.; THEARLING, K. An overview of data mining techniques. In:_____. **An overview of data mining techniques.** McGraw-Hill, 1999.

BLOCKEEL, H.; DZEROSKI, S. Multi-Relational Data Mining 2005: Workshop Report. **ACM SIGKDD Explorations Newsletter**, v. 7, n. 2, p.126-128, Dez. 2005.

BODON, F. A fast apriori implementation. In: **Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations**, v. 90, 2003.

BODON, F. **A survey on Frequent Itemset Mining**. f. 97. Tese (Doutorado), Budapest University of Technold Economics, 2006.

BRANDT, N. *et al*. Mining multi-relational data, Technical report, **ISTProject MiningMart**, IST-11993, 2001.

BUEHRER, G.; PARTHASARATHY, S.; GHOTING, A. Out-of-core frequent pattern mining on a commodity PC. In: **Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06**, p. 86. New York, New York, USA: ACM Press, 2006.

CEGLAR, A.; RODDICK, J. F. Association Mining. **Computing**, v. 38, n. 2, 2006.

CLARE, A.; WILLIAMS, H. E.; LESTER, N. Scalable multi-relational association mining. In: **Proceedings of the 4th IEEE International Conference on Data Mining (ICDM'04)**, p.355-358, Nov. 2004.

DEHASPE, L.; RAEDT, L. De. Mining association rules in multiple relations. In: **Proceedings of the 7th Intl. Workshop on Inductive Logic Programming**, Praga, República Tcheca, p.125-132, 1997.

DEHASPE, L.; TOIVONEN, H. T. T. Discovery of Relational Association Rules. In: DZEROSKI, S.; LAVRAC, N. **Relational data mining**, p.189-212, Springer-Verlag, 2001.

DEXTERS, N.; PURDOM, P.; GUCHT, D.V. Analysis of Candidate-Based Frequent itemset. In: **Proceedings of the ACM Symposium on Applied Computing - Data MiningTrack**, 2006.

DOMINGOS, P. Prospects and challenges for multi-relational data mining. **ACM SIGKDD Explorations Newsletter***,* v. 5, n. 1, Jul. 2003.

DUARTE, D. **Utilizando técnicas de programacao lógica indutiva para mineração de banco de dados relacional**. 78 f. Dissertação (Mestrado) Universidade Federal do Paraná. Curitiba, 2001

DUNHAM, M.; XIAO, Y.; SEYDIM, A.; GRUENWALD, L.; HOSSAIN, Z. A survey of association rules. **ACM Computing Surveys**, p. 1-65, 2002.

DZEROSKI, S. Multi-Relational Data Mining: An Introduction. **ACM SIGKDD Explorations Newsletter**, v. 5, n. 1, p.1-16, Jul. 2003.

DZEROSKI, S.; RAEDT, L. D.; WROBEL, S. Multi-Relational Data Mining 2003: Workshop Report. **ACM SIGKDD Explorations Newsletter***,* v. 5, n. 2, p.200-202, Dez. 2003.

FAYYAD, U.; PIATETSKY-SHAPIRO, G.; PADHRAIC, S. From Data Mining to Knowledge Discovery: An Overview. In: FAYYAD, U. M. (Ed.) *et al*. **Advances in Knowledge Discovery and Data Mining**. Menlo Park: AAAI Press, The MIT Press, 1996. cap.1, p. 1-34.

FAYYAD, U.; UTHURUSAMY, R. Data mining and knowledge discovery in databases. **Communications of the ACM**, v. 39, n. 11, p. 24-26, 1996.

GANTI, V.; GEHRKE, J.; RAMAKRISHNAN, R. Mining very large databases. **Computer**, v. 32, n. 8, p. 38-45. doi: 10.1109/2.781633, 1999.

GARCIA, E. **Mineração de regras de associação multi-relacional quantitativas**. Dissertação (Mestrado). f. 84. Universidade Metodista de Piracicaba, 2008.

GARCIA, E.; VIEIRA, M. Estudo de caso de mineração de dados multi-relacional. In: **XXIII Simpósio Brasileiro de Banco de Dados**, p.224-237, 2008.

GEIST, I. Declarative Data Mining: A framework for data mining and KDD. In: **Proceedings of the 2002 ACM symposium on Applied computing**, p.508-513, Mar. 2002.

GENG, L.; HAMILTON, H. J. Interestingness measures for data mining. **ACM Computing Surveys**, v. 38, n. 3, 2006.

GOETHALS, B. Survey on frequent pattern mining. **Manuscript**, p. 1-43, 2003.

GOPALAN, R.; SUCAHYO, Y. ITL-mine: Mining frequent itemsets more efficiently. In: **Proceedings of the 2002 Internatiional Conference on Fuzzy Systems and Knowledge Discovery**, p.167–172. Citeseer., 2002.

GOPALAN, R.; SUCAHYO, Y. Treeitl-mine: Mining frequent itemsets using pattern growth, tid intersection, and prefix tree. **Lecture notes in computer science**, p. 535–546. Springer, 2002.

GOPALAN, R.; SUCAHYO, Y. High performance frequent patterns extraction using compressed FP-tree. In: **Proceedings of the SIAM International Workshop on High Performance and Distributed Mining**, Orlando, USA, 2004.

GRAHNE, G.; ZHU, J. Efficiently using prefix-trees in mining frequent itemsets. In: **Proceedings of the ICDM Workshop on Frequent Itemset Mining Implementations**. Citeseer, 2003.

_____. Mining Frequent Itemsets from Secondary Memory. In: **Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM'04)**, p. 91-98, 2004.

_____. Fast algorithms for frequent itemset mining using FP-trees. **IEEE Transactions on Knowledge and Data Engineering**, v. 17, n. 10, p. 1347-1362. 2005.

GUO, H.; VIKTOR, H. Mining relational databases with multi-view learning. In: **Proceedings of the 4th international workshop on Multi-relational mining - MRDM '05**, p. 15-24. New York, USA: ACM Press, 2005.

GYORODI, R. **A Comparative Study of Iterative Algorithms in Association Rules Mining**, v. 12, n. 3, p. 205-213, 2003.

GYORODI, R.; GYORODI, C.; HOLBAN, S. A Comparative Study of Association Rules Mining Algorithms. In: **Proceedings of Hungarian Joint Symposium on Applied Computational Intelligence**, v. 40, 2004.

GYORODI, R.; GYORODI, C.; PATER, M.; BOC, O.; DAVID, Z. AFOPT algorithm for multilevel databases. In: **Proceedings of the IEEE Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'05)**, n. 1, p. 5, 2005.

HABRARD, A.; BERNARD, M.; JACQUENET, F. Multi-relational data mining in medical databases. **Artificial Intelligence in Medicine: 9th Conference on**, p. 1-10,

2003.

HAMIDA, M. **Patricia-Tree based algorithm to find frequent closed itemsets**. 70 f. Dissertação (Mestrado). Faculty of Sciences of Tunis, 2005.

HAMIDA, M.; SLIMANI, Y. A Patricia-Tree Approach for Frequent Closed Itemsets. In: **WEC'05: The Second World Enformatika Conference**, p.149-152. Istanbul, Turquia., 2005.

HAN, J.; CHEN, M.; YU, P. S. Data Mining: An Overview from Database Perspective. **IEEE Transactions On Knowledge And Data Engineering**, v. 8, n. 6, p.866-883, Dez. 1996.

HAN, J.; KAMBER, M. **Data mining: concepts and techniques.** 2 ed. San Francisco: Morgan Kaufmann Publishers, 2006. 743 p.

HAN, J.; PEI, J.; YIN, Y. Mining frequent patterns without candidate generation. In: **Proceedings of the 1996 ACM SIGMOD Int. Conf. Management of Data (SIGMOD'00)**, p.1-12, Dallas, Mai. 2000.

HIPP, J.; GÜNTZER, U.; NAKHAEIZADEH, G. Algorithms for association rule mining - a general survey and comparison. **ACM SIGKDD Explorations Newsletter**, v. 2, n. 1, p. 58–64. ACM New York, NY, USA, 2000.

IMIELINSKI, T; VIRMANI, A. Msql: A query language for database mining. **Data Mining and Knowledge Discovery**, v. 3, n. 4, p. 373–408, 1999.

INOKUCHI, A.; WASHIO, T.; MOTODA, H. An apriori-based algorithm for mining frequent substructures from graph data. In: **Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery**, p.13-23, 2000.

IVÁNCSY, R.; KOVÁCS, F.; VAJK, I. An analysis of association rule mining algorithms. In: **Proceedings of Fourth International ICSC Symposium**, 2004.

KANODIA, J. **Structural advances for pattern discovery in multi-relational databases.** 104 f. Dissertação(Mestrado), Rochester Institute Of Technology, 2005.

KANTARDZIC, M. **Data Mining: concepts, models, methods and algorithms**. New Jersey: John Wiley & Sons, 2003.

KETKAR, N. S.; HOLDER, L. B.; COOK, D. J. Comparison of graph-based and logic-based multi-relational data mining. **ACM SIGKDD Explorations Newsletter**, v. 7, n. 2, p.64-71, 2005.

KING, R.; SRINIVASAN, A.; DEHASPE, L. Warmr: a data mining tool for chemical data. **Journal of computer-aided molecular design**, v. 15, n. 2, p. 173-81, 2001.

KNOBBE, A. J. **Multi-Relational Data Mining.** 130 f. Tese (Doutorado), The Netherlands, 2004.

KNOBBE, A. J.; BLOCKEEL, H.; SIEBES, A.; VAN DER WALLEN, D.M.G. Multi-relational data mining. In: **Proceedings of Benelearn**, 1999.

KNOBBE, A. J.; SIEBES, A.; MARSEILLE, B. Involving Aggregate Functions in Multi-Relational Search. In: **Proceedings of PKDD 2002**, Helsinki, Finlândia, 2002.

KOOPMAN, A.; SIEBES, A. Characteristic Relational Patterns. In: **Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining**, p.437-445. Paris, 2009.

KURAMOCHI, M.; KARYPIS, G. An efficient algorithm for discovering frequent subgraphs. **IEEE Transactions On Knowledge and Data Engineering,** v. 16, n. 9, p.1038-1051, 2004.

LIU, G.; LU, H.; YU, J.; WEI, W.; XIAO, X. Afopt: An efficient implementation of pattern growth approach. In: **Proceedings of the ICDM workshop on frequent itemset mining implementations**, 2003.

LIU, J.; PAN, Y.; WANG, K.; HAN, J. Mining frequent item sets by opportunistic projection. **Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '02**, p. 229. New York, New York, USA: ACM Press, 2002.

LIU, L.; LI, E.; ZHANG, Y.; TANG, Z. Optimization of frequent itemset mining on multiple-core processor. In: **Proceedings of the 33rd international conference on Very large data bases**, p.1275-1285, VLDB Endowment, 2007.

MATSUDA, T.; HORIUCHI, T.; MOTODA, H.; WASHIO, T. Extension of graph-based induction for general graph structured data. In: **Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery**, p.420-431, 2000.

MORISSON, D. R. PATRICIA-Practical Algorithm To Retrieve Information Coded in Alphanumeric. **Journal of the ACM (JACM)**, v. 15, n. 4, p. 514-534, 1968.

NEZHAD, J.; SADREDDINI, M. PTclose: A novel algorithm for generation of closed frequent itemsets from dense and sparse datasets. In: **Proceedings of the World Congress on Engineering**, v. 1, p.3-7. London, 2007.

NIJSSEN, S.; KOK, J. Faster association rules for multiple relations. In: **International Joint Conference on Artificial Intelligence**, v. 17, p. 891–896. Citeseer, 2001.

NIJSSEN, S.; KOK, J. N. Faster association rules for multiple relations. NEBEL, B. (ed). In: **Proceedings of the 17th international joint conference on artificial intelligence**, Morgan Kaufmann, p.891-896, 2001.

ORDONEZ, C. Integrating K-means clustering with a relational DBMS using SQL. **IEEE Transactions on Knowledge and Data Engineering**, v. 18, n. 2, p. 188-201. doi: 10.1109/TKDE.2006.31, 2006.

OYAMA, F. T. **Extração de conhecimento em bases de dados multi-relacionais por meio de agrupamento de tuplas**. 51 f. Monografia (Graduação), Universidade Estadual Paulista, 2006.

OZKURAL, E.; AYKANAT, C. A space optimization for FP-growth. In: **Proceedings of the ICDM Workshop on Frequent Itemset Mining**, 2004.

PAGE, D.; CRAVEN, M. Biological applications of multi-relational data mining. **ACM SIGKDD Exploration Newsletter**, v. 5, n. 1, p.69-79, Jul. 2003.

PARK, J.; CHEN, M.; YU, P. An effective hash-based algorithm for mining association rules. In: **Proceedings of the 1995 ACM SIGMOD international conference on Management of data**, p.175–186. ACM New York, NY, USA. Retrieved from http://portal.acm.org/citation.cfm?id=223784.223813, 1995.

PATEL, D.; HSU, W.; LEE, M. Mining relationships among interval-based events for classification. In: **Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08**, p.393. New York, New York, USA: ACM Press, 2008.

PEI, J.; HAN, J.; LU, H.; *et. al*. H-mine: hyper-structure mining of frequent patterns in large databases. In: **Proceedings of the IEEE International Conference on Data Mining**, p.441-448. IEEE Computer Society, 2001.

PIETRACAPRINA, A.; ZANDOLIN, D. Mining frequent itemsets using patricia tries. In: **Proceedings of the IEEE ICDM Workshop Frequent Itemset Mining Implementations,** v. 80. Citeseer, 2003.

PIZZI, L. C. **Mineração de dados em múltiplas tabelas: o algoritmo GFP-Growth.** 106 f. Dissertação(Mestrado), Universidade Federal de São Carlos, 2006.

PIZZI, L.; RIBEIRO, M.; VIEIRA, M. Analysis of Hepatitis Dataset using Multirelational Association Rules. In: **Proceedings of the ECML/PKDD Discovery Challenge**, 2005.

RÁCZ, B. Nonordfp: An FP-growth variation without rebuilding the FP-tree. In: **Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04,** v. 126. Citeseer, 2004.

RIBEIRO, M. **Mineração de dados em múltiplas tabelas fato de um data warehouse**. 121 f. Dissertação (Mestrado), Universidade Federal de São Carlos, 2004.

RIBEIRO, M. X.; VIEIRA, M. T. P.; TRAINA, A. J. M. Mineração de Regras de Associação Usando Agrupamentos. In: **I workshop sobre algoritmos de mineração de dados**, p. 9-16. Uberlândia, Brasil, 2005.

SAID, A.; DOMINIC, P.; ABDULLAH, A. A Comparative Study of FP-growth Variations. **IJCSNS**, v. 9, n. 5, p. 266-272, 2009.

SARAWAGI, S.; THOMAS, S.; AGRAWAL, R. Integrating association rule mining

with relational database systems: Alternatives and implications. **Data Mining and Knowledge Discovery**, v. 4, n. 2, p. 89–125. Springer, 2000.

SAVASERE, A.; OMIECINSKI, E.; NAVATHE, S. **An efficient algorithm for mining association rules in large databases**. Atlanta: Georgia Institute of Technology, 1995.

SENKUL, P.; TOROSLU, H. ILP-based concept discovery in multi-relational data mining. **Expert Systems with Applications**, v. 36, n. 9, p. 11418-11428, 2009.

SHANG, X. **SQL Based Frequent Pattern Mining**. 146 f. Tese (Doutorado), 2005.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados**. 3 ed. São Paulo: Pearson Education do Brasil, 1999. 777 p.

SRIKANT, R.; AGRAWAL, R. Mining quantitative association rules in large relational tables. **ACM SIGMOD Record**, v. 25, p.1–12. NY, USA: ACM New York, 1996.

SUCAHYO, Y.; GOPALAN, R. CT-ITL: Efficient frequent item set mining using a compressed prefix tree with pattern growth. In: **Proceedings of the 14th Australasian database conference,** v. 17, p.95–104. Australian Computer Society, Inc. Darlinghurst, Australia, 2003.

TEREDESAI, A.; AHMAD, M.; KANODIA, J.; GABORSKI, R. CoMMA: a framework for integrated multimedia mining using multi-relational associations. **Knowledge and Information Systems**, v. 10, n. 2, p. 135-162, 2005.

TOIVONEN. H. Sampling Large Databases for Association Rules. In: **Proceedings of the 22nd. VLDB Conference**, 1996.

TSECHANSKY, M. S.; PLISKIN, N.; RABINOWITZ, G.; PORATH, A. Mining Relational Patterns from Multiple Relational Tables. **Decision Support Systems**, v. 27, n. 1-2, p.177-195, 1999.

WANG, K.; TANG, L.; HAN, J.; LIU, J. Top Down FP-Growth for Association Rule Mining. In: **Lecture notes in computer science**. p.334–340. Springer, 2002.

WANG, Y. **Categorization of Association Rule Mining Algorithms Framework**, TM Work Paper. p.1-20, 2004.

WOJCIECHOWSKI, M.; ZAKRZEWICZ, M. HASH-MINE: A New Framework for Discovery of Frequent Itemsets. In**: ADBIS-DASFAA Symposium,** p.215–222, 2000.

YAN, X.; CHENG, H.; HAN, J.; XIN, D. Summarizing itemset patterns: A profile-based approach. In: **Proceedings of the 2005 Int. Conf. Knowledge Discovery and Data Mining (KDD'05)**, Chicago, Ago. 2005.

YIN, X. **Scalable mining and link analysis across multiple database relations**, 154 f. Tese (Doutorado), University of Illinois at Urbana-Champaign, 2007.

YU, P.; CHEN, M.; HAN, J. Data mining: an overview from a database perspective. **IEEE Transactions on Knowledge and Data Engineering**, v. 8, n. 6, p. 866-883, 1996.

ZHANG, Y. Multi Relational Rules Mining in Data Warehouse. In: **Proceedings of the Second IEEE International Workshop on Knowledge Discovery and Data Mining**, p.944-947, IEEE Computer Society, 2009.

ZHAO, Q.; BHOWMICK, S. Association rule mining: A survey. **Nanyang Technological University, Singapore**, n. 2003116, 2006.

Apêndice A
Descoberta de Conhecimento em Bases de Dados

Han e Kamber (2006) conceituam a mineração de dados como a tarefa de descoberta de padrões interessantes extraídas de um grande volume de dados armazenados em base de dados, *data warehouse*[10] ou outro repositório. Para os autores, a mineração de dados deve ser vista como o próximo passo após o armazenamento dos dados, transformando-os em informações úteis. Ademais, apontam o *data mining* como um campo inter-disciplinar, que envolve áreas como sistemas de banco de dados, estatística, aprendizado de máquina, visualização de dados, computação de alto desempenho, redes neurais, reconhecimento de padrões, entre outros.

A confluência dessa diversidade de disciplinas originou uma grande variedade de sistemas de *data mining*. Para auxiliar na distinção e identificação desses sistemas, foram criados diferentes métodos de classificação, sendo que a classificação baseada no tipo de conhecimento extraído é a mais utilizada, pois apresenta mais claramente as diferenças de técnicas e requerimentos do *data mining* (HAN; CHEN; YU, 1996).

Com base nessa classificação, os parágrafos seguintes descrevem as principais funcionalidades do *data mining*, relacionando-as com os tipos de conhecimento que são capazes de extrair dos dados:

---

[10] *Data warehouse* ou armazém de dados: repositório de dados integrado que mantém uma relação estrita com o tempo (histórico). Além disso, é não-volátil, ou seja, é mantido separadamente da base de dados operacional, pois é especificamente utilizado para tomada de decisões (OLAP e *data mining*).

Classificação: é um processo composto de duas etapas. Na primeira, chamada de etapa de treinamento, um conjunto de modelos ou funções é construído, de modo que classifique um item em uma das classes pré-definidas. A segunda etapa, denominada classificação, destina-se ao uso dos modelos obtidos anteriormente para a classificação de novos itens. Exemplo: uma aplicação bancária que classifique uma operação de empréstimo como segura ou arriscada.

Regressão: processo análogo à classificação, porém os modelos (funções) são utilizados para predizer valores atualmente indisponíveis. É utilizado, por exemplo, para estimar a probabilidade de um paciente sobreviver dado um conjunto de resultados de testes.

Análise de associação: é a descoberta de relacionamentos ou regras de associação interessantes (*interesting)* entre itens em um conjunto de dados. Um exemplo clássico de uso é a "análise da cesta de compras"[11], que objetiva a busca por associações entre diferentes itens. A descoberta dessas associações pode ajudar, por exemplo, os estrategistas de *marketing* a localizar itens que são frequentemente comprados juntos pelos consumidores.

Análise de agrupamento[12]: realiza a divisão do conjunto de dados em agrupamentos (*clusters*) de objetos, de modo que os objetos pertencentes a um determinado grupo possuam alta similaridade entre si e baixa similaridade aos objetos dos demais grupos. Um exemplo de uso dessa funcionalidade é a identificação de sub-populações de consumidores. Agrupando-se indivíduos que compartilham características em comum - como, por exemplo, a renda mensal - é possível identificar potenciais grupos de compradores de um produto.

Sumarização: processo que envolve métodos para encontrar uma descrição compacta para um conjunto de dados. Um tipo de sumarização simples e que é utilizado em várias ocasiões é a representação de um conjunto de valores numéricos por meio da média e do desvio padrão.

Análise e detecção de desvios: análise que objetiva localizar objetos que não condizem com o comportamento geral do conjunto de dados. Embora na maioria dos casos haja o descarte de tais objetos, uma vez que eles são tratados como ruídos ou exceções, existem aplicações importantes que utilizam esse tipo de análise. Um dos

[11] Em inglês, *market basket analysis.*
[12] Em inglês, o termo correspondente é *clustering*.

mais conhecidos é o uso em detecção de fraudes, nos quais os eventos raros são mais interessantes que os regulares.

O processo de Descoberta de Conhecimento em Bases de Dados (KDD) é definido como um método não-trivial, iterativo, interativo, composto de várias etapas que, ao final, evidenciam padrões úteis e ocultos nos dados iniciais (GEIST, 2002). Embora os termos KDD e *data mining* muitas vezes são utilizados como sinônimos, há uma distinção entre eles. O termo KDD refere-se a todo o processo de descoberta de conhecimento útil de uma base de dados, enquanto que o *data mining* refere-se à aplicação de algoritmos para extração de padrões em uma base de dados, ou seja, caracteriza-se apenas como uma das etapas do processo de KDD (FAYYAD; PIATETSKY-SHAPIRO; PADHRAIC, 1996).

A Figura A.1 mostra um esquema do processo de obtenção de conhecimento a partir de um repositório de dados. As tarefas realizadas antes da etapa de *data mining* visam a preparação dos dados e, por esse motivo, são consideradas rotinas de pré-processamento. Por outro lado, as etapas seguintes ao *data mining* visam a apresentação do conhecimento extraído, sendo chamadas de rotinas de pós-processamento.

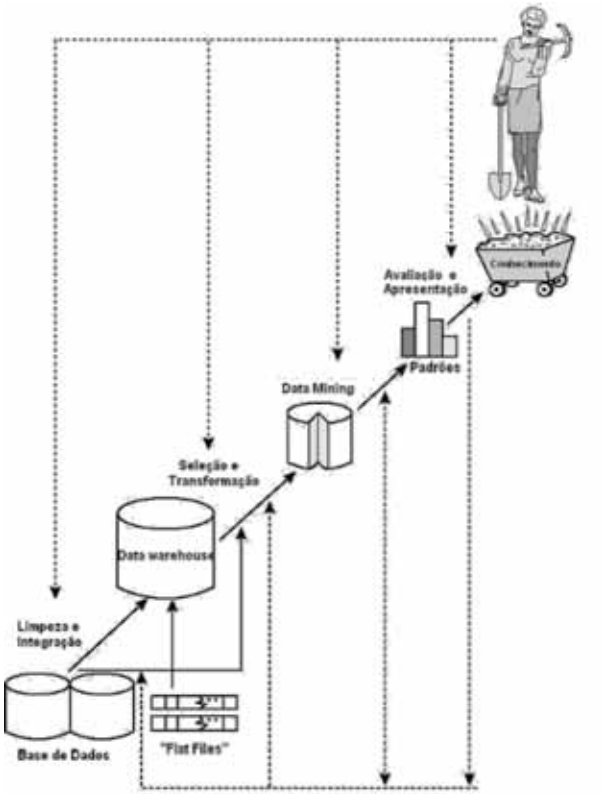

Figura A.1 Detalhes das etapas do KDD (HAN; KAMBER, 2006)

De acordo com o esquema acima, o processo de Descoberta de Conhecimento é uma sequência iterativa de passos, os quais são descritos por Han e Kamber (2006):

- **Limpeza de dados**: realiza o tratamento de valores nulos, a remoção de ruídos e a correção de dados inconsistentes. Este procedimento é importante para a garantia da qualidade dos resultados futuros;
- **Integração de dados**: múltiplas fontes de dados são combinadas em uma única, tratando eventuais redundâncias, conflitos e heterogeneidade de dados e de esquema;
- **Seleção de dados**: os dados relevantes à análise são separados dos demais e recuperados da base de dados, com o intuito de reduzir a carga de trabalho e, assim, melhorar o desempenho do processo;
- **Transformação de dados**: os dados são transformados para um formato apropriado para a operação de extração;
- ***Data mining***: processo essencial do KDD no qual técnicas são aplicadas para a extração de padrões dos dados;
- **Avaliação de padrões**: utiliza medidas de interesse para identificar padrões relevantes obtidos pelo *data mining*;
- **Apresentação do conhecimento**: etapa que utiliza técnicas de visualização de dados e representação do conhecimento para apresentar os resultados ao usuário.

É provável que, em certos contextos, um ou mais passos do processo de KDD sejam omitidos, como quando os dados já se encontram inicialmente "limpos" e integrados. Em contrapartida, há casos em que determinadas etapas são consideradas importantes e devem ser priorizadas. Por exemplo, se houver a necessidade de apresentar padrões de fácil interpretação ao analista, faz-se necessário um refinamento da etapa de apresentação do conhecimento, por meio do uso de técnicas elaboradas, como a apresentada no trabalho de Yan *et al.* (2005).

A única etapa que realmente deve estar presente para justificar o processo de extração de conhecimento é o *data mining*, considerada a etapa central e a mais importante do KDD.

Apêndice B
Algoritmos APRIORI e FP-GROWTH

B.1 Algoritmo APRIORI

O algoritmo APRIORI foi idealizado e formalizado por Agrawal e Srikant (1994), a partir dos modelos matemáticos para a extração de regras de associação booleanas. O seu funcionamento é baseado no conceito de "geração-e-teste de candidatos", que divide cada iteração do algoritmo em duas fases. A primeira fase visa a obtenção dos k-*itemsets* candidatos, a partir do conjunto de (k-1)-*itemsets* frequentes. Na segunda fase é realizada a contabilização dos k-*itemsets*, selecionando somente aqueles que atendem à frequência mínima pré-estabelecida.

Uma característica marcante do APRIORI é o processo de busca por largura, de modo que na i-ésima iteração são encontrados os i-*itemsets* frequentes. Além disso, como já foi indicado anteriormente, a geração dos k-*itemsets* candidatos depende do conjunto dos (k-1)-*itemsets* frequentes, ou seja, para obter padrões em uma dada iteração, é necessário consultar conhecimentos prévios (*prior knowledge*), fato este que deu origem ao nome do algoritmo (HAN; KAMBER, 2006).

A eficiência desse algoritmo está ligada com a possibilidade de redução do espaço de busca a cada iteração. Para isso, o algoritmo leva em consideração a propriedade APRIORI, a qual diz que se um *itemset* I é frequente, então todos os subconjuntos não-vazios de I são também frequentes (AGRAWAL; SRIKANT, 1994). Em outras palavras, se I é um i*temset* infrequente, qualquer outro *itemset* J que contenha I também é infrequente, pois a frequência de J não será maior que a frequência de I. Assim, o algoritmo consegue reduzir uma quantidade considerável

de *itemsets* candidatos gerados, eliminando aqueles que possuem subconjuntos não-frequentes.

O funcionamento básico do APRIORI pode ser descrito conforme o pseudocódigo a seguir:

```
Algoritmo: Apriori
Entradas: Base de dados D; minsup
Saída: Itemsets frequentes L

1)   L1 := 1-itemsets_frequentes( D, minsup );
2)   para ( k = 2; Lk-1 ≠ Ø; k++ ) {
3)        Ck = apriori_gen( Lk-1, minsup );
4)        para cada tupla (transação) t em D {
5)             Ct = subconjunto(Ck,t)
6)             para cada candidato c em Ct
7)                  c.contador++;
8)        }
9)        Lk = {c ∈ Ct | c.contador ≥ minsup }
10)  }
11)  retorne L = Uk Lk;
```

Figura B.2 Algoritmo APRIORI: principal (AGRAWAL; SRIKANT, 1994)

Para exemplificar o funcionamento do algoritmo, considere a base de dados da Tabela B.1 e *minsup*=2.

Tabela B.1 Exemplo de base de dados

TID	ITENS
1	leite, pão, café
2	pão, manteiga, frutas
3	carne, manteiga, pão
4	leite, frutas
5	leite, pão

Em primeiro lugar, o algoritmo examina a base de dados com o intuito de encontrar o conjunto de 1-*itemsets* que satisfazem o valor de suporte mínimo (*minsup*), originando o conjunto L_1 (linha 1). A Tabela B.2 apresenta os valores de suporte dos 1-*itemsets* da base de dados, obtidos por meio da contagem das ocorrências de cada item na base de dados. Por sua vez, o conjunto L_1, representado na Tabela B.3, é formado pelos 1-*itemsets* que apresentam o valor de suporte igual ou maior ao valor de *minsup*.

Tabela B.2 Suporte dos 1-*itemsets*

ITEMSET	SUPORTE
pão	4
leite	3
frutas	2
manteiga	2
café	1
carne	1

Tabela B.3 Conjunto L_1

ITEMSET	SUPORTE
pão	4
leite	3
frutas	2
manteiga	2

A seguir, o algoritmo entra no laço principal (linhas 2 a 10) que realizará alternadamente as tarefas de geração do conjunto de candidatos C_k e de obtenção do conjunto de *itemsets* frequentes L_k, sendo k >= 2, até que o processo termine na i-ésima iteração quando L_i for um conjunto vazio.

A função *apriori_gen* (linha 3) é responsável pela etapa de geração de C_k, utilizando para isso o conjunto de *itemsets* frequentes L_{k-1}. A função é detalhada no pseudocódigo abaixo:

```
Função: apriori_gen( L(k-1), minsup )
Entradas: conjunto L(k-1): (k-1)-itemsets frequentes; suporte mínimo minsup
Saída: conjunto Ck de k-itemsets candidatos

12) para cada itemset l1 ∈ L(k-1)
13)       para cada itemset l2 ∈ L(k-1)
14)             se( l1[1]=l2[1] ∧ l1[2]=l2[2] ∧ ... ∧ l1[k-2]=l2[k-2] ∧ l1[k-1]<l2[k-1] ) {
15)                   c = l1 ⋈ l2;
16)                   se tem_subconjunto_infrequente( c, L(k-1) )
17)                         exclua c;
18)                   senão adicione c ao Ck;
19)             }
20) retorne Ck;
```

Figura B.3 APRIORI: função apriori_gen (AGRAWAL; SRIKANT, 1994)

Esta função obtém os k-*itemsets* candidatos por meio da operação de junção, realizada dois a dois entre todos os (k-1)-*itemsets* do conjunto L_{k-1}. Assumindo que os itens de um *itemset* estão devidamente ordenados e que a notação $l_i[1]$ indica o primeiro item de l_i, $l_i[2]$ indica o segundo e, assim por diante, a junção pode ser definida como:

$$L_{k-1} \bowtie L_{k-1} = \{ l_1[1]l_1[2]...l_1[k-1]l_2[k-1], \text{ tal que } l_1, l_2 \in L_{k-1} \text{ e } l_1[1]=l_2[1] \wedge l_1[2]=l_2[2] \wedge ... \wedge l_1[k-2]=l_2[k-2] \wedge l_1[k-1]<l_2[k-1] \} \quad (A.1)$$

Para cada k-*itemset* obtido com a junção de dois (k-1)-*itemset* é, então, verificada a propriedade APRIORI, observando se o mesmo contém um subconjunto não-vazio infrequente. Dessa forma, apenas *itemsets* cujos subconjuntos são frequentes pertencerão ao conjunto de candidatos C_k. Reduzindo-se o tamanho de C_k, o algoritmo tenta diminuir a carga computacional das iterações seguintes.

Uma vez tendo o conjunto de candidatos C_k, o laço principal do algoritmo APRIORI executa a fase de contagem dos *itemsets* (linhas 4 a 8). A ideia é analisar as transações uma a uma, de modo a incrementar o contador de frequência dos k-itemsets que estão contidos nela. A seguir, os valores da ocorrência de cada *itemset* são comparados com o suporte mínimo esperado. Os *itemsets* que atenderem esse limitante são considerados frequentes e constituirão o conjunto L_k.

Considerando novamente o exemplo, o conjunto C_2 é obtido, segundo a função *apriori_gen*, por meio da combinação dos elementos pertencentes ao conjunto L_1. A Tabela B.4 apresenta os 2-*itemsets* originados dessa combinação, bem como os seus respectivos valores de suporte. Por sua vez, a Tabela B.5 apresenta o conjunto de 2-*itemsets* frequentes L_2, resultante da comparação dos valores de suporte dos elementos de C_2 com o limitante *minsup*, preservando somente aqueles que atendem ao valor especificado.

Tabela B.4 Conjunto C_2

ITEMSET	SUPORTE
pão, leite	2
pão, frutas	1
pão, manteiga	2
leite, frutas	1
leite, manteiga	0
frutas, manteiga	1

Tabela B.5 Conjunto L_2

ITEMSET	SUPORTE
pão, leite	2
pão, manteiga	2

Assim, o algoritmo incrementa o valor de k e inicia uma nova iteração, utilizando o conjunto de *itemsets* frequentes L_{k-1} para a geração dos novos k-*itemsets* candidatos e assim por diante.

Continuando o exemplo, pode-se obter o conjunto C_3 pela combinação dos elementos do conjunto L_2. Nesse caso, C_3 é formado por um único 3-*itemset*, cujo valor de suporte é 0. Como não há elementos em C_3 que apresentem valor de suporte igual ou superior ao *minsup*, o conjunto L_3 será vazio. Assim, o algoritmo chega à

sua condição de parada, retornando, no caso do exemplo, os conjuntos de *itemsets* frequentes L_1 e L_2.

Tabela B.6 Conjunto C_3

ITEMSET	SUPORTE
pão, leite, manteiga	0

B.2 Algoritmo FP-GROWTH

O algoritmo FP-GROWTH foi apresentado por Han, Pei e Yin (2000) como uma nova abordagem para extração de regras de associação, tentando reduzir as limitações impostas pelo método de "gerar-e-testar" do algoritmo APRIORI. Isso porque, à medida que se reduz o suporte mínimo, o número de candidatos cresce substancialmente, tornando a solução impraticável. Além disso, nos casos em que os *itemsets* frequentes são longos, o APRIORI necessita repetidamente de acessos a disco, prejudicando o desempenho do algoritmo (HAN; KAMBER, 2006).

A proposta do algoritmo FP-GROWTH é justamente extrair os *itemsets* frequentes de um modo mais eficiente, evitando a necessidade da etapa de geração de candidatos. A solução apresentada pelo algoritmo é a utilização de uma estrutura em memória denominada *FP-tree*, que representa comprimidamente a base de dados (HAN; PEI; YIN, 2000). Com essa estrutura, é possível obter os *itemsets* frequentes recursivamente utilizando-se de estratégias de busca de padrões em árvores. Devido à utilização da *FP-tree*, que se encontra em memória, o algoritmo FP-GROWTH reduz o número de varreduras na base de dados, sendo necessários apenas dois acessos a disco para a construção da árvore.

A *FP-tree* é composta de nós que armazenam informações referentes aos 1-itemsets frequentes da base de dados. Dentre as informações, está a frequência de ocorrência de cada *itemset*, a qual será necessária no processo de extração de padrões frequentes. O algoritmo de construção da árvore é exibido a seguir:

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

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

Figura B.4 Algoritmo FP-GROWTH: construção da FP-tree (HAN; PEI; YIN, 2000)

A etapa de construção da *FP-tree* inicia-se com uma varredura completa na base de dados, com o intuito de obter todos os 1-itemsets frequentes, de modo análogo ao algoritmo APRIORI (linha 1). Então, o conjunto de 1-itemsets frequentes é ordenado de modo decrescente pelo respectivo valor de suporte, originando uma lista ordenada L (linha 2). Além de informações de cada *itemset*, a lista L contém um campo denominado *node-link* para cada *itemset*, indicando qual o primeiro nó deste *itemset* inserido na árvore. Por esse motivo, tal lista também é conhecida como tabela *header*. Esse campo permite manter uma ligação entre os nós de mesmo *nome-item*, o que será útil no processo de busca de padrões frequentes.

Uma vez tendo o conjunto de 1-*itemsets* ordenados, inicia-se o processo propriamente dito de inclusão dos nós na árvore com a adição do nó raiz (linha 3). Cada nó na *FP-tree* possui basicamente o nome do item correspondente (*nome-item*), o contador de suporte e um ponteiro (*node-link*) utilizado para a ligação dos nós de mesmo *nome-item*. A Figura B.5 apresenta um exemplo da organização de uma *FP-tree*, juntamente com a lista L e as conexões dos *node-links*.

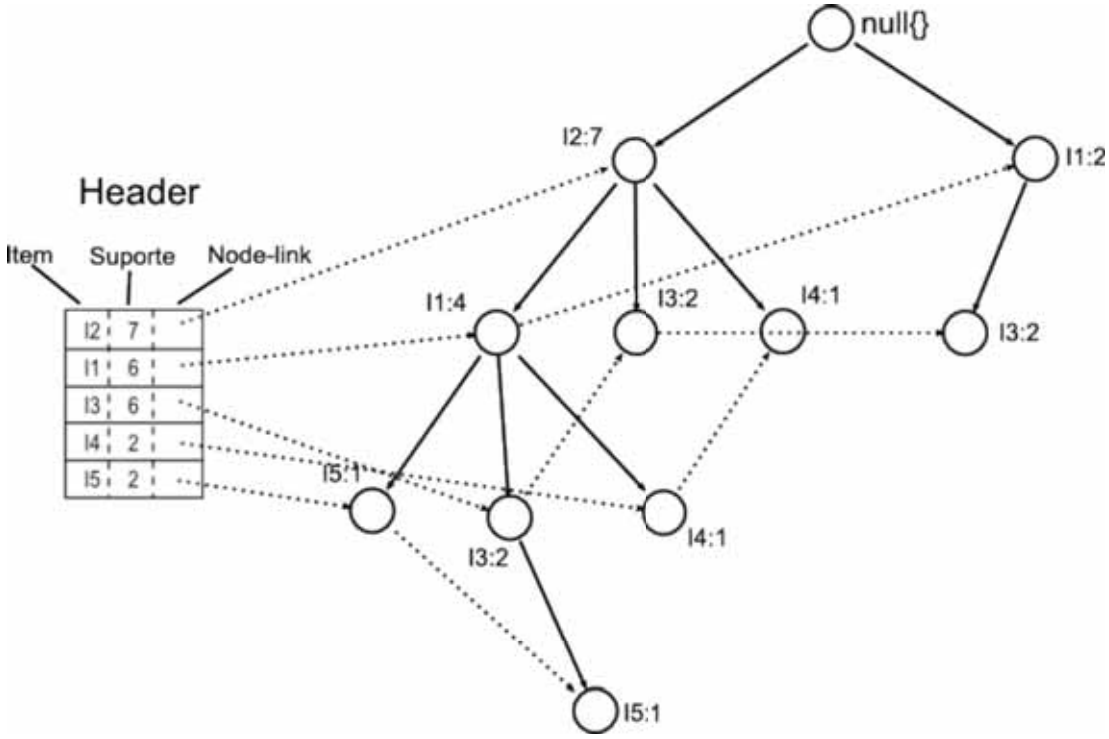

Figura B.5 Exemplo de uma *FP-tree* (HAN; KAMBER, 2006)

A seguir, é realizada a segunda varredura na base de dados. Nesse momento, cada transação é processada em duas etapas (linhas 4 a 7). A primeira etapa consiste em ordenar seus itens seguindo a ordenação da lista L. Por sua vez, a segunda etapa utiliza a função *insere_tree*, que é apresentada detalhadamente na figura abaixo, para a inclusão dos itens da transação na *FP-tree*.

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

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

Figura B.6 Algoritmo FP-GROWTH: função insere_tree (HAN; PEI; YIN, 2000)

A função *insere_tree* apresenta um funcionamento recursivo, inserindo ou atualizando um nó na *FP-tree* a cada chamada recursiva. Ela tem como parâmetros um conjunto ordenado, denotado por [p|P], e o nó "pai" N, no qual será inserido um novo nó, caso necessário. Quanto ao conjunto [p|P], p é o primeiro elemento da lista e P é o conjunto de elementos remanescentes.

A chamada desta função é disparada pelo algoritmo de construção da *FP-tree* (linha 6), passando como parâmetro o conjunto de itens da transação e o nó raiz da árvore correspondendo a, respectivamente, [p|P] e N. Inicialmente, a função *insere_tree* verifica se p – o primeiro item da lista – é um dos filhos do nó raiz. Se for, basta incrementar o contador de suporte do nó (linhas 1 e 2). Caso contrário, é necessária a adição deste nó na *FP-tree* como filho do nó N. Além disso, é feita a atualização dos *node-links* contemplando o novo nó (linhas 3 a 8). Daí, o processo recursivo é repetido agora envolvendo apenas o conjunto de itens remanescentes e tendo o nó adicionado ou atualizado como nó "pai" da nova recursão (linhas 9 e 10). Quando não houver itens remanescentes, ou seja, o conjunto P for vazio, a recursão pára indicando que todos os itens da transação em questão foram processados e devidamente inseridos na árvore.

Ao realizar o processamento de todas as transações, poderá ser iniciada a etapa de mineração da *FP-tree*, que objetiva a descoberta de padrões frequentes. De um modo geral, o processo consiste em gerar as bases de padrões condicionais (*conditional pattern base*) para cada 1-itemset presente na tabela *header*. Essas bases são utilizadas para a construção das *FP-tree* condicionais, as quais relacionam os caminhos frequentes que se conectam aos nós correspondentes ao 1-itemset em questão. Uma vez obtidas, tais *FP-tree* condicionais são utilizadas para encontrar os padrões frequentes que apresentam o 1-itemset como sufixo. Esse processo pode ser visto com mais detalhes no algoritmo exibido na figura abaixo:

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

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

Figura B.7 Algoritmo FP-GROWTH: extração de padrões da FP-tree (HAN; PEI; YIN, 2000)

A rotina de mineração de dados inicia-se com uma chamada na forma FP-GROWTH(T,null), sendo T a *FP-tree* em que se deseja extrair padrões. Inicialmente, o algoritmo consulta a tabela *header* da *FP-tree*, escolhendo o último elemento da lista, ou seja, aquele que apresenta o menor valor de suporte. Segundo Han e Kamber (2006), a razão disso é que tal escolha simplifica as operações posteriores, uma vez que o elemento não precisará ser considerado como parte do sufixo, pois todos os seus *itemsets* relacionados já terão sido encontrados.

Para exemplificar o funcionamento do algoritmo de extração de padrões, tome como exemplo a *FP-tree* da Figura B.5 e um suporte mínimo *minsup* = 2. Como a árvore apresenta mais de um caminho, a condição da linha 1 é falsa, de modo que o algoritmo executa o bloco "senão" (linhas 5 a 10). O primeiro elemento da tabela header a ser processado é o I5. Nesse caso, o padrão β seria o próprio I5, pois α foi inicializado com valor nulo. Então, o próximo passo é a construção das bases condicionais de β, que são os caminhos existentes na *FP-tree* cujo sufixo é o próprio β. Se β = I5, então, pode-se identificar pela *FP-tree* que os caminhos – e seus respectivos valores de suporte – que levam aos nós I5 são {(I2 I1 I5: 1)} e {(I2 I1 I3 I5: 1)}. Considerando que I5 é o sufixo desses caminhos, obtém-se a base condicional {(I2 I1: 1), (I2 I1 I3: 1)}.

No próximo passo, a base condicional é utilizada para a construção da *FP-tree* condicional $Tree_{\beta}$. Esse processo é semelhante à construção da *FP-tree* apresentada anteriormente. Assim, toma-se o primeiro caminho da base condicional (I2 I1: 1), inserindo-o ordenadamente na $Tree_{\beta}$. A seguir, repete-se a inserção para o

caminho (I2 I1 I3: 1), resultando na *FP-tree* condicional da Figura B.8. Nota-se que o nó I3 não se encontra na *FP-tree* condicional, visto que seu valor de suporte é menor que o *minsup*.

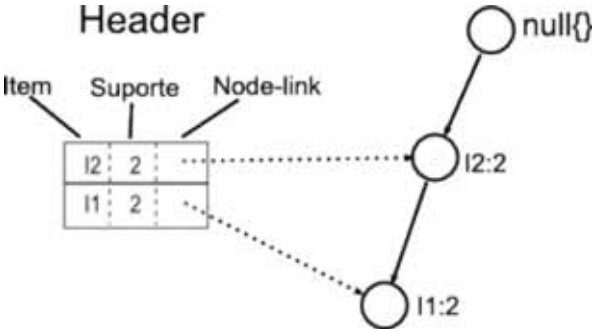

Figura B.8 *FP-tree* condicional para o nó I5 (HAN; KAMBER, 2006)

Após a construção da *FP-tree* condicional $Tree_\beta$ não-vazia, o algoritmo aplica uma chamada recursiva para processar a $Tree_\beta$ recém-criada. Agora, a condição da linha 1 é verdadeira, pois $Tree_\beta$ apresenta um único caminho P conectando todos os seus nós. Este é o caso base da recursão, em que finalmente são retornados os padrões frequentes relacionados a I5, que consistem em todas as combinações possíveis envolvendo os nós de P acrescidos do sufixo β. Assim, os padrões cujo sufixo é β = I5 são "I2 I5: 2", "I1 I5: 2" e "I2 I1 I5: 2".

A seguir, o algoritmo inicia o processamento envolvendo o próximo nó, I4, da tabela *header*. Primeiramente, é encontrada a sua base condicional, {(I2 I1: 1), (I2: 1) }, formada pelos caminhos que apresentam I4 como sufixo. A FP-tree condicional gerada com essa base consiste de apenas um nó {I2: 2}, já que I1 não satisfaz o suporte mínimo. Assim, o único padrão obtido é "I2 I4: 2".

O mesmo procedimento é aplicado sobre o próximo nó da tabela header, I3. Pela *FP-tree* consegue-se obter a base condicional de I3, que é composta de três caminhos {(I2 I1: 2), (I2: 2), (I1: 2)}. Daí, a *FP-tree* condicional $Tree_\beta$, com β = I3, possui dois ramos, conforme apresenta a figura abaixo.

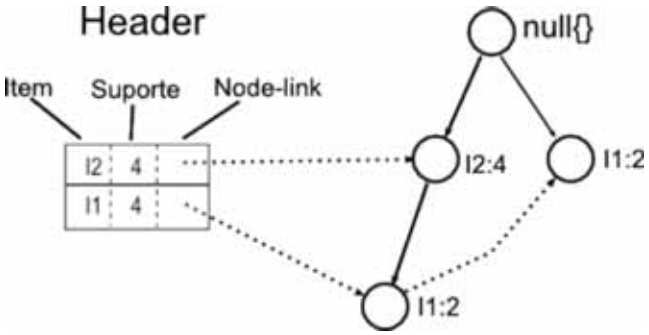

Figura B.9 *FP-tree* condicional para o nó I3 (HAN; KAMBER, 2006)

Diferentemente das anteriores, essa $FP\text{-}tree_{\beta}$ apresenta mais de um caminho, de modo que a chamada recursiva da função FP-Growth não cairá no caso base, necessitando ainda do processamento desta árvore condicional para a geração dos padrões frequentes. Inicialmente, é selecionado o último elemento da tabela *header* desta *FP-tree* condicional, ou seja, a_i = I1. Como o parâmetro α = I3, o valor de β torna-se "I1 I3". Um primeiro padrão frequente gerado é o β propriamente dito, de acordo com a linha 6 do algoritmo. O próximo padrão é encontrado gerando a base e a FP-tree condicional de β, as quais são compostas de um único nó I2, que origina o padrão frequente "I2 I1 I3: 2". Selecionando o próximo elemento, I2, da tabela *header* da $Tree_{\beta}$, o valor de β torna-se "I2 I3". Nota-se que não há nenhum caminho que tenha como sufixo esse padrão, de modo que a base e a *FP-tree* condicionais são vazias e não geram nenhum padrão frequente. Novamente, o próprio β é um padrão frequente, de modo que a mineração do item I3 da *FP-tree* resulta nos seguintes padrões: "I2 I3: 4", "I2 I1 I3: 2", "I1 I3: 2".

Analisando agora o item I1 na *FP-tree*, obtemos a base e a *FP-tree* condicional compostas de um único nó (I2: 4), que gera o padrão "I2 I1: 4". Por sua vez, o primeiro elemento da tabela *header* da *FP-tree*, I2, apresenta base condicional vazia, de modo que não há possibilidade de encontrar nenhum padrão cujo sufixo seja I2.

Apêndice C
Resultados experimentais do estudo comparativo

Esta seção visa apresentar os resultados brutos obtidos por meio dos diversos testes efetuados com o algoritmo MR-RADIX. Tais valores de métricas foram utilizados na confecção dos gráficos e tabelas do estudo comparativo previamente apresentado no "Capítulo 4 - Testes e Resultados".

C.1 Métricas dos testes envolvendo MR-RADIX e PATRICIAMINE

Tabela C.7 Tempo de execução e memória utilizada - Base HC

	MR-Radix		Patricia-Mine	
Suporte (%)	Tempo (ms)	Memória (bytes)	Tempo (ms)	Memória (bytes)
0,5	1047	1648200	6048	1804280
0,5	969	1501880	6047	1795576
0,5	969	1575288	6047	1798656
1	921	1396272	5875	1508736
1	922	1472008	5859	1513328
1	953	1561192	5859	1463288
10	797	1111208	5281	968312
10	828	1222632	5172	1006032
10	781	1181696	5156	1001232
20	765	1181352	5109	948856
20	766	1183520	5109	926664
20	766	1159664	5125	964472
30	766	1165352	5110	960520
30	765	1162144	5093	952408
30	843	1102008	5110	961880

Tabela C.8 Tempo de execução e memória utilizada - Base SIVAT

	MR-Radix		Patricia-Mine	
Suporte (%)	Tempo (ms)	Memória (bytes)	Tempo (ms)	Memória (bytes)
1	5828	7081936	12782	9121960
1	5719	7203032	12782	9214984
1	5703	7153760	12655	9095920
5	4891	4080384	11157	3957808
5	4859	4205904	11124	3908312
5	4937	4072296	11280	4025296
10	4610	3060488	10704	3269696
10	4687	3097720	10781	3262080
10	4641	3027824	10672	3397320
20	4422	2191488	10235	2409880
20	4375	2356424	10516	2377080
20	4437	2255248	10797	2284680
30	4266	1459720	10641	1878576
30	4234	1505840	10343	1825784
30	4313	1400640	10313	1808992

C.2 Métricas do teste envolvendo os algoritmos multirrelacionais

Tabela C.9 Tempo de execução e memória utilizada - multirrelacionais - Base HC

	MR-Radix		AprioriMR		GFP-Growth	
Suporte (%)	Tempo (ms)	Memória (bytes)	Tempo (ms)	Memória (bytes)	Tempo (ms)	Memória (bytes)
0,5	1031	1649816	103578	63273728	-	-
0,5	969	1616600	109562	63300488	-	-
0,5	984	1513240	110016	63276856	-	-
1	922	1482848	78500	37237200	-	-
1	922	1365352	78031	37315392	-	-
1	921	1535808	78360	37317512	-	-
10	781	1125688	44859	12130960	-	-
10	812	1092704	44172	11929824	-	-
10	812	1038272	44594	11796920	-	-
20	812	1150104	42375	11139392	-	-
20	782	1127368	42312	11067752	-	-
20	766	1191816	42516	11115560	-	-
30	766	1168376	41547	10793504	-	-
30	781	1137560	41750	11103768	-	-
30	781	1172944	41531	11023056	-	-

Tabela C.10 Tempo de execução e memória utilizada - multirrelacionais - Base SIVAT

	MR-Radix		AprioriMR		GFP-Growth	
Suporte (%)	Tempo (ms)	Memória (bytes)	Tempo (ms)	Memória (bytes)	Tempo (ms)	Memória (bytes)
1	5328	2979832	-	-	9094	6899168
1	5406	2967784	-	-	7218	6846712
1	5359	2839800	-	-	7765	7125984
5	4875	2953464	-	-	8234	3717696
5	4875	3054464	-	-	7687	3762424
5	4859	2843088	-	-	7156	4085880
10	4703	2886136	308734	37971888	7468	2690000
10	4750	2884664	308734	37971888	7651	3138408
10	4734	3027832	308734	37971888	7579	2832816
20	4578	2813736	119687	20543584	7423	2296008
20	4610	2874024	119687	20543584	7578	2506176
20	4594	2741544	119687	20543584	7345	3338344
30	4390	1894760	57359	15915424	6438	2657232
30	4421	1873400	57359	15915424	7391	2722398
30	4406	1835360	57359	15915424	8110	2655368

Tabela C.11 Tempo de execução e memória utilizada - multirrelacionais - Base CENSUS

	MR-Radix		MR-Radix-Partition		AprioriMR		GFP-Growth	
Sup (%)	Tempo (ms)	Memória (bytes)	Tempo (ms)	Memória (bytes)	Tempo (ms)	Memória (bytes)	Tempo (ms)	Memória (bytes)
10	-	-	514578	39993032	-	-	-	-
20	-	-	273078	39980680	-	-	-	-
30	76125	37747384	136860	39996336	-	-	-	-
40	68875	30741216	67938	32905376	-	-	-	-
50	64813	20617184	63375	20423616	-	-	-	-
60	60578	8290312	60093	8156448	-	-	-	-
70	58156	3720568	58187	3609240	-	-		
80	56672	2254320	56422	2165264	-	-	30094	31009000
90	54219	1548624	54828	1486936	878500	31141240	29625	21436896

Apêndice D
Ferramenta de apoio ao KDD

A fim de dar suporte ao processo de descoberta de conhecimento em base de dados, faz-se necessário o uso de uma ferramenta que seja capaz de proporcionar funcionalidades que complementem a etapa de mineração de dados. Para isso, utilizou-se a "Ferramenta de Data Mining Multirrelacional", ou simplesmente *fDMMR* (OYAMA, 2006), a qual implementa algumas etapas do KDD, tais como as fases de Seleção de Dados e de Visualização dos Resultados, bem como se apresenta como um relevante recurso computacional para o suporte aos algoritmos envolvidos, principalmente quando da efetivação dos testes e estudo comparativo.

No presente trabalho, foram efetuadas melhorias na *fDMMR* original, resultando em uma nova versão da ferramenta. Dentre os principais avanços obtidos, destacam-se:

- *Portabilidade*: adequação do código-fonte para uso da ferramenta em diferentes plataformas;
- *Interface gráfica*: além de haver a melhoria gráfica da ferramenta como um todo, deu-se especial atenção ao módulo de seleção de atributos, o qual apresenta visualmente todo o esquema da base de dados;
- *Módulo de medição de desempenho*: acréscimo das funcionalidades de coleta e armazenamento de métricas de desempenho dos algoritmos tais como tempo de execução, memória utilizada, número de nós nas estruturas e tempo de junção.

Na Figura D.10, é apresentada a interface principal da ferramenta, com destaque para a o módulo de seleção de atributos, o qual se encontra disposto na área à esquerda da figura. Neste, é apresentado visualmente o esquema da base de dados, possibilitando que o usuário da ferramenta veja o conjunto de relações e atributos, bem como o relacionamento entre as tabelas.

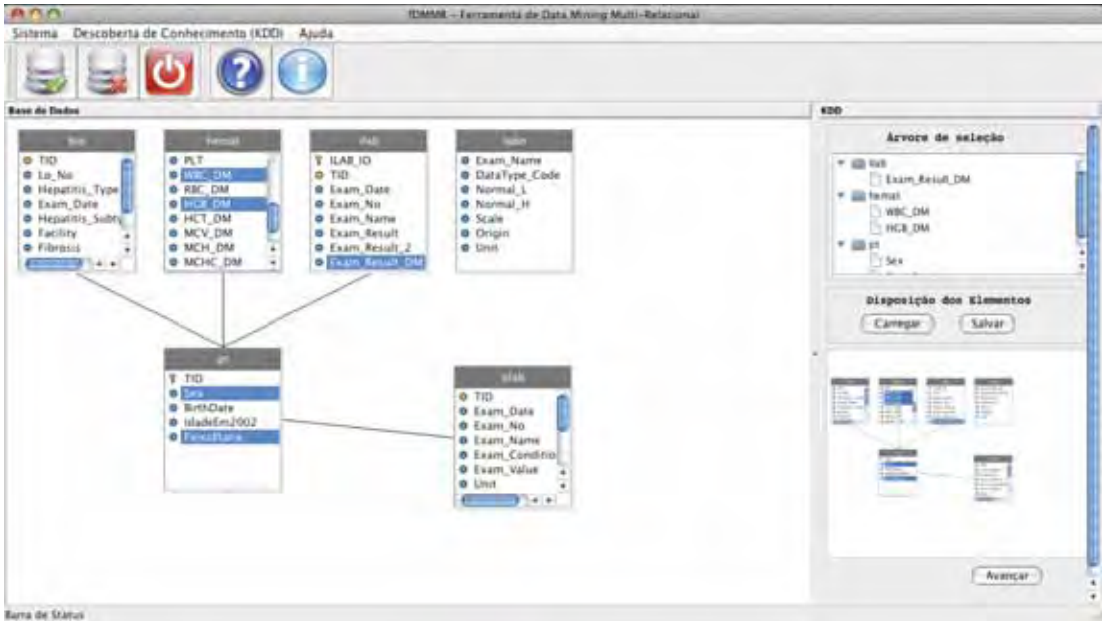

Figura D.10 Interface de seleção de atributos na fDMMR

Após a seleção dos atributos, pode-se efetuar a tarefa de mineração de dados, utilizando-se a interface apresentada na Figura D.11. Os parâmetros dos algoritmos, tais como valores de suporte e confiança, podem ser introduzidos pelo usuário com o uso do módulo especifico que se encontra à direita na figura mencionada.

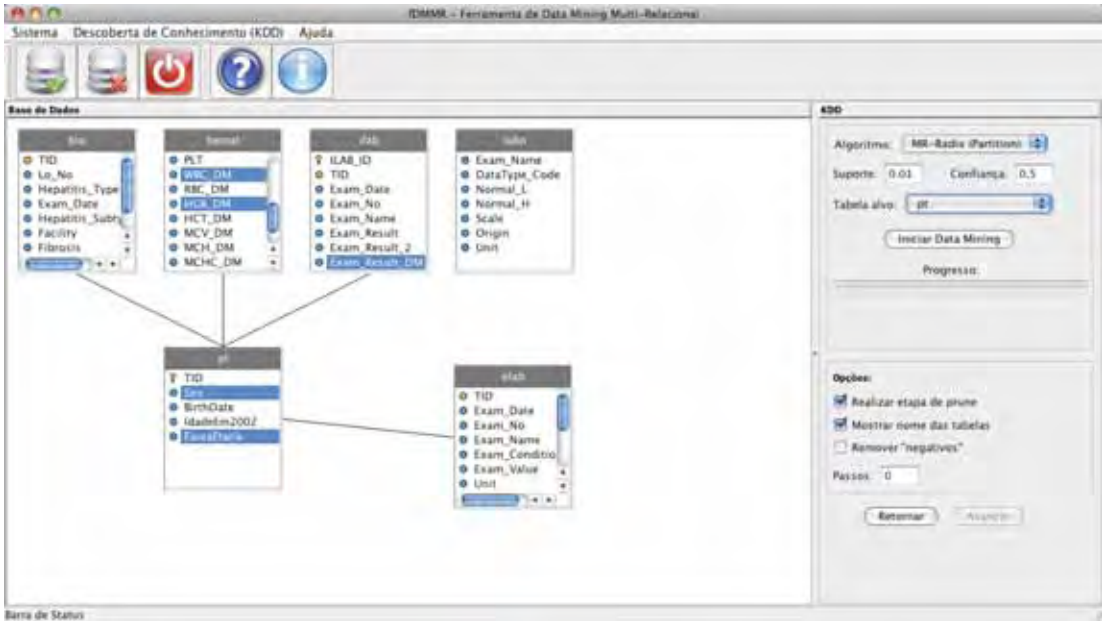

Figura D.11 Interface de mineração de dados na fDMMR

A ferramenta *fDMMR,* então, proporciona o suporte básico aos algoritmos de extração de regras de associação, uma vez que implementa funcionalidades auxiliares à mineração de dados, tais como a manipulação de dados relacionais, a seleção visual de atributos, a visualização de resultados e as medições de desempenho.