UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Mineração de Dados Em Múltiplas Tabelas: O Algoritmo GFP-Growth

Luciene Cristina Pizzi

São Carlos – SP
2006

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

P695mm

Pizzi, Luciene Cristina.
Mineração multi-relacional: o algoritmo GFP-growth / Luciene Cristina Pizzi. -- São Carlos : UFSCar, 2006.
107 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2006.

1. Tipos específicos de banco de dados. 2. Mineração multi-relacional. 3. Data mining (Mineração de dados). I. Título.

CDD: 005.75 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Mineração Multi-Relacional: O algoritmo GFP-Growth"

LUCIENE CRISTINA PIZZI

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Profa. Dra. Marina Teresa Pires Vieira
(Orientadora – PPG-CC/UFSCar)

Prof. Dr. Mauro Biajiz
(DC/UFSCar)

Prof. Dr. João Eduardo Ferreira
(IME/USP)

São Carlos
Maio/2006

Para meus pais, que apesar das dificuldades, sempre fizeram de tudo para que eu pudesse me dedicar aos estudos.

Agradecimentos

A Deus, aquele que sempre me indica os caminhos e orienta minhas escolhas;

À minha família, por acreditar em mim e apoiar meu trabalho;

À minha orientadora Marina, pela orientação, compreensão e prontidão em me ajudar;

Ao Thiago, pelo carinho, paciência e apoio nos momentos difíceis;

Às minhas amigas que conviveram comigo nesses dois anos de mestrado: Patrícia, Danielle, Aline, Thaíze e Glauce, pela paciência, compreensão e apoio;

Aos amigos do PPG-CC, por proporcionarem momentos alegres sempre que precisamos;

À Marcela, pela ajuda dispensada em meu trabalho;

A CAPES pelo apoio financeiro;

E a todos que, de alguma forma, contribuíram para essa minha vitória,

Muito obrigada!

Abstract

Data mining is the phase of the knowledge discovery in database process where an algorithm is applied to the available data, in order to prove a hypothesis or discover a still unknown pattern. The traditional data mining techniques can deal only with single tables; however it is interesting to look for patterns involving several related tables, aiming to analyze the existing relation between the entities present in one table and the data of the same entities present in another table.

Depending on the relationship existing between these tables, applying a traditional algorithm to the joint table is not sufficient, as the joint table may contain duplicated attribute values which interfere in the analysis process of the generated rules.

In order to solve this problem, this project adopts an approach which consists on looking for association rules mining the joint table. The adopted process considers the groups of tuples, where each group is formed by tuples of the same entity.

Following this approach the GFP-Growth algorithm was developed, which is presented in this monograph along with its results and comparisons with other multi-relational algorithms.

Resumo

A mineração de dados é a etapa do processo de descoberta de conhecimento na qual um algoritmo é aplicado sobre os dados disponíveis, com o intuito de provar uma hipótese ou descobrir algum padrão até então desconhecido. As técnicas tradicionais de mineração de dados tratam uma única tabela, no entanto é interessante buscar padrões que envolvam múltiplas tabelas relacionadas, com o intuito de analisar a relação existente entre os dados de uma entidade presentes em uma tabela e os dados dessa mesma entidade presentes em uma outra tabela.

Dependendo do tipo de relacionamento existente entre essas tabelas, não basta realizar a junção das mesmas para aplicar um algoritmo tradicional de mineração de dados na tabela resultante, pois essa tabela pode conter duplicação de valores de atributos que interferem no processo de análise das regras geradas.

Para resolver esse problema, este trabalho adota uma abordagem que consiste na busca por regras de associação, realizando a mineração na tabela resultante da junção. O processo adotado considera agrupamentos de tuplas, sendo que cada agrupamento é formado pelas tuplas de uma mesma entidade.

Seguindo essa abordagem foi desenvolvido o algoritmo GFP-Growth, o qual é apresentado nesta monografia juntamente com seus resultados e comparações com outros algoritmos multi-relacionais.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1. Introdução

1.1. Motivação

O processo de descoberta de conhecimento consiste na busca por padrões interessantes em bases de dados. A mineração de dados é uma etapa desse processo, em que diferentes técnicas podem ser aplicadas para processar os dados com o intuito de obter os padrões desejados.

As técnicas de mineração tradicionais realizam a busca por padrões em uma única tabela. Quando as informações necessárias para a análise estão separadas em várias tabelas, é necessária a junção das mesmas em uma única tabela para que possam ser aplicadas essas técnicas. O inconveniente, nesse caso, é que essa junção pode resultar em redundância ou perda de informações, não sendo apropriado utilizar as técnicas tradicionais por produzirem um resultado não confiável. Porém, se essas tabelas estão de alguma forma relacionadas, pode ser de interesse a sua análise conjunta.

Um grande esforço tem sido feito para melhoria do processo de mineração de dados envolvendo várias tabelas. O foco se concentra na diminuição dos custos com a operação de junção ou na adoção de uma estratégia que elimine a necessidade dessa operação. Os trabalhos encontrados nessa área realizam mineração de dados em tabelas que estejam de alguma forma relacionadas, utilizando o relacionamento existente entre as tabelas através de chaves estrangeiras (JENSEN; SOPARKAR, 2000) (NESTOROV; JUKIC, 2003) (NG; FU; WANG, 2002) (RIBEIRO, 2004) (RIBEIRO; VIEIRA, 2004) (RIBEIRO; VIEIRA; TRAINA, 2005) (KANODIA, 2005).

1.2. Objetivos

Este trabalho tem como objetivo apresentar uma estratégia para mineração de dados envolvendo várias tabelas que possuem pelo menos uma chave (primária ou estrangeira) em comum. Para isso é apresentada uma estratégia de mineração multi-relacional criada a partir do algoritmo FP-Growth (HAN; PEI; YIN, 2000). Essa abordagem consiste em uma nova versão do algoritmo Connection (RIBEIRO, 2004) (RIBEIRO, 2004) para aplicação sobre uma única tabela, a exemplo do Apriori Group (RIBEIRO; VIEIRA; TRAINA, 2005).

Essa estratégia utiliza a operação de junção entre as tabelas e desconsidera os valores redundantes dos atributos nos cálculos das medidas de interesse, para que não interfiram no

resultado do processo de mineração. Com isso é possível obter, além das regras multi-relacionais, aquelas cujos itens estão presentes em uma única tabela analisada.

Um ponto importante da estratégia adotada é que nenhum dado é descartado no processo de mineração. Para que isso ocorra as várias tabelas envolvidas no processo são relacionadas a partir da operação *FULL OUTER JOIN*. Apesar dessa operação gerar valores nulos, esses valores são tratados e se mostram úteis para contabilizar o quanto um item de uma tabela está relacionado às outras tabelas analisadas.

A estratégia apresentada consiste em uma melhoria do processo de mineração de dados e resulta em uma análise mais abrangente dos mesmos, o que pode gerar padrões adicionais além dos encontrados pelas técnicas tradicionais. A partir dessa estratégia foi desenvolvido o algoritmo GFP-Growth, o qual é apresentado neste trabalho. O GFP-Growth apresenta bons resultados quando comparado ao algoritmo Connection e, portanto, oferece mais uma opção ao usuário de mineração de dados.

1.3. Organização da Dissertação

Esta dissertação está organizada da seguinte maneira:

No capítulo 2 é mostrada a tarefa de associação juntamente com dois importantes algoritmos de associação, que serviram de base para este trabalho;

No capítulo 3 são apresentados conceitos sobre mineração multi-relacional, as diferentes abordagens que tratam esse assunto e alguns trabalhos desenvolvidos nessa área;

No capítulo 4 é apresentada a estratégia desenvolvida para a mineração de dados multi-relacional;

Finalmente, no capítulo 5 são apresentadas as conclusões deste trabalho e propostas para trabalhos futuros.

2. Algoritmos para Mineração de Regras de Associação

2.1. Considerações Iniciais

Segundo Han e Kamber (HAN; KAMBER, 2001), a mineração de dados pode ser vista como resultado da evolução da tecnologia da informação. Esse processo evolutivo envolve desde a criação dos sistemas gerenciadores de bancos de dados (SGBD's) e seus mecanismos para armazenamento e recuperação de dados, até o desenvolvimento de técnicas de análise de dados, incluindo *data warehouses* (CHAUDHURI; DAYAL, 1997) e mineração de dados.

Uma tarefa de mineração de dados define a técnica usada para buscar padrões e o tipo de padrão desejado. A tarefa de associação é uma das mais populares tarefas de mineração de dados. Um dos casos mais conhecidos refere-se à mineração de itens em cestas de compras, sendo possível encontrar itens que tendem a ser comprados em conjunto, o que pode contribuir para a tomada de decisões em relação ao estoque ou promoções desses itens, por exemplo. A seguir são apresentados mais detalhes sobre essa tarefa de mineração, a qual será explorada neste trabalho.

2.2. Mineração de Dados

A mineração de dados consiste na extração de conhecimento a partir de grandes quantidades de dados. A motivação para o desenvolvimento de tarefas de mineração de dados é resultado da grande quantidade de dados disponíveis para análise e da carência de mecanismos para tratamento desses dados.

A mineração de dados é considerada a principal fase do processo de descoberta de conhecimento (KDD), em que técnicas de mineração são aplicadas aos dados para obtenção dos padrões desejados. De acordo com Roden, Burl e Fowlkes (1999 apud RIBEIRO, 2004), a mineração de dados consiste em um campo multidisciplinar, envolvendo conceitos de banco de dados, processamento de imagens, estatística e inteligência artificial.

A seguir será detalhada a tarefa de associação, foco deste trabalho. No apêndice A são fornecidas mais informações sobre mineração de dados e outras tarefas de mineração. Nesse apêndice também são apresentados conceitos relacionados a data warehouses, considerando que alguns trabalhos citados nesta dissertação tratam a mineração de dados em data warehouses.

2.2.1. Diferentes Abordagens Envolvendo Regras de Associação – Breve Histórico

O problema de minerar regras de associação foi abordado pela primeira vez em (AGRAWAL; IMIELINSKI; SWAMI, 1993), e desde então, muita pesquisa tem sido feita com o intuito de resolver o problema da mineração de regras de associação.

Na literatura podem ser encontrados trabalhos que tratam dados armazenados de várias formas, como bancos de dados relacionais (SAVASERE; OMIECINSKI; NAVATHE, 1995) (ZAKI *et al.*, 1997), dados armazenados como programas lógicos (DESHAPE; RAEDT, 1997) e dados armazenados em *data warehouses* (JENSEN; SOPARKAR, 2000) (RIBEIRO, 2004). Além disso, podem ser encontrados trabalhos que tratam da mineração de dados distribuídos (AGRAWAL; EVFIMIEVSK; SRIKANT, 2003) (CHEUNG *et al.*, 1996) e mineração com preservação da privacidade dos dados (AGRAWAL; RAMAKRISHNAN, 2000) (KANTARCIOGLU; CLIFTON, 2002).

Uma análise sobre algoritmos de mineração de regras de associação pode ser encontrada em (IVÁNCSY; KOVÁCS; VAJK, 2004). Trabalhos relativos ao tratamento das regras encontradas podem ser encontrados em (SILBERSCHATZ; TUZHILIN, 1996) e (KLEMETTINEN *et al.*, 1994), sendo que este último apresenta uma forma para melhor visualizar as regras encontradas.

Uma abordagem interessante refere-se à integração da tarefa de associação em SGBD's, o que permitiria muita flexibilidade para realizar consultas nos dados, e é discutido em (IMIELINSKI; MANNILA, 1996) (MEO; PSAILA; CERI, 1996) (SARAWAGI; THOMAS; AGRAWAL, 1998).

Outros trabalhos tratam da integração das tarefas de clusterização com associação, podendo-se citar (ORDONEZ, 2005) e (LENT; SWAMI; WIDOM, 1997).

2.2.2. Tarefa de Associação

Uma tarefa de associação busca por padrões que demonstrem o relacionamento entre conjuntos de itens. Os padrões encontrados são representados na forma de uma implicação $X \rightarrow Y$, onde X e Y representam conjuntos de itens e o padrão encontrado sugere que a presença de X está relacionada à presença de Y. Esse relacionamento pode ser direto, quando a presença de X aumenta a possibilidade da presença de Y, ou inverso, quando a presença de X diminui a possibilidade da presença de Y.

Uma regra de associação pode ser caracterizada como unidimensional, quando os itens envolvidos na mesma derivam de um único atributo, ou multidimensional, quando existem dois ou mais atributos envolvidos na regra. Além disso, pode ser caracterizada quanto aos valores de seus atributos, podendo ser booleana, quando os atributos são categóricos; quantitativa, quando os atributos são numéricos, ou nebulosa, quando os atributos envolvem conceitos nebulosos.

Normalmente, um conjunto de itens é denominado itemset e um itemset com k elementos é chamado k-itemset. As medidas de interesse mais usadas em regras de associação são o suporte e a confiança. O suporte de uma regra $X \rightarrow Y$ indica a fração das tuplas do banco de dados que contêm X e Y e a confiança indica a fração das tuplas que contêm X que também contêm Y. Uma regra que satisfaz o suporte mínimo e a confiança mínima é uma regra forte.

Tabela 2.1 - Exemplo de cesta de compras

IdTransação	Item
100	Pão, leite, manteiga.
200	Pão, requeijão, leite.
300	Manteiga, farinha, leite.
400	Manteiga, pão, refrigerante.
500	Bolacha, leite, manteiga.

Para exemplificar o cálculo dessas medidas de interesse, considere a tabela 2.1. Analisando a regra *manteiga → pão*, nota-se que dentre as 5 transações existentes, pão e manteiga ocorrem em 2 transações. Além disso, dentre as 4 transações em que manteiga ocorre, pão ocorre em 2 dessas transações. Com isso, pode-se dizer que a regra *manteiga → pão* possui suporte de 40% e confiança de 50%, o que demonstra que essa regra pode revelar um padrão de comportamento dos clientes: "Clientes que compram manteiga tendem a comprar pão". Além disso, ela pode ser classificada como unidimensional, pois há apenas o atributo "item" envolvido, e como booleana, pois envolve um atributo categórico.

O cálculo do suporte de um itemset pode ser feito por contagem direta do suporte, através da varredura da base de dados e incremento de um contador para cada transação que contiver o itemset candidato. Outra forma de obter o suporte é através da intersecção de tidlists, que consiste em interseccionar subconjuntos do itemset em questão. Uma tidlist X.tidlist consiste na lista de identificadores das transações que contêm o itemset X, e o suporte de X é dado por |X.tidlist(X)|, que representa o número de elementos na tidlist.

Para encontrar os itemsets freqüentes no espaço de busca, pode ser realizada busca em profundidade ou em largura. Na busca em profundidade, o suporte dos nós é calculado

primeiro na profundidade, enquanto na busca em largura é calculado inicialmente o suporte dos itemsets de tamanho k-1 e posteriormente o suporte dos k-itemsets.

O problema de minerar regras de associação geralmente é descrito em duas etapas: identificação de itemsets freqüentes e geração das regras de associação (HAN; KAMBER, 2001). Na primeira etapa são selecionados os itemsets que possuem suporte maior que um suporte mínimo pré-estabelecido e a segunda etapa utiliza os itemsets da fase anterior para gerar as regras que satisfazem a confiança mínima pré-estabelecida. A fase mais pesada é a primeira e por isso novos algoritmos têm sido desenvolvidos com o objetivo de otimizá-la.

2.3. Algoritmos para Mineração de Regras de Associação

Entre os algoritmos mais conhecidos de mineração de regras de associação, podem ser citados o Apriori (AGRAWAL; SRIKANT, 1994), Partition (SAVASERE; OMIECINSKI; NAVATHE, 1995), FP-Growth (HAN; PEI; YIN, 2000) e Eclat (ZAKI *et al.*, 1997).

Ribeiro (RIBEIRO, 2004), constatou que dentre os algoritmos citados, os que melhor se adequam para o tratamento de múltiplas tabelas são o Apriori e FP-Growth, os quais serão adotados neste trabalho. Isso ocorre porque o algoritmo Apriori é relativamente simples se o compararmos com os demais algoritmos da literatura. Além disso, ele tem sido usado como base para o desenvolvimento de vários algoritmos e por isso já foi amplamente testado. Com relação ao FP-Growth, pode-se dizer que seu ponto forte é o desempenho, pois não há a geração de itemsets candidatos, como no Apriori.

Os algoritmos Apriori e FP-Growth usam diferentes abordagens para identificar os itemsets freqüentes, o que influi diretamente em seu desempenho. Ambos usam contagem direta do suporte para descobrir itemsets freqüentes, no entanto o Apriori usa busca em largura enquanto o FP-Growth usa busca em profundidade.

Conforme citado em Ribeiro (RIBEIRO, 2004), o algoritmo Apriori apresentou um grande avanço em relação à eficiência dos seus antecessores, no entanto ele ainda realiza muitas varreduras no banco de dados. Já o FP-Growth procurou sanar esse problema excluindo a fase de geração de candidatos, o que melhorou seu desempenho.

A seguir são apresentados os algoritmos Apriori e o FP-Growth.

2.3.1. Algoritmo Apriori

O Apriori (AGRAWAL; SRIKANT, 1994) é um algoritmo que minera regras de associação booleanas. É um algoritmo iterativo que busca por k-itemsets freqüentes a partir

dos (k-1)-itemsets. Para isso, é usada a "propriedade Apriori", que se baseia no fato de que um itemset freqüente não pode conter um subconjunto não freqüente.

Seu funcionamento consiste na geração do conjunto C_k, composto por itemsets candidatos, e sua avaliação. Essa avaliação é feita através da contagem do suporte dos itemsets, os quais devem satisfazer um limite mínimo pré-estabelecido. Caso os itemsets satisfaçam esse suporte mínimo, eles são adicionados ao conjunto L_k de itemsets freqüentes.

O algoritmo possui duas fases principais: a fase de junção e a fase de poda. Na primeira, é realizada a operação de junção entre conjuntos L_{k-1} para geração do conjunto C_k. A condição para que a junção ocorra é que os conjuntos L_{k-1} compartilhem (k-1) itens. Na fase de poda, é feita uma varredura no banco de dados para obter o suporte dos itemsets candidatos presentes em C_k. Dessa forma, é possível determinar quais itemsets candidatos são freqüentes e adicioná-los ao conjunto L_k.

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

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

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

Figura 2.1 Algoritmo Apriori (AGRAWAL; SRIKANT, 1994)

Na figura 2.1 é mostrado o algoritmo Apriori. Na linha 1, o conjunto L_1 é inicializado com os 1-itemsets freqüentes de D. Nas linhas 2 a 6, existe um laço no qual é gerado o conjunto L_k. Para cada iteração, o conjunto C_k de k-itemsets candidatos é gerado a partir de L_{k-1} usando a função *Apriori-gen*, o que pode ser visto na linha 3.

Nas linhas 4 a 6, é feita a contagem do suporte de todos os k-itemsets contidos em C_k. Na linha 7, o conjunto L_k é inicializado com todos os k-itemsets presentes em C_k que satisfazem o suporte mínimo. O algoritmo continua no laço buscando os conjuntos L_k, cujos itemsets tenham comprimentos maiores. O algoritmo termina quando não existirem mais itemsets freqüentes, ou seja, quando o conjunto L_k encontrado for vazio. Nesse caso, o algoritmo sai do laço e na linha 8 retorna o conjunto L composto por todos os L_k encontrados.

A função *Apriori-gen* é responsável pela geração de C_k a partir de L_{k-1}. Nas linhas 9 e 10 ocorre a junção dos itemsets de L_{k-1} respeitando a condição de que os mesmos compartilhem (k-1) itens (linhas 11 e 12). Na linha 13 é chamada a função *has_infrequet_subset*, que verifica se um determinado itemset possui um subconjunto de itens que não seja freqüente. Caso possua, esse itemset será excluído do conjunto C_k (linha 14), caso contrário, todos seus subconjuntos são itemsets freqüentes e ele será adicionado a C_k (linha 15). Na linha 17 a função termina e retorna o conjunto C_k gerado.

A função *has_infrequent_subset* garante a aplicação da propriedade Apriori, a qual limita o tamanho do conjunto C_k, ao antecipar a eliminação de alguns itemsets. Seu papel é verificar se existe algum subconjunto de tamanho (k-1) de um k-itemset que não pertence ao conjunto L_{k-1}. Dessa forma, na linha 18 tem início o teste, que é realizado para cada subconjunto do itemset em questão. Na linha 19 é verificado se o subconjunto não pertence ao conjunto L_{k-1}, e em caso afirmativo, a função retorna verdadeiro e finaliza (linha 20), fazendo com que o k-itemset seja eliminado de C_k. Caso contrário, a função continua verificando todos os subconjuntos do itemset e caso todos sejam encontrados em L_{k-1}, a função retorna falso (linha 21) e então finaliza.

A seguir é mostrado um exemplo de execução do algoritmo Apriori, considerando minsup = 2. A tabela 2.2 mostra a base de dados na qual serão buscados os padrões e as figuras 2.2 a 2.5 mostram os conjuntos C_k e L_k dos itemsets candidatos e freqüentes, respectivamente.

Tabela 2.2 - Exemplo de Base de Dados

IdTransação	Itens
100	pão, leite, manteiga
200	pão, requeijão, leite
300	manteiga, pão, refrigerante, bolacha
400	manteiga, farinha, leite
500	bolacha, leite, manteiga, pão

C_1

Itemset	Suporte
Pão	4
Leite	4
Manteiga	4
Requeijão	1
Refrigerante	1
Bolacha	2
Farinha	1

L_1

Itemset	Suporte
Pão	4
Leite	4
Manteiga	4
Bolacha	2

Figura 2.2 Conjuntos de 1-itemsets

C_2

Itemset	Suporte
Pão, leite	3
Pão, manteiga	3
Pão, bolacha	2
Leite, manteiga	3
Leite, bolacha	1
Manteiga, bolacha	2

L_2

Itemset	Suporte
Pão, leite	3
Pão, manteiga	3
Pão, bolacha	2
Leite, manteiga	3
Manteiga, bolacha	2

Figura 2.3 Conjuntos de 2-itemsets

C_3

Itemset	Suporte
Pão, leite, manteiga	2
Pão, leite, bolacha	1
Pão, bolacha, manteiga	2
Leite, manteiga, bolacha	1

L_3

Itemset	Suporte
Pão, leite, manteiga	2
Pão, bolacha, manteiga	2

Figura 2.4 Conjuntos de 3-itemsets

C_4

Itemset	Suporte
Pão, leite, manteiga, bolacha	1

L_4

Itemset	Suporte

Figura 2.5 Conjuntos de 4-itemsets

Na Figura 2.2 pode-se notar que o conjunto C_1 dos 1-itemsets candidatos possui 7 elementos, no entanto 3 desses elementos não satisfazem o suporte mínimo pré-estabelecido, pois aparecem apenas uma vez na base de dados da tabela 2.2. Esses elementos serão descartados, e os demais elementos passarão para o conjunto dos 1-itemsets freqüentes L_1. Esse processo é repetido para os demais k-itemsets, para k>1 conforme mostrado nas figuras 2.3 a 2.5.

Para a geração de conjuntos C_k são feitas combinações entre os elementos de L_{k-1} respeitando a condição de junção do algoritmo. Dessa forma, o conjunto de 2-itemsets candidatos C_2 é gerado a partir de combinações entre os elementos do conjunto de 1-itemsets freqüentes L_1. Esse processo é repetido até que algum conjunto de itemsets freqüentes gerado seja vazio, o que é mostrado na Figura 2.5.

Na tabela 2.3 é mostrado o conjunto de todos os itemsets freqüentes encontrados no exemplo da tabela 2.2, os quais correspondem aos conjuntos L_k das figuras 2.2 a 2.5.

Tabela 2.3 – Itemsets freqüentes encontrados com Apriori

Itemset	Suporte
pão	4
Leite	4
Manteiga	4
bolacha	2
Pão, leite	3
Pão, manteiga	3
Pão, bolacha	2
Leite, manteiga	3
Manteiga, bolacha	2
Pão, leite, manteiga	2
Pão, bolacha, manteiga	2

2.3.2. Algoritmo FP-Growth

O algoritmo FP-Growth (HAN; PEI; YIN, 2000) realiza a busca por regras de associação sem a necessidade de geração de um conjunto de itemsets candidatos, o que faz dele um algoritmo menos custoso que o Apriori.

Durante sua execução, o algoritmo usa uma estrutura de dados chamada *FP-Tree*, a qual armazena de forma compacta informações sobre os itemsets freqüentes, evitando a realização de varreduras constantes no banco de dados. Nessa estrutura, os 1-itemsets compõem os nós da árvore (Figura 2.7) e quanto maior a freqüência de um itemset, maior a possibilidade do mesmo compartilhar nós com vários ramos.

Para examinar a *FP-Tree* é preciso tomar um 1-itemset freqüente como sufixo e posteriormente analisar seu prefixo, que seria sua base de padrão condicional, a qual é formada pelos itemsets que ocorrem junto com esse sufixo. Posteriormente, é criada a *FP-Tree condicional* para esse 1-itemset e a mineração é feita recursivamente nessa estrutura.

A *FP-Tree* é composta por um nó raiz com o rótulo "null", um conjunto de nós com os itemsets freqüentes e uma tabela header, que facilita o acesso à árvore. Cada nó da *FP-Tree* consiste de 3 campos: o item representado (*item-name*), um contador com o número de vezes que o item aparece em transações representadas pelo ramo em que se encontra (*count*) e um ponteiro que aponta para o próximo nó cujo item-name seja o mesmo (*node-link*). Já na tabela header, cada entrada é composta por um item e por um ponteiro que aponta para o primeiro nó que armazena o item em questão.

Algoritmo: Construção da FP-Tree
Entrada: Banco de Dados *D*, suporte mínimo *minsup*
Saída: FP-Tree T

Função FPTree_Gen(D, minsup) {
1. Percorra o banco de dados D
2. Determine o conjunto de itemsets frequentes F e seus suportes
3. Associe a L o conjunto F em ordem decrescente de suporte
4. Crie a raiz da FP-Tree T e associe o valor "null" ao nó
5. Para cada transação Trans de D faça {
6. Selecione e ordene os itens de acordo com L
7. Chame o conjunto ordenado de [p|P], onde p é o primeiro elemento e P o restante do conjunto.
8. insere_tree([p|P], T)
9. }
10. retorne T
11. }

Função insere-tree([p|P], T) {
12. Se T tem um filho N tal que N.item-name=p.item-name
13. Então N.contador ++
14. Se não {
15. Cria-se um novo nó N com N.contador = 1, N.pai = T.
16. Associe N.node-link com os nós cujo item-name seja igual a N.item-name
17. }
18. Se P ≠ Ø
19. Então insere-tree(P,N) }

Figura 2.6 Construção da FP-Tree (HAN; PEI; YIN, 2000)

Na Figura 2.6 é mostrado o algoritmo para a geração da *FP-Tree*. Inicialmente o banco de dados é varrido e o conjunto dos itemsets freqüentes F é determinado, juntamente com seu suporte (passos 1 e 2). No passo 3 o conjunto L de itemsets freqüentes é criado com os itemsets em ordem decrescente de suporte. No passo 4 é criado o nó raiz da *FP-Tree* e associado o rótulo "null" ao mesmo. No passo 5 existe um laço onde para cada transação do banco de dados seus itens são ordenados de acordo com L (passo 6) e passados como parâmetro para o procedimento *insere-tree* (passo 7 e 8).

A função *insere-tree* verifica se já existe um nó com o item em questão (passo 12) e caso exista, incrementa seu contador (passo 13). Caso contrário, um novo nó é criado e adicionado à árvore (passos 14 a 17). Caso ainda existam itens em P, a função *insere-tree* é chamada recursivamente (passo 19).

Para a criação da *FP-Tree* são necessárias duas varreduras no banco de dados. A primeira gera a lista L de itens freqüentes, a qual é ordenada em ordem decrescente de suporte, e orienta a inserção dos nós na árvore. Na segunda varredura, cada transação é ordenada de acordo com L para que seus itens sejam inseridos na árvore nessa ordem. O objetivo dessa ordenação é fazer com que os itens mais freqüentes ocupem níveis mais altos na árvore, para que possam compartilhar nós com vários ramos. Na Figura 2.7 é mostrada a *FP-Tree* para a base de dados da tabela 2.2, considerando *minsup = 2*.

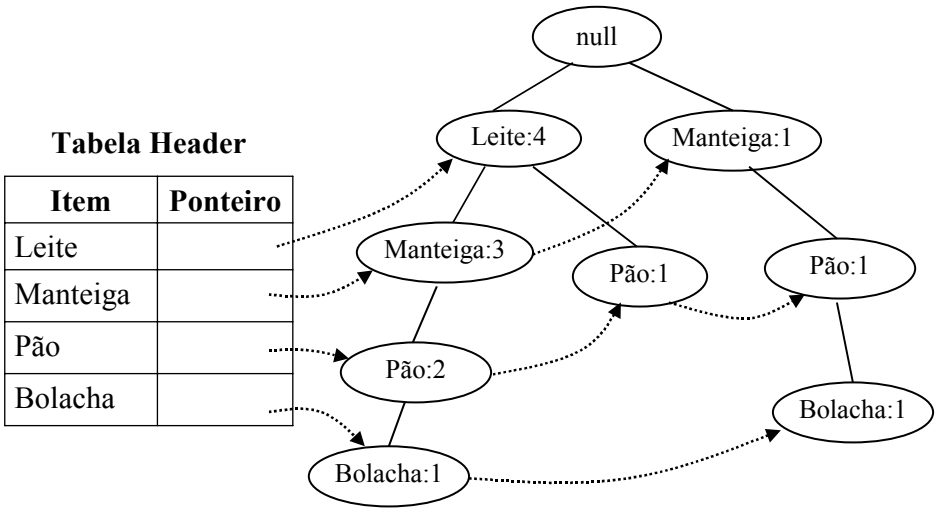

Figura 2.7 Exemplo de FP-Tree

No exemplo da Figura 2.7, o nó "pão" está presente em 3 ramos e para calcular os padrões freqüentes do qual faz parte, basta analisar os prefixos desse nó. Nesse caso, os prefixos do nó "pão" são {(leite:1), (leite, manteiga:2) (manteiga:1)}, e representam os

caminhos que levam ao nó “pão”, o que é chamado de base condicional do nó. O contador nesse caso assume o valor do nó “pão” no ramo especificado.

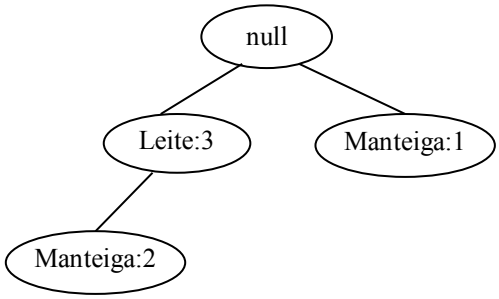

Figura 2.8 Exemplo de FP-Tree condicional

A base condicional do nó “pão” guarda informações sobre os padrões que ocorrem juntamente com o nó especificado. Essa base pode ser usada para criar uma estrutura chamada *FP-Tree condicional*, a qual armazena apenas os caminhos freqüentes que levam ao nó em questão. A Figura 2.8 mostra a *FP-Tree condicional* criada a partir da base condicional do nó “pão”.

Na Figura 2.9 é mostrado o algoritmo FP-Growth. No passo 1 verifica-se se a *FP-Tree* possui um único ramo. Caso possua, os nós desse ramo são combinados e são gerados padrões cujo suporte é igual ao mínimo suporte dos nós envolvidos na combinação (passos 2 e 3). Caso contrário, é construída a base e a *FP-Tree condicional* de cada itemset freqüente (passos 4 a 6) e caso essa *FP-Tree condicional* gerada não seja vazia, o método FP-Growth é chamado recursivamente para essa árvore e o item freqüente correspondente.

Algoritmo: FP-Growth
Entrada: FP-Tree T
Saída: Conjunto de itemsets frequentes
Método: Chamar FPGrowth(FP-Tree, null)

Função FPGrowth (Tree, α) {
1. Se Tree contém um único ramo *P*
2. Então para cada combinação β dos nós em *P* faça
3. Gere o padrão β∪α com suporte = mínimo suporte dos nós em β
4. Se não para cada a_i na tabela header de *Tree* faça{
5. Gere o padrão β= a_i∪α com suporte = a_i.suporte
6. Construa a base e a FP-Tree condicional $Tree_\beta$ de β
7. Se $Tree_\beta \neq \emptyset$
8. Então FPGrowth($Tree_\beta$, β)
9. }
10. }

Figura 2.9 Algoritmo FP-Growth (HAN; PEI; YIN, 2000)

Para obter os padrões freqüentes da base de dados da tabela 2.2, a *FP-Tree* da Figura 2.7 é passada como parâmetro para o algoritmo FP-Growth. O algoritmo consiste inicialmente na identificação de um 1-itemset e na construção de sua base e *FP-Tree condicional*. Posteriormente a mineração é realizada recursivamente na *FP-Tree condicional* gerada. Na Figura 2.10 são mostrados os padrões e estruturas gerados durante a execução do algoritmo FP-Growth para a mineração do itemset "pão".

Na Figura 2.10 pode-se notar que a partir do 1-itemset "pão" são gerados 3 novos padrões freqüentes: "manteiga, pão", "leite, manteiga, pão" e "leite, pão". A mineração do itemset "pão" tem início no passo 4 do algoritmo da Figura 2.9, já que a *FP-Tree* da Figura 2.7 não possui um único ramo. Inicialmente é encontrado o padrão "pão", e sua *FP-Tree* condicional é criada. Como ela possui mais de um ramo, o algoritmo retorna ao passo 4, para realizar a mineração de cada item da árvore.

Nesse ponto, nota-se na Figura 2.10 que é gerado o itemset "leite, pão", da combinação do item "leite" da tabela header com o padrão "pão" encontrado anteriormente. A *FP-Tree condicional* $\text{Tree}_{\text{leite, pão}}$ gerada é vazia, o que faz com que o algoritmo retorne para minerar o próximo item da tabela header de $\text{Tree}_{\text{pão}}$.

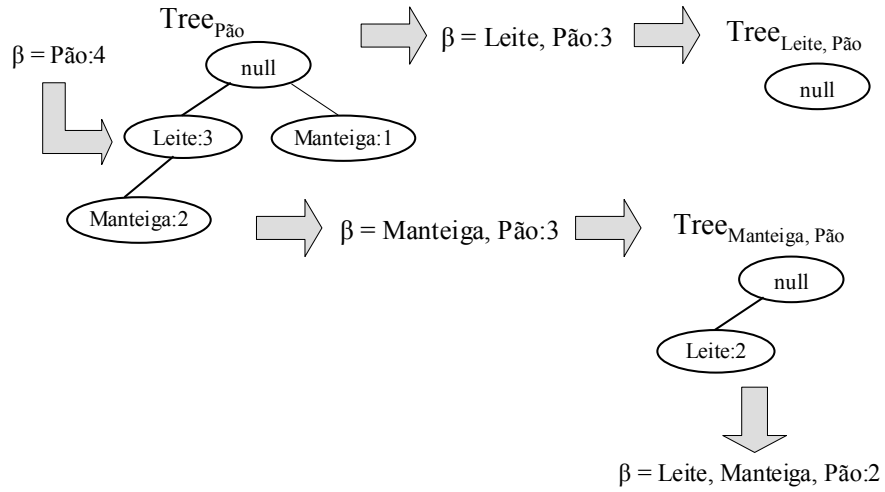

Figura 2.10 Estruturas e padrões gerados pelo itemset "pão"

O próximo passo é minerar o item "manteiga", o que gera inicialmente o itemset "manteiga, pão". Sua *FP-Tree condicional* $\text{Tree}_{\text{manteiga, pão}}$ é gerada e possui apenas um ramo. Dessa árvore pode-se obter o padrão "leite, manteiga, pão" e a mineração do itemset "pão" finaliza.

Tabela 2.4 - Bases e *FP-Tree* condicionais geradas

Item	Base de padrão condicional	*FP-Tree condicional*
Bolacha	{(leite, manteiga, pão:1), (manteiga, pão:1)}	{(manteiga, pão:2)}
Pão	{(leite, manteiga:2), (leite:1), (manteiga:1)}	{(leite:3, manteiga:2), (manteiga:1)}
Manteiga	{(leite:3)}	{(leite:3)}
Leite	Ø	Ø

Tabela 2.5 - Padrões freqüentes gerados

Item	Padrões Freqüentes
Bolacha	(bolacha, manteiga):2, (bolacha, pão):2, (bolacha, pão, manteiga):2
Pão	(pão):4, (leite, pão):3, (manteiga, pão):3, (leite, manteiga, pão):2
Manteiga	(manteiga):4, (leite, manteiga):3
Leite	(leite):4

Na tabela 2.4 são mostradas as bases condicionais e as *FP-Tree condicionais* geradas pelo algoritmo para cada item. Na tabela 2.5 são mostrados os padrões freqüentes gerados pelo algoritmo, juntamente com o item que o gerou.

2.4. Considerações Finais

Neste capítulo foram apresentados os algoritmos Apriori e FP-Growth, que realizam a mineração de regras de associação. O Apriori é um dos algoritmos mais conhecidos e constituiu a base para desenvolvimento de vários algoritmos disponíveis atualmente. Já o FP-Growth adota uma estratégia diferente com o objetivo de solucionar os problemas de desempenho do Apriori, o que faz dele um dos algoritmos de melhor desempenho. Esses dois algoritmos foram importantes para o desenvolvimento deste trabalho, pois serviram de base para resolução do problema proposto, conforme é apresentado no capítulo 4.

3. Mineração Multi-Relacional

3.1. Considerações Iniciais

A mineração multi-relacional consiste na busca por padrões em múltiplas tabelas, ao contrário das técnicas de mineração tradicionais, que consideram uma única tabela. Trata-se de um processo mais custoso que a mineração tradicional e por isso, muitos trabalhos têm sido desenvolvidos nessa área com o intuito de propor melhorias.

Neste capítulo são apresentadas diferentes abordagens utilizadas para tratar o problema da mineração multi-relacional, as quais podem envolver conceitos de banco de dados, inteligência artificial e lógica indutiva.

3.2. Definição de Mineração Multi-Relacional

A mineração multi-relacional, também citada como mineração relacional em (DžEROSKI, 2003) (DžEROSKI; ŽENKO, 2002), tem por objetivo analisar múltiplas tabelas em conjunto. Recentemente, os tipos de padrões e técnicas de mineração de dados tradicionais têm sido estendidos para o caso da mineração multi-relacional, sendo aplicados satisfatoriamente em várias áreas, principalmente em bioinformática (DžEROSKI, 2003).

Segundo Džeroski e Raedt (DžEROSKI; RAEDT, 2002), a mineração multi-relacional constitui-se uma área multidisciplinar envolvendo lógica indutiva, aprendizado de máquina, KDD e bancos de dados relacionais. O foco deste trabalho concentra-se nas áreas de KDD e bancos de dados.

A mineração multi-relacional tem sido aplicada em áreas cujos dados são estruturados e existe um conhecimento prévio do domínio. Normalmente, são tarefas que seriam mais complexas de se realizar usando as técnicas tradicionais de mineração. Entre as áreas de aplicação pode-se incluir: análise de dados de negócios, engenharia de tráfego e ambiental, mineração da web e principalmente bioinformática, no projeto de drogas e genoma (DžEROSKI, 2003).

Uma outra possibilidade de aplicação de técnicas de mineração multi-relacional é citada em (KNOBBE; SIEBES; WALLEN, 1999), e refere-se ao tratamento de objetos complexos ou estruturados, que precisam ser representados por múltiplas tabelas. Os autores ainda reforçam que as técnicas tradicionais de mineração de dados permitem a análise apenas de

objetos simples, os quais são representados por um número fixo de atributos monovalorados e, portanto não consideram a estrutura dos objetos.

Segundo Domingos (DOMINGOS, 2003), o campo de mineração multi-relacional ficou durante um bom tempo estagnado devido à limitada escalabilidade dos algoritmos tradicionais, à sua inabilidade para expressar ruídos e incertezas e pela falta de aplicativos robustos. Com a melhoria desses campos, a mineração multi-relacional teve um grande crescimento, mas ainda há muito a ser feito, como a melhoria da interface entre as tarefas de mineração multi-relacional e os bancos de dados relacionais e a extensão de técnicas usadas com sucesso nos algoritmos tradicionais para o modo relacional. Džeroski (DžEROSKI, 2003) afirma que essa última poderia ser realizada reformulando apenas as medidas chave do algoritmo, mantendo o restante do mesmo inalterado.

A complexidade dos algoritmos de mineração multi-relacional é maior que a dos algoritmos tradicionais, o espaço de busca das hipóteses aumenta e a avaliação das mesmas se torna mais complexa. Na terminologia de bancos de dados, a avaliação de uma hipótese deve envolver junções entre tabelas, o que não é o caso dos métodos tradicionais de mineração de dados.

Conforme citam Blockeel e Sebag (BLOCKEEL; SEBAG, 2003), existem métodos que melhoram a eficiência em detrimento da corretude dos dados, e existem métodos que preservam a corretude. Um algoritmo pode não preservar a corretude, e mesmo assim produzir um resultado que tenha uma probabilidade de ser similar ao resultado correto. Nesse caso, a probabilidade e a similaridade são definidas como parâmetros do algoritmo. Outra otimização possível refere-se ao pré-processamento e materialização dos dados que serão utilizados. A transformação do problema para a forma de uma única tabela pode melhorar a eficiência do processo, mas limita a expressividade dos padrões encontrados.

Devido à grande quantidade de padrões que podem ser encontrados, torna-se necessária a adoção de alguns procedimentos para limitar esse número. Uma possibilidade é a restrição da análise a uma parcela da base de dados, minerando apenas as tabelas mais importantes para o processo, além da especificação das chaves que ligam essas tabelas ou do tipo de padrão desejado.

Existem diferentes estratégias para o tratamento da mineração multi-relacional. Uma abordagem bem conhecida envolve a área de programação lógica indutiva, a qual se preocupa com a busca de padrões que são expressos como programas lógicos. Džeroski (DžEROSKI, 2003) situa essa área na intersecção dos campos de aprendizado de máquina e programação lógica.

Knobbe, Siebes e Wallen (KNOBBE; SIEBES; WALLEN, 1999) enfatizam que a análise feita pelas técnicas de programação lógica indutiva considera apenas dados armazenados em programas PROLOG, não levando em conta a modelagem relacional dos dados, que poderia ajudar no processo de busca. Os autores ainda ressaltam a possibilidade de um ganho em eficiência quando se utiliza o conhecimento do domínio presente na modelagem dos dados. Segundo Yin, Han e Yang (YIN; HAN; YANG, 2005), uma desvantagem dessa abordagem é que a mesma não é escalável quanto ao número de relações ou atributos envolvidos no processo, o que pode trazer problemas com relação ao desempenho.

Na literatura podem ser encontrados vários trabalhos que abordam essa área (DEHASPE; TOIVONEN, 1999) (DEHASPE; TOIVONEN, 2001) (DžEROSKI, 2003), no entanto, técnicas de lógica indutiva não serão tratadas neste trabalho. A estratégia a ser adotada envolve a área de descoberta de conhecimento em bancos de dados relacionais. Na próxima seção são apresentados alguns trabalhos de mineração multi-relacional cujas estratégias são semelhantes às adotadas neste trabalho.

3.3. Motivação para Realização da Mineração Multi-Relacional

Para realizar mineração multi-relacional usando técnicas de mineração tradicionais, é necessária a realização de junções nos dados antes dos algoritmos serem aplicados, o que pode resultar em redundância e inconsistências (RIBEIRO, 2004).

Esse processo exige grande esforço e conhecimento (DžEROSKI; RAEDT, 2002) (DžEROSKI; RAEDT; WROBEL, 2003) e pode resultar em perda de informações dos dados e de seu significado (DžEROSKI, 2003). Um outro problema desse processo refere-se ao grande tamanho da tabela resultante da junção e às repetições de valores encontrados na mesma, o que pode afetar o desempenho da mineração (JENSEN; SOPARKAR, 2000) ou até mesmo inviabilizar o processo (NG; FU; WANG, 2002).

Para ilustrar esses problemas, será utilizada a modelagem E-R de um sistema bancário mostrada na Figura 3.1. Aqui se deseja analisar dois assuntos: a movimentação bancária dos clientes e o contrato de serviços pelos mesmos. Os relacionamentos *Contrata* e *Movimenta* guardam os valores e datas das movimentações bancárias e contratos de serviços realizados pelos clientes. Como esses dois relacionamentos são M:N, duas novas tabelas serão geradas para armazenar seus valores.

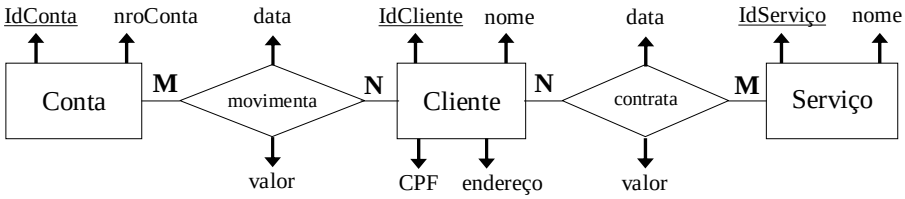

Figura 3.1 Modelagem referente a um sistema bancário

Nesse caso, pode ser interessante analisar a relação existente entre a movimentação bancária de determinado cliente e os serviços contratados pelo mesmo. Como os algoritmos tradicionais de mineração foram concebidos para processar somente uma tabela, para realizar essa análise usando as técnicas tradicionais de mineração seria necessário efetuar a junção das tabelas referentes aos relacionamentos *Movimenta* e *Contrata*, para então aplicar um desses algoritmos. Essa junção é realizada através do atributo *IdCliente*, que se refere à chave compartilhada pelas tabelas.

Na figura 3.2 é mostrada uma parcela dos dados das tabelas *Movimenta* e *Contrata*, e a tabela resultante da junção dessas duas tabelas através do atributo *IdCliente*.

Movimenta

IdConta	IdCliente	Valor
Ct1	Cl1	10000..15000
Ct2	Cl2	5000..10000
Ct3	Cl2	10000..15000
Ct4	Cl3	15000..20000
Ct5	Cl3	10000..15000

Contrata

IdServiço	IdCliente	Valor
Sr1	Cl1	200..500
Sr2	Cl1	100..200
Sr1	Cl2	500..1000
Sr2	Cl2	200..500
Sr3	Cl2	100..200
Sr3	Cl3	200..500
Sr2	Cl4	100..200

Movimenta X Contrata

IdCliente	IdConta	IdServiço	ValorMovimento	ValorServiço
Cl1	Ct1	Sr1	10000..15000	200..500
Cl1	Ct1	Sr2	10000..15000	100..200
Cl2	Ct2	Sr1	5000..10000	500..1000
Cl2	Ct2	Sr2	5000..10000	200..500
Cl2	Ct2	Sr3	5000..10000	100..200
Cl2	Ct3	Sr1	10000..15000	500..1000
Cl2	Ct3	Sr2	10000..15000	200..500
Cl2	Ct3	Sr3	10000..15000	100..200
Cl3	Ct4	Sr3	15000..20000	200..500
Cl3	Ct5	Sr3	10000..15000	200..500

Figura 3.2 Relacionamentos e sua junção

Analisando a figura 3.2, nota-se que a junção resultou em um grande número de valores duplicados. Isso pode ser visualizado ao considerar o itemset *{IdConta = "Ct1", IdCliente =*

"Cl1"} que ocorre apenas uma vez na tabela *Movimenta*, mas na tabela resultante da junção ele ocorre duas vezes. Analisando a tabela *Contrata*, nota-se que esse fato também ocorre. Um exemplo é o itemset *{IdServiço= "Sr2", IdCliente= "Cl2"}*, que ocorre apenas uma vez na tabela Contrata, mas duas vezes na tabela resultante da junção.

Além disso, o valor do suporte dos itemsets nessa nova tabela é diferente do valor encontrado nas tabelas originais, o que pode levar a inconsistências na geração das regras. Um exemplo ocorre com o itemset *{IdConta= "Ct2", IdCliente= "Cl2"},* cujo suporte é igual a 20% na tabela *Movimenta*, mas na tabela resultante da junção esse valor muda para 30%. Considerando agora a tabela *Contrata*, nota-se que o mesmo problema ocorre. Um exemplo é o itemset *{IdServiço= "Sr3", IdCliente= "Cl2"}*, cujo suporte é 14% na tabela *Contrata*, mas na tabela resultante da junção o suporte passou a ser 20%.

Com esse exemplo, pode-se notar que a junção das duas tabelas pode resultar na geração de padrões inconsistentes ou impedir a descoberta de padrões válidos. Isso ocorre porque a movimentação bancária dos clientes e seus contratos de serviços não têm relação direta entre si, ou seja, a ocorrência de movimentações bancárias não está diretamente atrelada aos contratos de serviços realizados. Porém, pode haver um padrão de comportamento de clientes relativo aos seus movimentos bancários e seus contratos de serviços realizados. Por essa razão, é interessante a utilização de técnicas de mineração que consigam extrair esses padrões.

3.4. Estratégias para Realizar Mineração Multi-Relacional

As técnicas de mineração multi-relacional têm como objetivo buscar soluções para os problemas citados na seção anterior, não atendidos pelo processo de mineração tradicional. Para isso, cada abordagem propõe uma nova forma de relacionar as várias tabelas envolvidas, evitando a perda de significado ou inconsistências.

Em (KANODIA, 2005) e (NG; FU; WANG, 2002) pode-se observar o uso de estruturas de dados, como árvores e vetores, para armazenar dados temporários das várias tabelas envolvidas, e posteriormente realizar a busca por conhecimento nessas estruturas. Já em (YIN; HAN; YANG, 2005) foi utilizada a propagação de tuplas, realizando uma operação chamada pelos autores de junção virtual.

(JENSEN; SOPARKAR, 2000) e (RIBEIRO, 2004) tratam a mineração multi-relacional em um *data warehouse*, aproveitando a etapa de pré-processamento pela qual os dados passaram. As abordagens encontradas em (JENSEN; SOPARKAR, 2000) e (RIBEIRO;

VIEIRA; TRAINA, 2005) realizam a junção das tabelas envolvidas e tratam os problemas que essa operação pode ocasionar; já (NESTOROV; JUKIC, 2003) usa SQL para relacionar as tabelas e restringir o espaço de busca.

As estratégias abordadas por esses vários autores são descritas nas próximas seções. Outros trabalhos que não abordam a tarefa de associação, mas tratam múltiplas tabelas podem ser encontrados em (DING; PARIKH, 2004) (ATRAMENTOV; LEIVA; HONAVAR, 2003). Além disso, outros trabalhos que envolvem a busca por regras de associação multi-relacionais, cuja abordagem não segue necessariamente a adotada por este trabalho, podem ser encontrados em (CLARE; WILLIAMS; LESTER, 2004) (DESHAPE; RAEDT, 1997).

3.4.1. Descoberta de Regras de Associação em DW's usando SQL

Nestorov e Jukic (NESTOROV; JUKIC, 2003) apresentam um framework para mineração de regras de associação em DW que leva em consideração o relacionamento entre as dimensões e a tabela fato, o que permite a descoberta de regras de associação não encontradas pelas técnicas tradicionais de mineração, as quais analisam apenas a tabela fato.

Para exemplificar essa situação, considere o DW da Figura 3.3. Usando as técnicas tradicionais de mineração de dados, seria possível encontrar os serviços que são adquiridos freqüentemente em conjunto. Nesse caso, o relacionamento da tabela fato com suas dimensões não é considerado, e a mineração é realizada para todos os clientes em qualquer data.

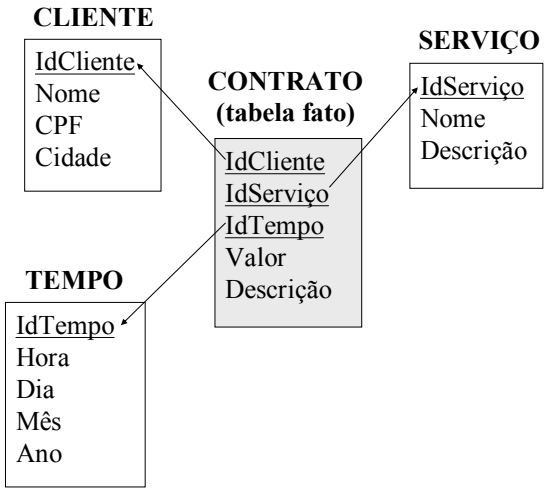

Figura 3.3 – Contrato de serviços em um sistema bancário

No entanto, regras adicionais podem ser encontradas caso o espaço de busca seja limitado através da especificação de determinados valores para as dimensões do DW. Nesse

caso, no DW da Figura 3.3 seria possível minerar uma regra que mostrasse os serviços adquiridos em conjunto para clientes de uma mesma cidade em um determinado ano.

Uma extensão da notação das regras de associação é necessária para representação do novo modelo de regra proposto. Nesse caso, X→Y (Z) representa a regra estendida e pode ser lida da seguinte forma: as transações que satisfazem as restrições de Z e contêm X, tendem a conter Y. Um exemplo de padrão nessa representação seria: *seguro de vida → plano previdência privada (cidade=São Carlos, ano=2004)*, a qual mostra que clientes da cidade de São Carlos que adquiriram seguro de vida em 2004, também adquiriram um plano de previdência privada.

Dessa forma, os padrões encontrados são de granularidade mais fina, pois especificam grupos de clientes de uma cidade em um determinado ano, ao contrário das técnicas tradicionais, que buscam padrões de granularidade mais grossa.

A mineração das regras é feita através de consultas SQL que implementam as regras de associação estendidas, o que é realizado pelo próprio SGBD utilizado. São usadas tabelas temporárias para armazenar resultados intermediários, o que auxilia na otimização do processo. Durante a mineração das regras, são necessárias junções entre as tabelas, o que consiste em uma operação de alto custo.

Como as tabelas de um DW ocupam um grande espaço, não é desejável que ocorram operações de junção que envolvam todos os registros dessas tabelas, especialmente quando se trata de uma tabela fato. No entanto, essa situação muitas vezes é necessária, como no caso em que se deseja buscar itens que ocorrem freqüentemente juntos, e para isso torna-se necessário realizar a junção da tabela fato com ela mesma.

O processo de otimização proposto consiste em elaborar uma seqüência de consultas SQL, de forma que uma consulta retorne apenas dados relevantes para a próxima consulta. Dessa forma, transações que não satisfazem a regra são excluídas já no início do processo, o que resulta em tabelas temporárias menores e conseqüentemente, evita junções de tabelas muito grandes. No exemplo dos itens que ocorrem juntos em transações de uma tabela fato, seria necessária a junção de uma tabela temporária com a tabela fato original, o que reduziria enormemente o custo da operação.

Um outro fator considerado na otimização é a ordem em que as consultas aparecem na seqüência. É preciso levar em conta o fator de redução do espaço de busca que a consulta proporciona e a quantidade de junções que a mesma realiza. A partir daí é estruturada a seqüência, de forma que a última consulta retorne os padrões que satisfazem a regra de associação desejada.

3.4.2. Descoberta de Regras de Associação em DW's de Forma Distribuída

Ng, Fu e Wang (NG; FU; WANG, 2002) apresentam uma nova forma de minerar regras de associação em DW's modelados de acordo com o esquema estrela. A inovação consiste no fato de que não é necessário realizar a operação de junção para obter informações a respeito do relacionamento entre a tabela fato e suas dimensões, o que diminui o custo do processo.

O processo é dividido em duas fases: na primeira é usado um algoritmo existente para obter os itemsets freqüentes das dimensões envolvidas e na segunda fase, verificam-se quais combinações dos itemsets freqüentes encontrados anteriormente são freqüentes em duas ou mais dimensões. Dessa forma, o processo consiste inicialmente na busca por itemsets freqüentes locais e posteriormente, na combinação dos mesmos para obtenção dos itemsets freqüentes globais.

O método apresentado trata valores categóricos e se propõe a minerar regras na junção da tabela fato com suas dimensões, sem de fato realizar essa junção. É constituído por três etapas: pré-processamento, mineração local e global.

1. **Pré-processamento**: as dimensões são varridas e transformadas para o formato binário, conforme mostrado na figura 3.4. Para cada item freqüente da tabela, é criada uma coluna na nova tabela e os valores 1 e 0 representam se o item pertence ou não à transação, respectivamente. Existe uma ordem pré-estabelecida entre os atributos dessa nova tabela, o que facilita a implementação do algoritmo. Além disso, é criado um vetor com as freqüências em que as transações da dimensão aparecem na tabela fato.
2. **Mineração local**: os itemsets freqüentes de cada tabela dimensão são descobertos usando um algoritmo de mineração tradicional com a medida do suporte alterada para ser contada na tabela fato.
3. **Mineração global**: subdividida em três fases.
    3.1. Varredura da tabela fato e armazenamento das informações em estruturas de dados.
    3.2. Mineração de 2-itemsets para duas tabelas dimensão.
    3.3. Mineração de k-itemsets para k=3,4...

Posteriormente a freqüência é contada e a partir dos candidatos gerados são obtidos os itemsets freqüentes. Os passos 3.2 e 3.3 são repetidos para demais dimensões, e constituem a fase de ligação do processo.

	V_1	V_2	V_3
a_1	R_2	R_0	R_1
a_2	R_0	R_0	R_1
a_3	R_1	R_1	R_0

	$V_1=R_0$	$V_1=R_1$	$V_1=R_2$	$V_2=R_0$	$V_2=R_1$	$V_3=R_0$	$V_3=R_1$
	x_1	x_2	x_3	x_4	x_5	x_6	x_7
a_1	0	0	1	1	0	0	1
a_2	1	0	0	1	0	0	1
a_3	0	1	0	0	1	1	0

Figura 3.4 - Exemplo de conversão de tabela dimensão para modo binário (NG; FU; WANG, 2002)

Considerando a figura 3.3, para calcular a freqüência de um itemset X que contém itens das tabelas Cliente e Serviço, é necessário seguir os seguintes passos: determinar as transações T_1 da tabela Cliente que contêm itens de X, determinar as transações T_2 da tabela Serviço que aparecem junto com transações de T_1 na tabela Contrato, determinar o conjunto de transações T_3 da tabela Serviço que contêm itens de X. A freqüência de X é obtida da intersecção de T_2 e T_3.

A

Tid	Itens
a_1	x_1, x_3, x_5
a_2	x_2, x_3, x_6
a_3	x_1, x_3, x_6
a_4	x_1, x_4, x_6

FT

Tid(A)	Tid(B)
a_1	b_5
a_1	b_3
a_1	b_2
a_3	b_2
a_3	b_5
a_1	b_5

B

Tid	Itens
b_1	y_1, y_3, y_5
b_2	y_1, y_3, y_6
b_3	y_2, y_4, y_6
b_4	y_1, y_4, y_5
b_5	y_1, y_4, y_6

Figura 3.5 - Exemplo de esquema estrela (NG; FU; WANG, 2002)

Para isso é usada uma estrutura chamada tid_list, que consiste em uma forma compacta de armazenar os identificadores das transações. Dessa forma, são definidas as seguintes notações:

- $tid_A(x_i)$: lista das transações da tabela A que contém x_i, da forma tid(count).
- $tid_A(X)$: similar ao anterior, exceto pelo fato de X ser um itemset.
- B_key(a_n): lista das transações da tabela B na forma tid(count) que possuem itens juntamente com a_n na tabela fato.
- B_tid(x_i): lista das transações de B que ocorrem junto na tabela fato com transações que contém x_i em outra dimensão.
- B_tid(X): similar ao anterior, exceto pelo fato de que X é um itemset.

Na figura 3.5 é mostrado um exemplo de esquema estrela. Considere para este exemplo *minsup = 5*. Para minerar um itemset freqüente, é preciso inicialmente encontrar os itemsets freqüentes da tabela A e B separadamente. Nesse caso, x_1x_3 é freqüente, pois ocorrem juntos em a_1 e a_3, e essas duas transações aparecem 6 vezes na tabela fato. No próximo passo, analisam-se quais combinações podem ser feitas com transações de B para gerar itemsets freqüentes. O itemset y_1y_6 é freqüente em B com freqüência 5. Para verificar se $x_1x_3y_1y_6$ é freqüente são necessários os seguintes passos:

1. $tid_A(x_1) = \{a_1(4), a_3(2)\}$, $tid_A(x_3) = \{a_1(4), a_3(2)\}$, $tid_A(x_1\ x_3) = tid_A(x_1) \cap tid_A(x_3) = \{a_1(4), a_3(2)\}$, $tid_B(y_1y_6) = \{b_2(2), b_5(3)\}$.
2. $B\_key(a_1) = \{b_2(1), b_3(1), b_5(2)\}$, $B\_key(a_3) = \{b_2(1), b_5(1)\}$.
3. $B\_tid(x_1x_3) = B\_key(a_1) \cup B\_key(a_3) = \{b_2(2), b_3(1), b_5(3)\}$.
4. $B\_tid(x_1x_3) \cap tid_B(y_1y_6) = \{b_2(2), b_5(3)\}$.
5. Freqüência total após combinação = $\{b_2(2), b_5(3)\} = 5$. Então, $x_1x_3y_1y_6$ é freqüente.

Quando existem mais de duas tabelas dimensão, a ligação é realizada para duas tabelas de cada vez e o resultado dessa operação é considerado uma nova tabela, a qual é ligada à terceira tabela e assim sucessivamente. Um fato a ser destacado é que essas novas tabelas são na realidade armazenadas como árvores de prefixos. Quando a ligação entre duas dimensões é realizada, dois níveis da árvore são fundidos e os nós desse novo nível recebem novos rótulos com os novos identificadores de transações. Na figura 3.6 é mostrada a árvore de prefixos referente ao DW da figura 3.5.

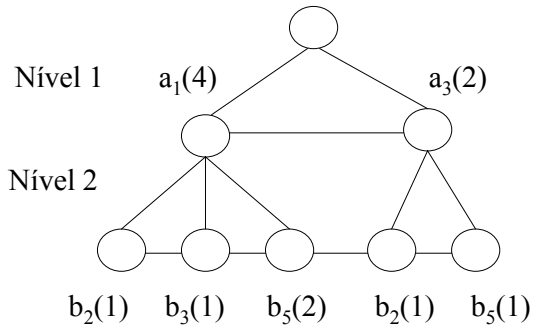

Figura 3.6 - Árvore de prefixos

No início do processo é necessária uma varredura da tabela A para obter $tid_A(x_1)$ para cada x_1. Em uma outra varredura é possível obter os contadores das transações das dimensões juntamente com $B\_key(a_1)$ para cada a_1 de cada dimensão. Isso é feito no passo 3.1, e as

informações sobre as transações são guardadas em uma árvore de prefixos, cujos nós são da forma tid(count) e o número de níveis representa o número de dimensões envolvidas.

Essa árvore representa uma forma compacta de armazenar informações da tabela fato e pode ser usada para realizar as operações necessárias sobre os dados. Por exemplo, para calcular B_key(a_1) basta procurar seus filhos na árvore, o que resulta em {$b_2(1)$, $b_3(1)$, $b_5(2)$}. Além disso, os itemsets freqüentes de mesmo tamanho são guardados em *FI-tree's (Frequent Itemset)* separadas, para que possam ser utilizados na geração de itemsets maiores.

3.4.3. Contagem de Itemsets Freqüentes em DW's

Jensen e Soparkar (JENSEN; SOPARKAR, 2000) apresentam uma técnica de mineração de dados envolvendo uma tabela fato e várias dimensões de um DW, através do uso das chaves estrangeiras que relacionam essas tabelas. Esse modelo pode igualmente ser usado no modelo entidade relacionamento, considerando que as dimensões referem-se às entidades e a tabela fato refere-se ao relacionamento.

A proposta consiste na mineração descentralizada de cada tabela envolvida e na posterior fusão dos resultados. Para isso são necessárias a alteração dos algoritmos disponíveis e análise de desempenho do processo. Com isso, é possível obter os mesmos padrões que seriam encontrados caso os algoritmos tradicionais fossem aplicados na tabela resultante da junção das tabelas envolvidas.

O modelo proposto é formado por duas etapas: busca por itemsets freqüentes em tabelas separadas e fusão dos resultados obtidos através do relacionamento de chaves estrangeiras existente entre as tabelas.

Para que um itemset seja considerado freqüente, todos os seus subconjuntos devem ser freqüentes. Dessa forma, basta encontrar os itemsets freqüentes de cada tabela isoladamente e posteriormente realizar combinações entre os itemsets que aparecem juntos nas transações da tabela fato.

Para isso, é usado um vetor de n dimensões, onde n é o número de dimensões envolvidas, e os elementos referem-se aos itemsets freqüentes encontrados nessas tabelas. Os itemsets das várias dimensões são combinados para formar os itemsets presentes na tabela fato. Dessa forma, é possível contar a freqüência dos itemsets freqüentes com uma varredura na tabela fato e com auxílio desse vetor.

Considerando T_{1n} uma tabela fato e T_t uma tabela dimensão, $T = T_{1n} \bowtie T_1 \bowtie T_2 \bowtie ... \bowtie T_n$ é a tabela resultante da junção da tabela fato e suas dimensões. O método apresentado realiza a mineração nessa tabela T, que é calculada e usada uma única vez. Dessa forma, T não é materializada, o que diminui os custos com armazenamento.

O algoritmo apresentado consiste em uma versão descentralizada do algoritmo Apriori com uma alteração na contagem do suporte dos itemsets, a qual é feita na tabela fato. Esse algoritmo é composto por duas fases:

Fase 1:

1. Contagem das ocorrências das chaves estrangeiras na tabela fato e armazenamento desses valores em vetores. Para cada tabela dimensão é criado um vetor cujo tamanho é igual ao número de transações da dimensão tratada.
2. Contagem das ocorrências dos itemsets nas tabelas dimensão utilizando a versão modificada do Apriori.

Fase 2: Em cada linha de T e para cada dimensão T_t, é identificado o conjunto i_t de todos os itemsets freqüentes que fazem parte dessa transação de T_t. Dessa forma, são calculados todos i_t para t = 1..n, sendo n o número de dimensões envolvidas. No próximo passo, o vetor é analisado e todo elemento I_T que consiste em uma combinação de elementos de i_n (i_1 x i_2 x ... x i_n) tem seu contador incrementado. No final do processo, esse vetor conterá o número de ocorrências de todos os itemsets candidatos e será usado para analisar quais deles são realmente freqüentes.

Na figura 3.7 são mostradas as dimensões *Cliente* e *Serviço* e a tabela fato *Contrato*. O primeiro passo para minerar essas tabelas consiste na varredura da tabela *Contrato* e na contagem do número de ocorrências das chaves estrangeiras da mesma. São criados 2 vetores, um para a tabela *Cliente* e outro para *Serviço*, sendo que cada posição desses vetores refere-se a um valor de chave primária das dimensões envolvidas e o valor armazenado consiste no número de ocorrências dessa chave na tabela fato. Posteriormente é aplicado o algoritmo Apriori modificado nas dimensões isoladamente para buscar o conjunto de itemsets freqüentes.

Na segunda fase, é criado um vetor para armazenar a freqüência dos itemsets encontrados. Esse vetor possui 2 dimensões: os elementos de uma dimensão referem-se aos itemsets freqüentes da tabela *Cliente* e a outra, aos da tabela *Serviço*.

Cliente

IdCliente	Sexo	Idade	Região
Cl1	F	20..25	Sul
Cl2	M	25..30	Centro
Cl3	M	20..25	Centro
Cl4	F	20..25	Sul
Cl5	F	25..30	Sul
Cl6	F	20..25	Norte
Cl7	M	25..30	Centro

Contrato

IdServiço	IdCliente	Valor
Sr1	Cl2	50..100
Sr1	Cl1	100..500
Sr3	Cl7	50..100
Sr2	Cl7	50..100
Sr2	Cl2	30..50
Sr1	Cl4	50..100
Sr4	Cl3	50..100
Sr2	Cl5	30..50
Sr1	Cl5	0..10
Sr1	Cl6	30..50

Serviço

IdServiço	Nome
Sr1	Previdência privada
Sr2	Seguro de vida
Sr3	Financiamento
Sr4	Empréstimo

Figura 3.7 – Tabelas de um esquema estrela

Além disso, é realizada a junção T = Cliente * Contrato * Serviço, porém seu resultado não é materializado. Essa tabela é varrida, e para cada linha da mesma é analisado se existem itemsets formados por itens freqüentes da tabela *Cliente* e da tabela *Serviço*. Caso exista, o valor do contador armazenado no vetor citado acima é incrementado.

O método apresentado concentra-se na otimização dos custos com operações de entrada e saída. Uma desvantagem desse método é a possível geração de um grande número de itemsets candidatos quando o número de itemsets das dimensões é alto, o que aumenta o número de falsos candidatos gerados. Dessa forma, Jensen e Soparkar (JENSEN; SOPARKAR, 2000) apresentam algumas variações desse algoritmo, uma para otimizar o uso da memória e uma versão híbrida, que ora otimiza o uso da memória, ora as operações de E/S.

3.4.4. Algoritmo Connection

O algoritmo Connection (RIBEIRO, 2004) minera regras de associação multifatos sem a necessidade de junção das tabelas envolvidas. Para isso, é necessário que essas tabelas tenham uma dimensão em comum. O Connection foi desenvolvido a partir do algoritmo FP-Growth (HAN; PEI; YIN, 2000) e serviu como base para o desenvolvimento deste trabalho.

O algoritmo Connection utiliza uma estrutura chamada *MFP-Tree* para determinação dos itemsets freqüentes#[1] locais. Essa estrutura consiste em uma adaptação da *FP-Tree* do algoritmo FP-Growth (HAN; PEI; YIN, 2000), e o processo de construção dessas duas estruturas é semelhante, exceto pelo fato que para a construção da *MFP-Tree* são

[1] Um itemset é *freqüente*# quando satisfaz o mínimo suporte# e o mínimo peso pré-estabelecidos pelo usuário.

considerados os conceitos de blocos e segmentos, além dos itemsets frequentes# e da inclusão do campo “peso” em sua tabela header.

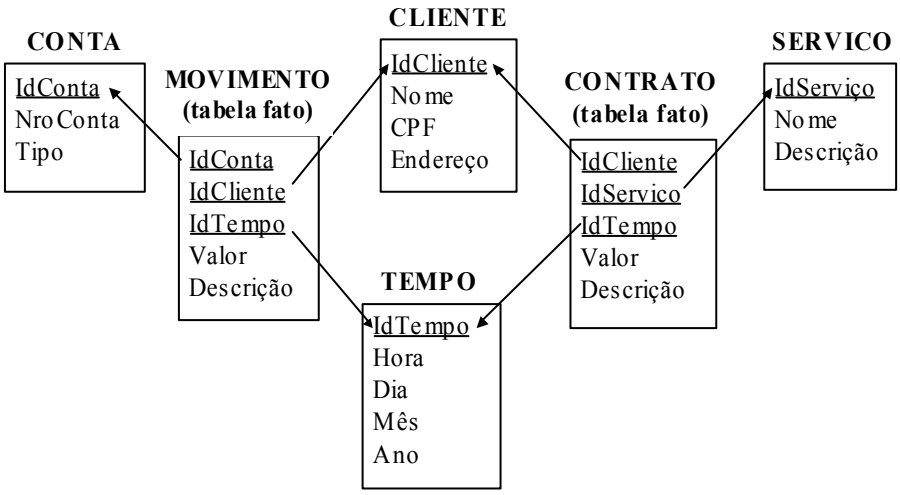

Figura 3.8 DW com informações bancárias

Considerando o exemplo da Figura 3.8, o algoritmo permite analisar a relação entre a quantia movimentada por um usuário e os serviços contratados pelo mesmo. Dessa forma, devem ser consideradas as tabelas fato *Movimento* e *Contrato*, que compartilham a tabela dimensão *Cliente*.

Movimento

IdConta	IdCliente	Valor
Ct1	Cl1	10000..15000
Ct2	Cl2	5000..10000
Ct3	Cl2	10000..15000
Ct4	Cl3	15000..20000
Ct5	Cl3	10000..15000

Contrato

IdServico	IdCliente	Valor
Sr1	Cl1	200..500
Sr2	Cl1	100..200
Sr1	Cl2	500..1000
Sr2	Cl2	200..500
Sr3	Cl2	100..200
Sr3	Cl3	200..500
Sr2	Cl4	100..200

Figura 3.9 - Blocos e segmentos das tabelas fato envolvidas

O algoritmo Connection modifica algumas definições de medidas de interesse e para isso considera os conceitos de blocos e segmentos. Na Figura 3.9 são mostrados os dados das duas tabelas fato envolvidas no processo, juntamente com seus blocos e segmentos. Os blocos correspondem aos conjuntos de tuplas de uma tabela com um valor de atributo em comum, no caso o atributo *IdCliente*. Nesse caso, os blocos são mostrados em cores alternadas.

Um segmento corresponde a um conjunto de blocos de uma tabela que possuem blocos correspondentes em outra tabela; no exemplo são mostrados segmentos que compartilham o atributo *IdCliente*. A correspondência entre os segmentos é indicada pelas setas que ligam as duas tabelas. Alguns blocos podem não formar segmento, como é o caso do bloco referente a *IdCliente = ”Cl4”* da tabela *Contrato*.

As medidas de interesse usadas pelo algoritmo são o peso, o suporte# e a confiança#. O peso de um item corresponde à razão entre o número de segmentos que contém determinado item, e o número de blocos que o contém. O suporte# de um item corresponde à razão entre o número de segmentos que contém o item e o número total de segmentos. Já o suporte# de uma regra X→Y corresponde ao número de segmentos que contém o itemset XUY, pelo número de segmentos totais. A confiança# de uma regra X→Y corresponde à razão entre o número de segmentos em que XUY ocorre e o número de segmentos em que apenas X ocorre. Dessa forma, uma regra é freqüente caso satisfaça os limites mínimos dessas medidas de interesse pré-estabelecidos.

Como exemplo será considerada a regra *ValorMovimento=10000..15000 → IdServiço=”Sr1”* obtida a partir dos dados da figura 3.9. Nesse caso, pode-se obter:

Peso(*ValorMovimento=10000..15000*) = 3/3 = 1

Peso(*IdServiço=”Sr1”*) = 2/2 = 1

Suporte#(*ValorMovimento=10000..15000→IdServiço=”Sr1”)* = 2/3 = 0,66

Confiança#(*ValorMovimento=10000..15000→IdServiço=”Sr1”)*=2/3= 0,66

Ribeiro (2004), apresenta o algoritmo de mineração de regras de associação multifatos dividido em 4 fases:

- Identificação dos segmentos
- Cálculo do suporte# e do peso dos itemsets de cada tabela fato.
- Busca pelos itemsets freqüentes globais
- Geração de regras de associação multifatos.

O algoritmo Connection busca os itemsets freqüentes locais fazendo contagem direta do suporte. Posteriormente, combina itemsets provenientes de relações distintas para encontrar os itemsets freqüentes globais. Para obter o suporte# desses itemsets globais, é utilizada a intersecção de *tidlists*. Finalmente, a partir dos itemsets freqüentes# são criadas as regras fortes.

Na Figura 3.10 é mostrado o algoritmo Connection. Na linha 1 os segmentos de todas as tabelas envolvidas são encontrados e os identificadores dos blocos desses segmentos são

adicionados a T. Nas linhas 2 e 3, os conjuntos M e R, que vão armazenar as estruturas de dados MFP-Tree's e os itemsets freqüentes# locais respectivamente, são inicializados com vazio.

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

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

Figura 3.10 Algoritmo Connection (RIBEIRO; 2004)

Nas linhas 4 a 9 existe um laço, o qual é executado para cada tabela fato. Na linha 5, é construída a *MFP-Tree* da tabela fato e na linha 6, a *MFP-Tree* é usada para obter os itemsets freqüentes# locais dessa tabela fato. Nas linhas 7 e 8 a estrutura *MFP-Tree* e o conjunto de itemsets freqüentes# locais adicionados aos conjuntos M e R, respectivamente. O laço reinicia para a próxima tabela fato envolvida.

Na linha 10, os conjuntos M e R são usados para obter os itemsets freqüentes globais através da intersecção de tidlists das MFP-Tree's de M. Na linha 11, as regras de associação fortes são geradas a partir dos itemsets freqüentes# globais. Na linha 12 o conjunto de regras é retornado e o algoritmo finaliza.

Para exemplificar o funcionamento do algoritmo Connection, será considerada a figura 3.9. Os valores estabelecidos para as medidas de interesse são min_peso = 70%, min_conf# = 80% e min_sup# = 60%. Nas figuras 3.11 e 3.12 são mostradas as MFP-Tree's criadas para as tabelas fato envolvidas.

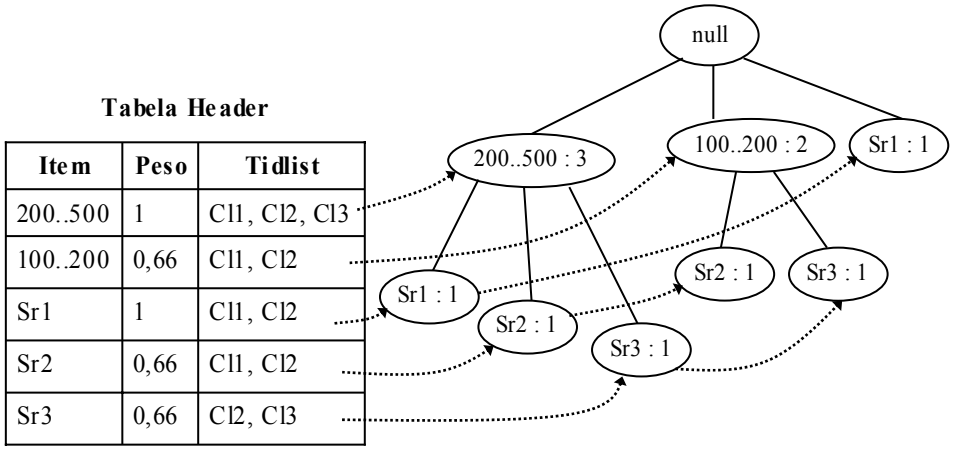

Figura 3.11 - *MFP-Tree* criada para a tabela fato Contrato

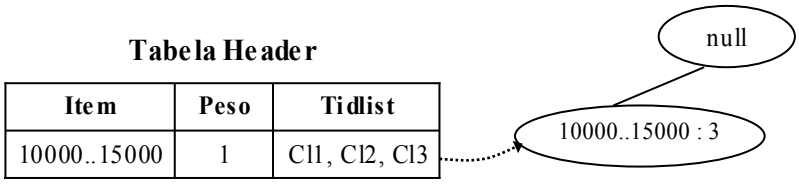

Figura 3.12 - *MFP-Tree* criada para a tabela fato Movimento

A partir das MFP-Tree's é possível obter os itemsets freqüentes# locais, os quais são combinados para obter os itemsets freqüentes# globais. Abaixo são mostrados dois padrões que foram encontrados neste exemplo.

- *ValorMovimento=10000..15000 → IdServiço="Sr1"* onde peso(*ValorMovimento*=10000..15000)=100%, peso(*IdServiço="Sr1"*)=100%, suporte# = 66% e confiança# = 66%.
- *ValorMovimento=10000..15000 → ValorContrato=200..500* onde peso(*ValorMovimento=10000..15000*)=100%, peso(*ValorContrato=200..500*)=100%, suporte# = 100% e confiança# = 100%.

O primeiro padrão indica que clientes que movimentam a quantia de R$ 10.000,00 a R$ 15.000,00 por ano tendem a adquirir o serviço "Sr1" e o segundo padrão indica que o mesmo grupo de clientes tende a adquirir serviços cujo preço está na faixa de R$ 200,00 a R$ 500,00. Nesse caso, os padrões encontrados indicam a existência de um grupo de compradores em potencial para determinado serviço ou grupo de serviços, o que pode ser usado para direcionar o envio de promoções de serviços para um grupo de clientes.

3.4.5 Algoritmo MRFP Growth

Kanodia (KANODIA, 2005) apresenta o algoritmo MRFP Growth, que consiste em uma adaptação do FP-Growth para que o mesmo possa tratar várias tabelas relacionadas entre si através de um atributo em comum. Esse algoritmo considera um conjunto de entrada composto por uma tabela primária e possivelmente várias tabelas secundárias. Quando não há tabelas secundárias, o MRFP Growth se comporta como o FP-Growth tradicional.

É possível minerar vários conjuntos de entrada, sendo que o processo é feito de forma incremental para cada conjunto, e o resultado final dessas fases é integrado. Além disso, os atributos que não são interessantes para a geração de regras podem ser filtrados.

A abordagem em questão utiliza os conceitos de ID e ID set, os quais são importantes para referenciar o local em que os itens ocorrem, funcionando como índices. O ID de um item representa o valor da chave primária P_K da tupla onde esse item ocorreu. Por exemplo, se ID(A) = {3,4} isso indica que o item A ocorreu nas tuplas em que $P_K = 3$ ou $P_K = 4$. Já o ID set representa o conjunto de identificadores no qual um padrão freqüente ocorre. Considerando A_i um item do padrão freqüente P, então o ID set de P = $\left\{ \bigcup_{i=1}^{k} ID(A_i) \right\}$.

O processo de mineração realizado pelo MRFP Growth funciona da seguinte forma: inicialmente é necessário minerar cada tabela envolvida separadamente e os resultados gerados são armazenados em uma tabela temporária. Em uma segunda fase, essa tabela temporária é minerada para gerar os padrões multi-relacionais freqüentes.

Quando se usa a tabela temporária, garante-se que apenas os registros mais freqüentes serão armazenados, evitando que registros não interessantes sobrecarreguem a memória como ocorre na operação de junção. Dessa forma, os ID's sets das tabelas envolvidas são propagados para a tabela temporária, com o objetivo de realizar a junção das tabelas relacionadas com um custo bem inferior ao da operação de junção tradicional.

Na figura 3.13 é mostrado o algoritmo MRFP Growth, o qual possui as etapas de geração da árvore e mineração da mesma em comum com o FP-Growth. Uma outra etapa utiliza o procedimento *ID_Item_Mapping*, que faz o mapeamento de ID's para itens na fase final do processo.

Considerando ainda a figura 3.13, pode-se notar que o algoritmo tem como entrada duas medidas de interesse: *suporte s* e *cross suporte* χ. A medida *suporte* é a mesma medida usada pelo algoritmo FP-Growth (HAN; PEI; YIN, 2000), e é usada na primeira fase do algoritmo,

quando cada tabela envolvida é minerada localmente. Já a medida *cross suporte* é nova, e é útil para quantificar o interesse dos padrões multi-relacionais.

Algoritmo: MRFP Growth
Entrada: Uma tabela primária P, sua chave primária Pk, n tabelas secundárias Sn, seus atributos, valores de suporte e cross suporte.
Saída*:* O conjunto com os padrões multi-relacionais freqüentes
Método*:*

1. Para cada tabela secundária Sn e para a tabela primária P
   a. Gere uma MRFP-Tree para os itens da tabela
   b. Armazene o Id de cada item frequente
   c. Minere a MRFP-tree gerada.
   d. Armazene os ID sets dos padrões freqüentes gerados
2. Construa uma MRFP-tree para os ID's de ID sets de todos os padrões freqüentes gerados pelas tabelas secundárias.
3. Minere a MRFP-tree final. Para os padrões freqüentes de ID, obtenha os padrões freqüentes associados a esses ID's usando o procedimento ID_item_Mapping e finalmente obtenha os padrões multi-relacionais freqüentes.

Figura 3.13 Algoritmo MRFP Growth (KANODIA, 2005)

O valor de *cross suporte* χ de um padrão P indica quantas vezes seu ID set é repetido em diferentes padrões freqüentes em qualquer tabela do banco de dados, ou seja, indica a freqüência com que o padrão ocorre no banco de dados todo. Como exemplo, considere o padrão freqüente A, B, C, D, E, G {1, 4} composto pelos padrões A, B {1,4}, C, D {1,4} e E, G {1,4}, freqüentes nas tabelas T_1, T_2 e T_3, onde χ(A,B)=5, χ(C,D)=3 e χ(E,G)=10. Nesse caso, pode-se dizer que χ(A, B, C, D, E, G)=3.

A árvore gerada para cada tabela envolvida, chamada de MRFP Tree, é semelhante à *FP-Tree* do FP-Growth (HAN; PEI; YIN, 2000), exceto pelo fato de que a *MRFP Tree* guarda além do suporte do item em questão, o seu índice (ID), que indica onde o item ocorre no banco de dados analisado. Após a criação das MRFP Tree's, cada uma será minerada separadamente da mesma forma como ocorre no algoritmo FP-Growth.

Finalmente as tabelas poderão ser analisadas em conjunto, com o auxílio da tabela temporária *frequent_patterns*, a partir da qual é criada a *MRFP-Tree* final, a partir dos ID sets dos padrões. Para a construção dessa última árvore, cada ID de um ID set é tratado como um item e cada identificador único de um padrão é tratado como um ID. Uma vez que a árvore é criada, a mesma é minerada em busca de padrões freqüentes. No entanto, dessa vez os

padrões freqüentes encontrados consistem de ID's e são mapeados para itens reais usando o procedimento ID_Item_Mapping, o qual é mostrado na figura 3.14.

Procedimento*:* ID_Item_Mapping
Entradas: ID set com os ID's dos padrões freqüentes encontrados na tabela *frequent_patterns*.
Saída: O conjunto com os padrões freqüentes, dado um ID set.
Método*:*

1. Encontre todos os registros da tabela *frequent_patterns* onde o ID esteja em ID set.
2. Construa um novo padrão P.
3. Para cada transação T encontrada faça
   a. P=P ∪ T, P.suporte = mínimo T.suporte encontrado, e P.ID set = P.ID set ∩ T.ID set.
   b. Adicione T à tabela de regras com T.suporte como suporte e T.ID set como ID set.
4. Adicione P à tabela de regras.

Figura 3.14 Procedimento ID_Item_Mapping (KANODIA, 2005)

Para exemplificar o funcionamento do algoritmo, o MRFP Growth será aplicado sobre as tabelas 3.1, 3.2 e 3.3. A tabela 3.1 representa a tabela primária do conjunto de entrada e contém informações sobre os clientes.

Tabela 3.1 Tabela Primária *Cliente*

IdCliente	Faixa_Etária	Classe_Social	Escolaridade
Cl1	30..35	A	Superior
Cl2	40..45	B	Médio
Cl3	35..40	A	Superior
Cl4	30..35	B	Superior

As tabelas 3.2 e 3.3 representam as tabelas secundárias do conjunto de entrada do MRFP Growth, e contêm respectivamente informações sobre as movimentações bancárias dos clientes e sobre os serviços contratados pelos mesmos.

Tabela 3.2 Tabela Secundária *Movimento*

IdCliente	Tipo Conta	Quantia
Cl1	Conta Corrente	10000..15000
Cl2	Poupança	5000..10000
Cl3	Conta Corrente	15000..20000
Cl3	Poupança	10000..15000
Cl4	Conta Corrente	10000..15000

Tabela 3.3 Tabela Secundária *Contrato*

IdCliente	Serviço	Valor
Cl1	Sr1	200..500
Cl1	Sr2	100..200
Cl2	Sr1	500..1000
Cl2	Sr2	200..500
Cl2	Sr3	100..200
Cl3	Sr3	200..500
Cl4	Sr2	100..200

A primeira fase do algoritmo consiste em analisar cada tabela separadamente, construindo sua árvore e posteriormente realizando a mineração, para obter os padrões freqüentes locais. Nas figuras 3.15, 3.16 e 3.17 são mostradas as MRFP Tree's geradas a partir das tabelas 3.1, 3.2 e 3.3, respectivamente.

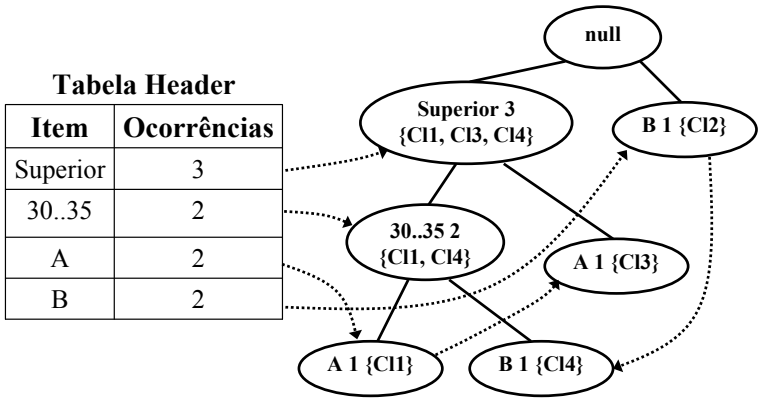

Figura 3.15 *MRFP Tree* referente à tabela primária *Cliente*

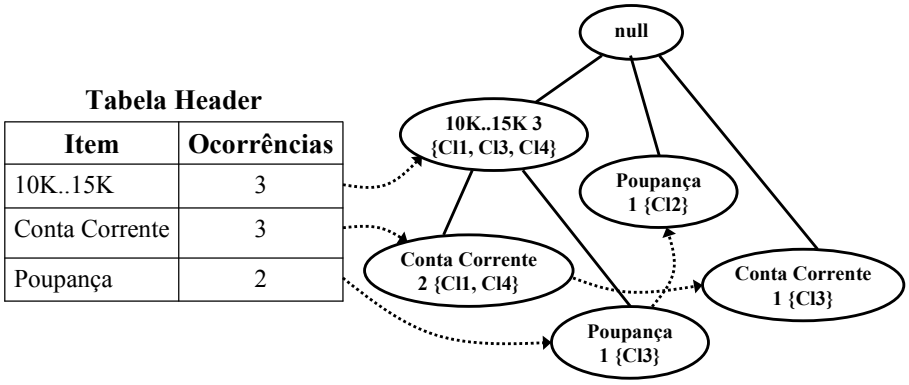

Figura 3.16 *MRFP Tree* referente à tabela secundária *Movimento*

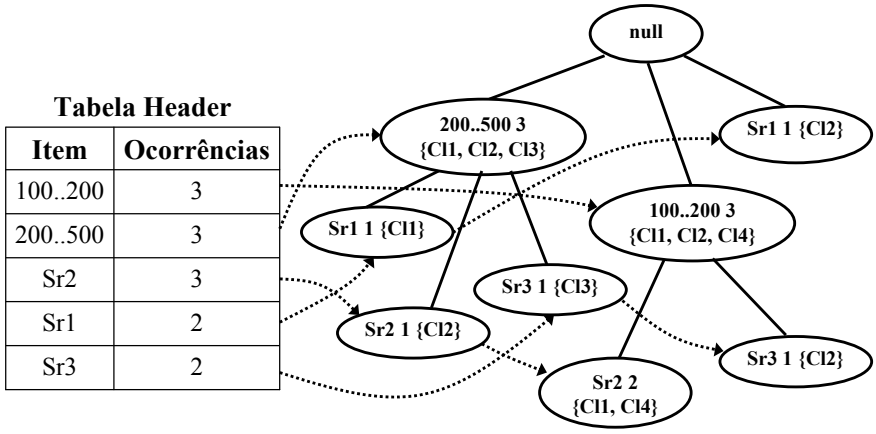

Figura 3.17 *MRFP Tree* referente à tabela secundária *Contrato*

Cada *MRFP Tree* gerada é minerada, e os padrões freqüentes encontrados são usados para preencher a tabela *frequent_patterns,* que pode ser visualizada na tabela 3.4. Para cada árvore analisada, os padrões encontrados são inseridos em *frequent_patterns* à medida que são encontrados, de modo que no final da primeira fase, a mesma contenha todos os padrões freqüentes encontrados isoladamente em cada tabela envolvida.

Tabela 3.4 *Frequent_patterns*

ID	ID set	Padrões Freqüentes	Suporte
1	Cl1, Cl4	Superior, 30..35	2
2	Cl1, Cl4	100..200, Sr2	2
3	Cl1, Cl4	10K..15K, Conta Corrente	2

A partir desse ponto tem início a segunda fase do processo, a qual é responsável pela descoberta de padrões freqüentes multi-relacionais. Para isso, a tabela 3.4 é analisada, considerando os identificadores dos clientes (campo ID set) como itens e os identificadores como ID's (campo ID). Seguindo essas definições, a *MRFP Tree* deste exemplo foi criada e pode ser visualizada na figura 3.18.

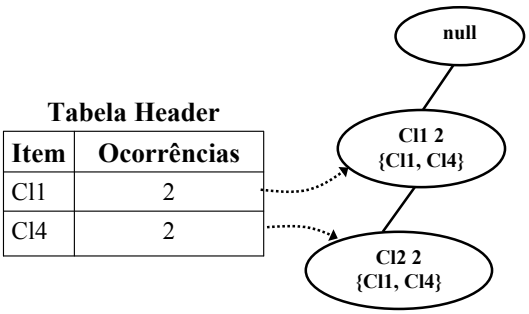

Figura 3.18 *MRFP Tree* final

Para obter os padrões freqüentes multi-relacionais, a *MRFP Tree* final da figura 3.18 é minerada e os padrões freqüentes descobertos são inseridos na tabela 3.5. A partir desses padrões, compostos pelos ID sets da tabela *frequent_patterns* (tabela 3.4), é possível fazer combinações entre seus itens para obter os padrões freqüentes multi-relacionais.

Na tabela 3.5 é mostrado o padrão freqüente multi-relacional encontrado nesse exemplo. Esse padrão foi obtido a partir da união dos itens da tabela *frequent_patterns* (tabela 3.4), cujo ID set era *{Cl1, Cl4}*, que é o padrão freqüente encontrado na árvore anterior. É interessante observar que além desse padrão multi-relacional, o algoritmo também encontra os padrões freqüentes locais de cada tabela envolvida, como pode ser observado na tabela 3.4. Um outro exemplo de aplicação deste algoritmo pode ser encontrado em (TEREDESAI; KANODIA; GABORSKI, 2004).

Tabela 3.5 Padrões freqüentes multi-relacionais

Cross Support	ID set	ID	Padrões Freqüentes
2	Cl1, Cl4	1 2 3	Superior, 30..35, 100..200, Sr2, 10K..15K, Conta Corrente

3.4.6. Algoritmo Apriori Group

O algoritmo Apriori Group (RIBEIRO; VIEIRA; TRAINA, 2005) descobre regras de associação multi-relacionais baseando-se nos agrupamentos formados pelos itens presentes no banco de dados. Para obtenção dos padrões multi-relacionais é necessária uma etapa de pré-processamento na qual é realizada a operação de junção entre as múltiplas tabelas envolvidas, e o algoritmo é aplicado sobre a tabela resultante da junção.

O Apriori Group, como o nome sugere, consiste em uma adaptação do algoritmo Apriori (AGRAWAL; SRIKANT, 1994). A adaptação consiste em considerar como unidade de análise os agrupamentos formados pelos itens, e não somente as transações como no Apriori tradicional. Isso se deve ao fato de que no caso multi-relacional são necessárias algumas alterações na contagem das ocorrências dos itens, a fim de evitar os possíveis problemas causados pela redundância gerada pela operação de junção.

Tabela 3.6 Informações sobre Clientes

IdCliente	Faixa_Etária	Escolaridade
Cl1	30..35	Superior
Cl2	40..45	Médio
Cl3	35..40	Superior
Cl4	30..35	Superior

Para mostrar como o Apriori Group analisa os agrupamentos, considere as tabelas 3.6 e 3.7, que mostram respectivamente informações sobre os clientes de um banco e a relação dos serviços contratados pelos mesmos.

Tabela 3.7 Contrato de Serviços

IdCliente	Serviço	Valor
Cl1	Seguro de vida	30K..50K
Cl1	Previdência privada	50K..100K
Cl2	Seguro de vida	10K..30K
Cl2	Previdência privada	30K..50K
Cl2	Empréstimo	10K..30K
Cl3	Empréstimo	30K..50K
Cl4	Previdência privada	50K..100K

Para que o Apriori Group busque por padrões multi-relacionais envolvendo essas duas tabelas, é necessário realizar previamente a junção entre as mesmas. Isso somente é possível, caso as tabelas envolvidas possuam pelo menos um atributo em comum. Nas tabelas anteriores, pode-se observar que ambas possuem o identificador *IdCliente*, e é esse atributo que será usado para relacioná-las. O resultado da junção dessas duas tabelas é mostrado na tabela 3.8, que é usada como entrada para o algoritmo em questão.

Tabela 3.8 Tabela Resultante da Junção

IdCliente	Faixa Etária	Escolaridade	Serviço	Valor
Cl1	30..35	Superior	Seguro de vida	30K..50K
Cl1	30..35	Superior	Previdência privada	50K..100K
Cl2	40..45	Médio	Seguro de vida	10K..30K
Cl2	40..45	Médio	Previdência privada	30K..50K
Cl2	40..45	Médio	Empréstimo	10K..30K
Cl3	35..40	Superior	Empréstimo	30K..50K
Cl4	30..35	Superior	Previdência privada	50K..100K

Na tabela 3.8 pode-se observar que existe redundância nos valores referentes à *faixa etária* e à *escolaridade* dos clientes, o que pode distorcer os valores das medidas de interesse *suporte* e *confiança*. Por essa razão, a mineração será realizada considerando o agrupamento formado por cada *cliente*, de forma que os valores redundantes serão contabilizados uma única vez em cada agrupamento.

Os agrupamentos podem ser visualizados na tabela 3.8 em cores alternadas. Por exemplo, o agrupamento formado pelo cliente cujo identificador é *Cl1,* compreende o conjunto de tuplas nas quais *IdCliente=Cl1*. Isso representa todas as informações disponíveis a respeito do cliente *Cl1,* compreendendo duas transações. Note que é possível a ocorrência

de um caso especial em que todos os agrupamentos sejam formados por apenas uma transação. Nesse caso, o algoritmo *Apriori Group* comporta-se como o *Apriori* tradicional.

A introdução do conceito *agrupamento* sugere ainda alterações nas definições das medidas de interesse usadas, que também passarão a ser contabilizadas tendo como referência os agrupamentos analisados, e não mais as transações isoladas. Dessa forma, são definidas as medidas *suporte usando agrupamentos supa* e *confiança usando agrupamentos confa.*

Assim, pode-se dizer que *supa(X)* representa o *suporte usando agrupamento* de um *itemset X*, que corresponde ao número de agrupamentos que contêm *X* pelo número total de agrupamentos existentes na base. Essa mesma medida também pode ser calculada para uma regra de associação, ou seja, *supa(X→Y)* corresponde ao número de agrupamentos que contêm $X \cup Y$ pelo número total de agrupamentos da base.

Da mesma forma, o valor *confa(X→Y)* representa a confiança usando agrupamento da regra *X→Y*, que corresponde ao número de agrupamentos em que $X \cup Y$ ocorre pelo número de agrupamentos onde somente *X* ocorre.

Para exemplificar essas medidas, considere a tabela 3.8. Nessa tabela, pode-se observar que o *supa(faixa_etária=40..45)=1/4=25%,* pois o item *faixa_etária=40..45* ocorreu em apenas 1 agrupamento, apesar de ocorrer em 3 transações, e a base total possui 4 agrupamentos.

Considere agora a regra *Serviço=previdência_privada → faixa_etária=30..35.* Pode-se observar que *supa(Serviço=previdência_privada → faixa_etária=30..35)=2/4=50%,* pois o *itemset {previdência_privada, faixa_etária}* ocorre em 50% dos agrupamentos base. Analisando a *confiança usando agrupamentos* da regra, pode-se observar que *confa(Serviço=previdência_privada → faixa_etária=30..35)=2/3=66%,* pois o item *Serviço=previdência_privada* ocorre em 3 agrupamentos e o item *faixa_etária=30..35* ocorre em 2 dos mesmos.

Na figura 3.19 o algoritmo Apriori Group é detalhado. No passo 1, o conjunto de agrupamentos G é inicializado. Nos passos de 2 a 7, a base é percorrida para verificação dos agrupamentos presentes, os quais são adicionados ao conjunto G. Nos passos de 8 a 10, é feita a contagem das ocorrências dos itens, considerando os agrupamentos.

Nos passos seguintes, a estratégia é semelhante ao Apriori tradicional. No passo 11, o conjunto L_1 de *1-itemsets* freqüentes é preenchido como os itens que satisfazem o mínimo valor de *supa*.

Algoritmo: Apriori-Group
Entrada: A base de dados D, o valor de suporte mínimo minsup, valor de mínima confiança minconf
Saída: conjunto R de regras de associação

```
1. G = Ø
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}
12.para(k = 2;L_{k-1}≠ Φ;k++)
13.   C_k =Apriori-gen(Lk-1)
14.   para cada agrupamento g  G
15.      para cada candidato c ϵ C_k contido em g faça
16.            c.contador++
17. L_k = { c ϵ C_k | c.contador/|G|  >= minsup }
18.L = ∪_k L_k
19.R = Ø
20.para cada itemset X ϵ L
21.  adicione a R todas as regras Y→X-Y, onde Y⊂X e sup(X)/sup(Y) >= minconf
```

Figura 3.19 Algoritmo Apriori Group (RIBEIRO; VIEIRA; TRAINA, 2005)

No passo 13 são feitas combinações entre os itens de um conjunto L_{k-1} para formação de um conjunto C_k de *k-itemsets* candidatos, a partir da função *Apriori_Gen* (AGRAWAL; SRIKANT, 1994). Nos passos 14 a 16, para cada conjunto C_k a base é varrida, e para cada *k-itemset* de C_k, seu contador é incrementado caso seja a primeira vez que ocorre em seu agrupamento.

No passo 17 um novo conjunto L_k de *itemsets* freqüentes é formado com os candidatos que satisfizeram o valor mínimo de *supa*. E finalmente são geradas as regras que satisfazem o valor mínimo de *confa* nos passos 20 e 21.

3.5. Considerações Finais

Neste capítulo foram apresentados o conceito de mineração multi-relacional e suas aplicações em diversas áreas. Além disso, foi discutida a abordagem do problema utilizando Lógica Indutiva e posteriormente foram apresentados alguns trabalhos que tratam o mesmo problema do ponto de vista da área de Banco de Dados. Nesse último caso, pode-se notar que os trabalhos apresentados concentram-se na mineração de *data warehouses* e consideram tabelas ligadas através da modelagem do banco de dados. É importante salientar que as

estratégias propostas originalmente para DW's podem do mesmo modo ser aplicadas para tabelas modeladas segundo o esquema entidade relacionamento.

Tabela 3.9 Características dos algoritmos multi-relacionais

Algoritmo	Vantagens	Desvantagens
Mineração Usando SQL	Espaço de busca limitado; Estende o formato das regras; Otimiza o processo usando tabelas temporárias; Evita junções entre grandes tabelas; Otimiza ordem de consultas.	Cláusula WHERE do SQL implica em junções, podendo ocorrer várias durante o processo; Não é um algoritmo e sim um framework, já que é realizado usando SQL pelo SGBD; Como realiza junção, os itens que não possuem correspondente em outra tabela são descartados; Não redefine medidas de interesse para o caso multi-relacional.
Mineração distribuída em DW	Não realiza junção; Armazena tabelas temporárias em árvores de prefixo; Minimiza as varreduras no banco, armazenando informações em estruturas temporárias como as árvores de prefixo; Redefinição das medidas de interesse para o modo multi-relacional.	Pré-processamento excessivo; Transformação da tabela para formato binário pode aumentar demasiadamente seu tamanho; Muitas estruturas temporárias e auxiliares; Geração de itemsets candidatos; Itens que não possuem correspondente em outra tabela são descartados; Gera somente regras multi-relacionais.
Apriori Descentralizado	Junção não é materializada; Otimização dos custos com operações de E/S; Redefinição das medidas suporte e confiança; para o modo multi-relacional.	Geração de itemsets candidatos; Armazenamento de candidatos em vetores, os quais podem se tornar muito grandes dependendo do número de falsos candidatos; Descarta itens das tabelas, visto que o resultado equivale à mineração na junção das tabelas; Gera somente regras multi-relacionais.
Connection	Não realiza junção; Não descarta itens que não possuem correspondente em outra tabela; Uma nova medida de interesse (peso) é usada para tratar esses itens; Redefinição das medidas suporte e confiança para o modo multi-relacional.	Gera somente regras multi-relacionais; É gerada uma árvore para cada tabela, o que pode consumir uma grande quantidade de memória.
MRFP Growth	Não realiza junção; Encontra regras locais e multi-relacionais; Uma nova medida de interesse (cross support) é definida para o modo multi-relacional; Não descarta itens que não possuem correspondente em outra tabela, podendo participar de regras locais.	Gera muitas estruturas temporárias, como tabelas e árvores; Para N tabelas envolvidas, são geradas N+1 árvores; Os itens que não possuem correspondentes em todas as tabelas não geram regras multi-relacionais, porém a parcela de itens descartados não é contabilizada; Processo de mineração complexo.
Apriori-Group	Simplifica o processo tratando uma única tabela; Encontra regras locais e multi-relacionais; Redefinição das medidas suporte e confiança. para o modo multi-relacional.	No pré-processamento (junção), itens que não possuem correspondente em outra tabela são descartados; Geração de itemsets candidatos.
GFP-Growth	Simplifica o processo tratando uma única tabela; Encontra regras locais e multi-relacionais; Redefinição das medidas suporte e confiança para o modo multi-relacional; Não descarta itens que não possuem correspondente em outra tabela. A medida de interesse peso é usada para esses itens; Gera somente uma árvore.	Árvore gerada pode ocupar muita memória; O pré-processamento (full outer join) pode resultar em uma tabela muito grande.

Um fato importante a ser destacado refere-se à escalabilidade, que tem se tornado um fator importante na mineração multi-relacional. Isso tem impulsionado o desenvolvimento de técnicas de melhoria dos algoritmos existentes, as quais normalmente tentam reduzir a quantidade ou a complexidade dos dados, ou então transformam o problema para a forma de uma única tabela para melhorar o desempenho (DžEROSKI; RAEDT; WROBEL, 2003).

Na tabela 3.9 podem-se observar alguns aspectos dos algoritmos apresentados neste capítulo. A partir da análise desses aspectos foi desenvolvido o algoritmo multi-relacional GFP-Growth, cujas características são mostradas no final da tabela 3.9. O algoritmo GFP-Growth é apresentado detalhadamente no próximo capítulo.

4. O Algoritmo GFP-Growth para Mineração Multi-Relacional de Regras de Associação

4.1. Considerações Iniciais

As técnicas de mineração multi-relacionais disponíveis para tratamento de dados modelados em várias tabelas relacionais, tentam de alguma forma adaptar as técnicas tradicionais de mineração de dados, para que as mesmas possam ser usadas para descoberta de conhecimento envolvendo várias tabelas.

Neste capítulo é apresentado o algoritmo desenvolvido para mineração multi-relacional, chamado de GFP-Growth (Grouping FP-Growth). São também apresentados os resultados experimentais obtidos a partir da aplicação do GFP-Growth sobre uma base de teste e comparações desses resultados com resultados obtidos por outros algoritmos de mineração disponíveis.

4.2. Motivação e Histórico do Projeto

O foco principal deste trabalho é a mineração multi-relacional de regras de associação. Inicialmente foi realizado um estudo para analisar quais eram as abordagens já utilizadas para tratar esse problema. A maioria das técnicas encontradas envolve os conceitos de lógica indutiva, enquanto os trabalhos que consideram que os dados estão organizados em um banco de dados são escassos, apesar de terem aumentado consideravelmente no período em que este trabalho foi desenvolvido. Nos trabalhos encontrados na literatura há um grande destaque para a mineração em *data warehouses* assim como a preocupação com a realização ou não da operação de junção, conforme apresentado no capítulo 3.

Procurando realizar uma análise mais profunda do problema foram realizados experimentos com o algoritmo multi-relacional *Connection* comparando o tempo de resposta e as regras obtidas desse algoritmo com os dos algoritmos de mineração tradicionais *FP-Growth* e *Apriori*. Os experimentos foram realizados utilizando uma base de dados com informações médicas sobre pacientes portadores de hepatite, disponibilizadas em *http://lisp.vse.cz/challenge/index.html*, para o evento *ECML/PKDD Discovery Challenge.*

Para realização desses experimentos, foram analisados alguns trabalhos que já haviam tratado a mesma base, como (WATANABE *et al.*, 2003) (OHSAKI *et al.*, 2003) (HO *et al.*, 2002) (OHARA *et al.*, 2004) (MATSUDA *et al.*, 2002) (YOSHIDA *et al.*, 2004) (HO *et al.*, 2003), com o intuito de verificar as etapas de pré-processamento, como discretização dos

dados e extração de uma parcela significativa dos mesmos, e realizar uma comparação dos resultados obtidos.

A partir dessas informações, foi possível direcionar o estudo para o caso multi-relacional e para isso foi utilizado como base o algoritmo *Connection*. Assim, foi possível conhecer as fases do processo de descoberta de conhecimento e analisar quais seriam as maiores dificuldades no desenvolvimento deste trabalho. Os resultados obtidos foram considerados muito satisfatórios, tendo sido parcialmente reportados em (PIZZI; RIBEIRO; VIEIRA, 2005). A descrição completa sobre os experimentos realizados pode ser encontrada em (PIZZI; VIEIRA, 2006).

A partir dessa análise percebeu-se que poderia ser simplificada a abordagem de tratamento de múltiplas tabelas reunindo-se os dados em uma única tabela, mantendo as informações dos agrupamentos, a exemplo do que foi feito no algoritmo *Apriori-Group* (RIBEIRO; VIEIRA; TRAINA, 2005). Foi então desenvolvido o algoritmo GFP-Growth baseado no *FP-Growth* (HAN; PEI; YIN, 2000). As medidas de interesse multi-relacionais usadas foram baseadas nas medidas definidas em (RIBEIRO; VIEIRA, 2004) e (RIBEIRO, 2004).

4.3. Definição do problema

A figura 4.1 mostra parte da modelagem do banco de dados disponibilizado em *http://lisp.vse.cz/challenge/index.htm* e contém informações bancárias. Para auxiliar as discussões, as informações selecionadas desse banco de dados se concentram nos clientes, suas contas e cartões vinculados às contas, além das operações realizadas, como empréstimos e ordens de pagamento.

Nesse banco de dados pode ser interessante analisar se existe alguma influência entre o status de um empréstimo e os tipos de cartões de crédito vinculados a uma conta. Para isso é necessário encontrar regras de associação envolvendo dados de duas tabelas: *Cartão* e *Empréstimo*. Essas regras indicam a relação entre os itens envolvidos e as medidas de interesse utilizadas indicarão a relevância da regra obtida.

O algoritmo Connection, apresentado no capítulo anterior, realiza a mineração de dados nessas duas tabelas, mantendo-as separadas. Este trabalho propõe uma forma alternativa em que os dados das tabelas são reunidos em uma única tabela para serem processados. Com isso obtêm-se mais regras do que usando o algoritmo Connection, já que também são consideradas as regras presentes em uma única tabela.

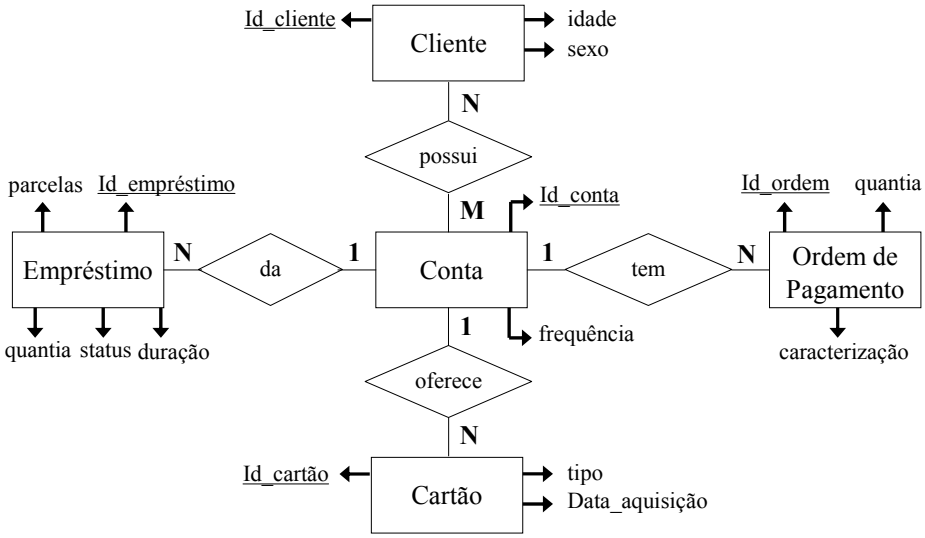

Figura 4.1 Informações bancárias

4.4. Estratégia Proposta

Para resolver o problema apresentado na seção anterior é necessário minerar em conjunto as tabelas *Cartão* e *Empréstimo*. Considere que essas tabelas possuem os dados mostrados na tabela 4.1. Nessas tabelas pode-se observar que uma conta pode possuir vários empréstimos, o que gera repetições dos identificadores de conta na tabela Empréstimo. Além disso, uma conta pode ser associada a vários cartões. No exemplo considerado, somente a conta Ct4 possui mais de um cartão para simplificar as discussões e ilustrações utilizadas neste capítulo.

Tabela 4.1 Tabelas com informações bancárias

Cartão

IdCartão	IdConta	Tipo	Data Aquisição
C1	Ct1	classic	0-1
C2	Ct2	classic	1-3
C3	Ct3	júnior	0-1
C4	Ct4	júnior	1-3
C5	Ct4	gold	1-3
C6	Ct5	gold	3-5
C7	Ct7	júnior	0-1
C8	Ct8	gold	1-3

Empréstimo

IdEmpréstimo	IdConta	Quantia	Status
E1	Ct1	0-80	A
E2	Ct3	80-180	B
E3	Ct3	0-80	A
E4	Ct3	80-180	A
E5	Ct4	0-80	A
E6	Ct6	180-600	C
E7	Ct6	80-180	A
E8	Ct7	80-180	D
E9	Ct7	180-600	C

Para realizar a mineração multi-relacional faz-se a transferência de todo o conteúdo dessas tabelas para uma única tabela, respeitando o relacionamento indicado pela chave *Id_Conta*. Para tanto, usa-se a operação *Full Outer Join*, cujo resultado pode ser visualizado na tabela 4.2.

Nesse ponto, todas as informações necessárias para a extração de conhecimento já estão reunidas em uma única tabela. No entanto, se a avaliação das medidas de interesse for feita da forma tradicional, podem ocorrer valores inconsistentes para as mesmas. Isso ocorre porque a tabela 4.2 possui dependências multivaloradas e, portanto, não está na quarta forma normal. Há duplicações dos valores dos atributos para uma mesma conta, assim como valores *null*, os quais entrariam na contagem do número total de tuplas, gerando distorções nas medidas de interesse *suporte* e *confiança*.

Tabela 4.2 Junção das tabelas envolvidas no processo de mineração

IdConta	Tipo	Data Aquisição	Quantia	Status
Ct1	Classic	0-1	0-80	A
Ct2	Classic	1-3	*null*	*null*
Ct3	Junior	0-1	80-180	B
Ct3	Junior	0-1	0-80	A
Ct3	Junior	0-1	80-180	A
Ct4	Junior	1-3	0-80	A
Ct4	Gold	1-3	0-80	A
Ct5	Gold	3-5	*null*	*null*
Ct6	*null*	*null*	180-600	C
Ct6	*null*	*null*	80-180	A
Ct7	Junior	0-1	80-180	D
Ct7	Junior	0-1	180-600	C
Ct8	Gold	1-3	*null*	*null*

Analisado o esquema mostrado na figura 4.1 nota-se que as tabelas Cartão e Empréstimo são relacionadas a partir de relacionamentos 1:N com a tabela Conta. Informalmente, sempre que dois relacionamentos 1:N independentes A:B e A:C são combinados na mesma relação, uma dependência multivalorada pode aparecer.

Para realizar a mineração da tabela 4.2 são necessárias algumas alterações em um algoritmo tradicional para que o mesmo contabilize apenas uma vez os dados que se repetem para uma mesma conta. Dessa forma, será necessário considerar que os dados da tabela estão agrupados por conta, através do atributo *IdConta,* conforme mostrado em cores alternadas na tabela 4.2. Nessa tabela nota-se que nem todas as contas possuem todos os campos preenchidos. Os registros incompletos não podem participar da contagem das medidas de interesse *suporte* e *confiança* e, portanto, não geram padrões. Todavia, esses registros serão

usados inicialmente para determinar qual a porcentagem de cada item foi realmente utilizada para extração de conhecimento.

Dessa forma, a estratégia proposta consiste na mineração de uma única tabela na qual estão presentes todos os dados pertinentes à análise, levando em conta a freqüência com que os dados ocorrem nas tabelas originais.

Para executar a estratégia proposta foi desenvolvido o algoritmo GFP-Growth, tendo como base o algoritmo FP-Growth (HAN; PEI; YIN, 2000). Resumindo, o GFP-Growth realiza a mineração sobre uma única tabela, cujos dados são provenientes de múltiplas tabelas relacionadas, considerando os agrupamentos existentes na mesma.

Na próxima seção são apresentadas algumas definições utilizadas na elaboração da estratégia apresentada.

4.4.1. Definições

Inicialmente é necessário definir como os dados da tabela serão analisados. Para isso é fundamental analisar os conceitos de blocos e agrupamentos, que foram introduzidos em (RIBEIRO, 2004) e (RIBEIRO; VIEIRA; TRAINA, 2005), respectivamente. Novas definições dos conceitos de blocos e agrupamentos são apresentadas a seguir, para se adequar ao problema aqui tratado.

Um bloco consiste no conjunto de tuplas de uma tabela que se referem a uma mesma entidade, ou seja, o conjunto de tuplas que possuem um valor de atributo identificador em comum, conforme descrito na definição 1 a seguir.

Definição 1: *Sejam T uma tabela de tamanho n, X um conjunto de atributos de T e t[X] o valor do atributo X na transação t. O conjunto de transações* $b=\{t_1,t_2,..,t_m\}$ *pertencentes a T, tal que* $t_k[X]=t_j[X]$, $\forall\, k, j \mid 1 \leq k,j \leq m$, *onde* $m=|b|$, *é chamado* ***bloco*** *de T .*

Na tabela 4.2 podem-se observar os blocos em cores alternadas. Considerando que cada bloco compreende todas as informações sobre a entidade representada, na tabela 4.2 pode-se dizer que todas as informações disponíveis sobre determinada conta se concentram em um único bloco.

Tome como exemplo o bloco cujo identificador é *Ct6*. Nesse caso, o bloco pode ser representado da seguinte forma: ID(*Ct6*)={<*null*, *null*, 180-600, C> <*null*, *null*, 80-180, A>}. Nesse caso, pode-se observar que não estão disponíveis informações sobre o tipo do cartão e o tempo que o mesmo é vinculado à conta *Ct6*, já que os atributos que representam tais

informações são nulos. Portanto, não tem sentido relacionar, para esse bloco, as informações sobre o *cartão* com as informações sobre o *empréstimo*.

Um agrupamento refere-se a um bloco em que todos os atributos possuem valores diferentes de *null*. Considerando T_1, T_2, **...** , T_n como as tabelas iniciais a serem analisadas e Tr a tabela resultante, então um agrupamento de Tr agrupa informações provenientes de T_1, T_2, **...** , T_n de uma mesma entidade. Na definição 2 pode ser encontrada uma descrição mais formal sobre agrupamentos.

Definição 2*: Sejam t_1, t_2, ... t_n o conjunto de transações de um bloco b_j de T. O bloco b_j representa um agrupamento Gr se* $\forall$ *$i=1, \ n \Rightarrow t_i[A_j] \neq null$, para $j=1,...,p$, onde A_j é um atributo de T e p é o grau de T (ou seja, o número de atributos de T).*

Se um bloco forma um agrupamento em Tr, isso indica que a entidade representada possui registros em todas as tabelas T_1, T_2, **...** , T_n analisadas. Ao contrário, quando um bloco não forma agrupamento, pode-se dizer que a entidade em questão não possui registros em pelo menos uma das tabelas T_1, T_2, **...** , T_n. A tabela 4.3 mostra os 4 agrupamentos encontrados na tabela 4.2, relativos às contas Ct1, Ct3, Ct4 e Ct7. Esses agrupamentos podem ser representados da seguinte forma:

Gr(Tr)={{<Ct1, tipo=classic, data_aquisição=0-1, quantia=0-80, status=A>}
{<Ct3, tipo=junior, data_aquisição=0-1, quantia=80-180, status=B>}
<Ct3, tipo=junior, data_aquisição=0-1, quantia=0-80, status=A>
<Ct3, tipo=junior, data_aquisição=0-1, quantia=80-180, status=A>}
{<Ct4, tipo=junior, data_aquisição=1-3, quantia=0-80, status=A>
<Ct4, tipo=gold, data_aquisição=1-3, quantia=0-80, status=A>}
{<Ct7, tipo=junior, data_aquisição=0-1, quantia=80-180, status=D>
<Ct7, tipo=junior, data_aquisição=0-1, quantia=180-600, status=C>}}

Ou, para simplificar, denota-se por: *Gr(Tr)={Ct1, Ct3, Ct4 ,Ct7}*.

Com base nos conceitos apresentados, pode-se dizer que a estratégia busca por regras de associação em uma tabela resultante da junção de múltiplas tabelas relacionadas, considerando os agrupamentos encontrados, ou seja, o processo visa descobrir conhecimento sobre determinada entidade. A seguir é apresentada uma definição formal da estratégia desenvolvida.

Tabela 4.3 Blocos e agrupamentos da base de dados

IdConta	Tipo	Data Aquisição	Quantia	Status	
Ct1	Classic	0-1	0-80	A	Ct1
Ct2	Classic	1-3	*null*	*null*	
Ct3	Júnior	0-1	80-180	B	Ct3
Ct3	Júnior	0-1	0-80	A	
Ct3	Júnior	0-1	80-180	A	
Ct4	Júnior	1-3	0-80	A	Ct4
Ct4	Gold	1-3	0-80	A	
Ct5	Gold	3-5	*null*	*null*	
Ct6	*null*	*null*	180-600	C	
Ct6	*null*	*null*	80-180	A	
Ct7	Júnior	0-1	80-180	D	Ct7
Ct7	Júnior	0-1	180-600	C	
Ct8	Gold	1-3	*null*	*null*	

***Definição 3**: Sejam T_1, T_2,...,T_n tabelas que possuem pelo menos um atributo K em comum, sendo K uma chave primária ou estrangeira de T_i, para qualquer i=1,n. Seja Tr a tabela resultante de uma operação de full outer join de T_1, T_2,...,T_n e Gr(Tr) o conjunto de agrupamentos de Tr. As regras geradas pelo **GFP-Growth** são da forma $X \rightarrow Y$, onde X, Y são itemsets, tal que $X, Y \in Gr(Tr)$ e $|Y|=1$.*

Dessa forma, um exemplo de regra gerada a partir da tabela 4.2 seria *tipo=Junior, tempo=0-1 → quantia=0-80,* o que indica que os itens *tipo=Junior, data_aquisição=0-1* e *quantia=0-80* foram encontrados com uma certa freqüência para uma mesma entidade. Como as regras são encontradas nos agrupamentos, pode-se interpretá-las com base na entidade representada, no caso a conta. Nesse caso, a interpretação seria: *"As contas que possuem cartões do tipo júnior por um período de até 1 ano, tendem a apresentar empréstimos vinculados com valores até R$80.000"*.

Para quantificar o interesse das regras multi-relacionais geradas são utilizadas as medidas *suporte usando agrupamento,* denominada *sup_a*, *confiança usando agrupamento,* denominada *conf_a* e *peso*. As medidas *sup_a* e *conf_a* são baseadas em medidas definidas em (RIBEIRO; VIEIRA; TRAINA, 2005) e consistem em alterações das medidas *suporte* e *confiança,* utilizadas freqüentemente para quantificar o interesse de regras de associação. A medida *peso* foi baseada em uma medida definida em (RIBEIRO, 2004), (RIBEIRO; VIEIRA, 2004) para quantificar o interesse de regras multi-relacionais.

O *peso* de um item indica a porcentagem de ocorrências desse item realmente utilizada no processo de descoberta de conhecimento. Essa medida é importante para quantificar a expressividade de um item frente à sua ocorrência na base analisada como um todo, conforme citado em (RIBEIRO, 2004). A definição dessa medida sofreu ajustes relativa àquela apresentada em (RIBEIRO, 2004), (RIBEIRO; VIEIRA, 2004), devido ao fato de ser aplicada a uma única tabela, conforme apresentado na definição a seguir.

***Definição 4**: O **peso** de um **item** x é a razão entre o número de agrupamentos que contém x e o número de blocos em que x ocorre. Isto é, **peso**=|Gr(x)|/|b(x)|.*

Para exemplificar o cálculo do peso, considere o item *tipo=Classic* mostrado na tabela 4.3. Nessa tabela, os blocos estão indicados por cores alternadas e os agrupamentos indicados à direita da figura. Pode-se observar que o item *tipo=Classic* ocorre em dois blocos, *Ct1* e *Ct2*, no entanto somente o bloco *Ct1* forma agrupamento. Dessa forma, o peso do item *tipo=Classic* é 50%, o que indica que 50% de suas ocorrências na base foram utilizadas para geração de regras. Isso ocorre porque a estratégia adotada busca por conhecimento apenas nos agrupamentos. O bloco *Ct2* será ignorado, pois não possui registros em todas as tabelas origens, sendo assim seus itens não contribuirão para a geração de regras multi-relacionais.

Ao gerar a regra de associação, é necessário que se analise o peso de cada item presente na mesma, conforme descrito na definição 5.

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

Considere o peso mínimo de 40% e a regra *tipo=classic, data_aquisição=0-1→ status=A,* cujos valores de peso são: *peso(tipo=classic)=50%, peso(data_aquisição=0-1)=100%* e *peso(status=A)=75%.* Pode-se dizer que essa regra satisfaz o peso mínimo, pois nenhum de seus itens possui um valor menor que 40% para o peso. Isso indica que 50% das contas que possuem cartão do tipo classic e 100% das contas que possuem cartão com data de aquisição até 1 ano possuem empréstimo. Além disso, 75% das contas que possuem empréstimo com status A possuem um cartão vinculado.

Uma outra medida de interesse utilizada é o *sup_a*. O *sup_a* de um item quantifica a freqüência com que o mesmo ocorre na base analisada. Sua definição formal é dada a seguir:

***Definição 6:** O suporte de um item x de Tr denotado por **sup_a** (suporte considerando agrupamentos) é a razão entre o número de agrupamentos em que x ocorre e o número total de agrupamentos*.

Para a medição do *sup_a* são considerados apenas os blocos que formam agrupamentos, o que corresponde à parcela dos dados efetivamente usada na geração das regras, a qual é mostrada na tabela 4.4.

Tabela 4.4 Base de dados utilizada para mineração

IdConta	Tipo	Data Aquisição	Quantia	Status
Ct1	Classic	0-1	0-80	A
Ct3	Junior	0-1	80-180	B
Ct3	Junior	0-1	0-80	A
Ct3	Junior	0-1	80-180	A
Ct4	Junior	1-3	0-80	A
Ct4	Gold	1-3	0-80	A
Ct7	Junior	0-1	80-180	D
Ct7	Junior	0-1	180-600	C

Na tabela 4.4 são mostrados os 4 agrupamentos (*Ct1*, *Ct3*, *Ct4* e *Ct7*) presentes na base de dados referente à tabela 4.3. Tomando-se como exemplo o item *status=A,* pode-se observar que o mesmo ocorre nos agrupamentos *Ct1, Ct3 e Ct4*, o que indica que *sup_a(status=A)=3/4*. Isso indica que o item *status=A* ocorre em 75% das entidades analisadas no processo de mineração, ou seja, em 75% das contas que possuem cartão e empréstimo vinculados, há pelo menos um empréstimo com status A.

Para que uma regra satisfaça o *sup_a* mínimo, é necessário analisar a freqüência com que todos os itens pertencentes à regra ocorrem juntos nas mesmas entidades da base. O *sup_a* de uma regra é dado na definição 7.

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

Assim, para obter o *sup_a* da regra *tipo=classic, data_aquisição=0-1→ status=A,* é necessário obter o *sup_a* do itemset *(tipo=classic, data_aquisição=0-1, status=A).* Analisando a tabela 4.4 pode-se constatar que os itens da regra ocorrem juntos somente no agrupamento *Ct1*, portanto *sup_a(tipo=classic, data_aquisição=0-1, status=A)=1/4.* Assim, pode-se afirmar que a regra citada anteriormente foi verificada em 25% das entidades da figura 4.4. Isso indica que em 25% das contas que possuem empréstimo e cartão vinculados, o tipo do cartão é classic, a data de aquisição é de até 1 ano e o status do empréstimo é A.

Uma outra medida de interesse utilizada na estratégia apresentada é a *conf_a*, que mede qual a probabilidade do conseqüente da regra ocorrer junto com o antecedente. A definição 8 formaliza o conceito de *conf_a*.

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

Para avaliar a *conf_a* de uma regra *X→Y*, pode-se fazer o seguinte cálculo*: sup_a(XUY)/sup_a(X).* Dessa forma, para calcular a *conf_a* da regra *tipo=classic, data_aquisição=0-1→status=A,* é necessário calcular *sup_a(tipo=classic, data_aquisição=0-1, status=A)/sup_a(tipo=classic, data_aquisição=0-1).* O resultado obtido é 100%, o que indica que o item *status=A* ocorre 100% das vezes em que o itemset *(tipo=classic, data_aquisição=0-1)* ocorre. Isso indica que todas as contas que possuem cartão e empréstimo vinculados, e o cartão é do tipo classic com data de aquisição de até 1 ano, pelo menos um empréstimo vinculado tem status A.

Definidas as medidas de interesse, é importante definir quando um itemset é considerado interessante para a análise. Neste trabalho esse itemset é chamado de itemset *freqüente_a e sua definição é dada* a seguir:

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

A partir dos itemsets *freqüentes_a* são determinadas as regras de associação, que constituem o resultado final da estratégia desenvolvida. Uma regra considerada interessante, e conseqüentemente gerada pelo processo de mineração, é chamada de regra *forte*, conforme descrito na definição 10.

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

Como exemplo, considere os seguintes valores mínimos pré-estabelecidos para as medidas de interesse: *sup_a*=20%, *conf_a*=75% e peso=50%. Uma regra de associação obtida pelo GFP-Growth é da forma:

tipo=classic, data_aquisição=0-1→status=A, sup_a=25%, conf_a = 100%, peso(tipo=classic) = 50%, peso(data_aquisição=0-1) = 100, peso(status=A) = 75%.

Analisando as medidas de interesse, nota-se que todas elas satisfizeram o mínimo pré-estabelecido e, portanto, essa é uma regra forte.

Interpretando a regra acima verifica-se que 25% das contas que possuem cartão e empréstimos vinculados, possuem pelo menos um cartão do tipo classic adquirido há 1 ano ou menos e pelo menos um empréstimo com status A. Além disso, dentre essas contas, todas

aquelas que possuem o cartão do tipo classic adquirido há no máximo 1 ano, possuem pelo menos um empréstimo com status A.

Quanto à porcentagem dos valores efetivamente usados na mineração, pode-se dizer que 50% das contas que possuem cartão do tipo classic e todas as contas que possuem cartões adquiridos há 1 ano ou menos, possuem ainda um empréstimo vinculado. Com relação às contas com empréstimo com status A, pode-se dizer que 75% delas também possuem pelo menos um cartão vinculado.

4.5. Algoritmo GFP-Growth

Com base nos conceitos e definições apresentados nas seções anteriores, foi desenvolvido o algoritmo GFP-Growth para descoberta de regras de associação multi-relacionais. O algoritmo é aplicado sobre uma única tabela que agrupa informações sobre uma entidade de interesse, sendo essas informações provenientes das várias tabelas que se deseja analisar.

O algoritmo utilizado pode ser dividido em 3 fases:

- Fase 1: Identificação dos blocos e agrupamentos.
- Fase 2: Geração de padrões *freqüentes_a*, analisando *sup_a* e peso dos itens.
- Fase 3: Geração das regras de associação, analisando *conf_a* das regras.

O algoritmo GFP-Growth, assim como o FP-Growth, realiza duas varreduras na base de dados. Na primeira varredura são calculadas as medidas de interesse usadas para avaliar se o item é interessante para análise, e na segunda varredura os itens são inseridos na árvore criada, de acordo com a ordem pré-estabelecida anteriormente pelo algoritmo.

Algoritmo GFP-Growth
Entrada: Base de dados D, sup_a mínimo, conf_a mínima, peso mínimo
Saída: Conjunto R de regras de associação

```
1. G = findGroups(D)
2. fpt = geraGFPTree(null)
3. FP = GFP-Growth(fpt, null)
4. Para cada elemento de FP
5.     Gere as regras envolvendo seus itens tal que conf_a >= mínima conf_a
```

Figura 4.2 Algoritmo GFP-Growth

Na figura 4.2 é apresentado o algoritmo utilizado para mineração de dados. O algoritmo GFP-Growth inicialmente identifica os agrupamentos presentes na base de dados analisada

através da função *findGroups* (passo 1). Nos passos 2 e 3 a *GFP-Tree* é criada através da função *geraGFPTree* e passada como parâmetro para o método GFP-Growth, para que seja minerada e o conjunto de padrões freqüentes *FP* seja gerado. Esses dois passos compreendem a fase 2 do algoritmo. Finalmente tem início a fase 3 do algoritmo, na qual os padrões *freqüentes_a* encontrados são utilizados para criação de regras de associação, com base na *conf_a* mínima estabelecida (passos 4 e 5).

```
Algoritmo: Busca por agrupamentos
Entrada: Base de Dados D
Saída: Conjunto de agrupamentos G
Método: findGroups
1.  G = { }
2.  Percorra o banco de dados D
3.    Para cada transação t {
4.      Se t não contém campos nulos {
5.        Para cada item i que ocorre em t
6.          Se for a primeira vez que ocorre nesse agrupamento{
7.            i.blocos++
8.            i.agrupamentos++
9.          }
10.       Se existir um agrupamento g ϵ G tal que ID(g) = ID(t)
11.         Adicione t ao agrupamento g
12.       Se não {
13.         Crie um novo agrupamento g tal que ID(g) = ID(t)
14.         Adicione t ao agrupamento g
15.       }
16.   }
17.   Se não
18.       Para cada item i que ocorre em t
19.         i.blocos++
20.   }
21. Retorne G
```

Figura 4.3 Método *findGroups*

A função *findGroups* é mostrada na figura 4.3 e funciona da seguinte forma: no passo 1 o conjunto de agrupamentos é inicializado. A base de dados é percorrida (passo 2) e para cada transação que não contenha campos nulos (passos 3 e 4), cada item que estiver ocorrendo pela primeira vez no agrupamento em questão terá seus contadores de agrupamentos e blocos incrementados (passos 5 a 9). Se o item já tiver ocorrido antes no mesmo agrupamento, sua ocorrência é ignorada.

Se o agrupamento em questão já existir no conjunto G, a transação é inserida a esse agrupamento (passos 10 e 11), caso contrário, um novo agrupamento é criado e a transação é inserida no novo agrupamento (passos 12 a 15). Se a transação em questão contiver campos nulos, seus itens terão apenas o contador de blocos incrementado (passos 17 a 20). No passo 21 o conjunto de agrupamentos G é retornado.

O método GFP-Growth, mostrado no passo 3 da figura 4.3 funciona da mesma forma que o método *FP-Growth* do algoritmo FP-Growth (HAN; PEI; YIN; 2000). Basicamente, esse método faz combinações entre os itens de um ramo da *GFP-Tree*, caso ela possua somente esse ramo. Caso possua mais de um ramo, cada item da header dessa árvore será combinado com o item mais próximo da extremidade do ramo, e uma nova *GFP-Tree* será gerada para essa combinação. A mineração dessa nova *GFP-Tree* é reiniciada recursivamente e cada combinação é guardada no conjunto de padrões freqüentes *FP*.

A *GFP-Tree* gerada pelo GFP-Growth é semelhante à gerada pelo algoritmo FP-Growth (HAN; PEI; YIN; 2000), porém a *GFP-Tree* armazena os agrupamentos de cada item em seus nós e na tabela header. Além disso, os itens que fazem parte da *GFP-Tree* são aqueles que compõem o conjunto L do GFP-Growth, cuja definição difere do FP-Growth.

No GFP-Growth a construção do conjunto L considera os itemsets *freqüentes_a*, os quais dependem dos valores do peso e *sup_a* de cada item. Assim, o conjunto L pode ser descrito como: *L = {item | item.agrupamentos/|G| >= mínimo sup_a E item. agrupamentos/ item.blocos >= mínimo peso}* em ordem decrescente de *sup_a.* Ou seja, L compreende os itens que satisfazem o mínimo *sup_a* e mínimo *peso*, classificados em ordem decrescente de *sup_a*.

Para exemplificar a geração do conjunto L, considere a tabela 4.2 como a base que se deseja analisar. Pode-se observar na tabela 4.5 os itens da forma como são expressos na base de dados pré-processada, juntamente com seu *sup_a* e peso. Suponha ainda que os valores mínimos pré-estabelecidos para as medidas *sup_a* e peso são 25% e 60% respectivamente. Na tabela 4.5 pode-se observar em cinza os itens que não participarão do conjunto L, pois não satisfizeram as restrições de mínimo *sup_a* ou peso e, portanto, não são *freqüentes_a*.

Tabela 4.5 *Sup_a* e peso dos itens da base de dados

Item	*Sup_a*	Peso
Tipo=Classic	25%	50%
Tipo=Júnior	75%	100%
Tipo=Gold	25%	33%
Data_aquisição=0-1	75%	100%
Data_aquisição=1-3	25%	33%
Data_aquisição=3-5	0%	0%
Quantia=0-80	75%	100%
Quantia=80-180	50%	66%
Quantia=180-600	25%	50%
Status=A	75%	75%
Status=B	25%	100%
Status=C	25%	50%
Status=D	25%	100%

Nesse ponto, os itens considerados *freqüentes_a* são classificados em ordem decrescente de *sup_a* e em seguida por ordem lexicográfica, quando dois itens apresentam o mesmo valor de *sup_a*. Os itens que vão compor o conjunto L já classificados em ordem decrescente de *sup_a* são mostrados na tabela 4.6. O conjunto *L* obtido é o seguinte: *L={Data_aquisição=0-1, Quantia=0-80, Status=A, Tipo=Júnior, Quantia=80-180, Status=B, Status=D}*.

Tabela 4.6 Itens do conjunto L

Item	*Sup_a*	Peso
Data_aquisição=0-1	75%	100%
Quantia=0-80	75%	100%
Status=A	75%	75%
Tipo=Júnior	75%	100%
Quantia=80-180	50%	66%
Status=B	25%	100%
Status=D	25%	100%

4.5.1. A *GFP-Tree* Gerada pelo GFP-Growth

Antes da geração da *GFP-Tree* é realizada uma análise de itens individuais. Nessa fase são armazenadas informações sobre as ocorrências dos itens em blocos que formam agrupamentos e blocos que não formam agrupamentos, durante o passo *findGroups* citado anteriormente. Essas informações permitem contabilizar os pesos dos itens logo no início do processo de mineração, tornando possível eliminar da análise os itens não *freqüentes_a*.

É importante salientar que do mesmo modo são medidos os valores de *sup_a* de cada item e essa medida é usada em conjunto com o peso para indicar se um item é *freqüente_a*. A contribuição da medida peso se concentra nessa fase, porém o *sup_a* ainda será usado no processo, estando presente nos nós das *GFP-Tree's* geradas, além de ser um parâmetro para identificar se um itemset é *freqüente_a* ou não.

A partir do momento em que a *GFP-Tree* é gerada, o algoritmo se concentra na análise dos itemsets, preocupando-se com a geração de itemsets *freqüentes_a*, os quais se tornarão regras de associação. Conseqüentemente as medidas de interesse envolvidas nos passos seguintes são aquelas que se aplicam aos agrupamentos de itens, que são o *sup_a* e *conf_a.*

Na figura 4.4 é mostrada a *GFP-Tree* referente à base da tabela 4.2. Cada nó possui um item, um contador e uma lista de agrupamentos. O contador e a lista de agrupamentos indicam os agrupamentos nos quais os itens desse ramo ocorreram em conjunto. Nessa figura pode-se notar que o nó possui um ponteiro que indica o próximo nó onde o mesmo item ocorre, e além

desse existem outros três ponteiros que não foram mostrados na figura, os quais indicam o nó pai, o primeiro nó filho e o próximo nó irmão.

O item, além da sua identificação nominal, é constituído por um contador e por um identificador de agrupamentos. Esse contador é contabilizado no início do processo e indica o número de ocorrências totais do item nos agrupamentos da base. O identificador de agrupamentos é útil durante a geração da *GFP-Tree* para verificar se o item já ocorreu com determinado agrupamento.

Para facilitar e orientar o acesso à *GFP-Tree* é usada uma tabela header, conforme pode ser observado na figura 4.4. Ela contém a identificação dos itens, seus contadores e os agrupamentos nos quais os itens ocorreram. Além disso, existem ponteiros que indicam a primeira ocorrência de um determinado item na *GFP-Tree*.

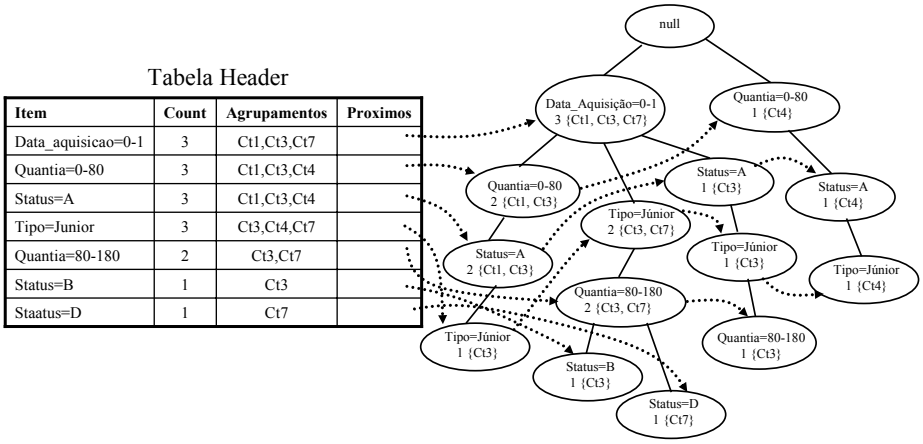

Figura 4.4 *GFP-Tree* e sua tabela header

Uma observação deve ser feita com relação aos contadores presentes na figura 4.4. Pode-se observar que o item *Quantia=80-180* possui contador 2 na tabela header, enquanto na *GFP-Tree* o mesmo ocorre em dois nós, um com contador 2 e outro com contador 1, o que resultaria em um total de 3 ocorrências. No entanto, esse item ocorreu em 2 agrupamentos *{Ct3, Ct7},* e portanto seu número correto de ocorrências é 2, que corresponde ao valor mostrado na tabela header.

Dessa forma, o armazenamento dos agrupamentos dos itens nos nós é importante para descobrir o número de ocorrências totais de um item isoladamente ou em conjunto com os outros itens do ramo. Para preencher corretamente o contador do item na tabela header (count), é feita a união dos agrupamentos de todos os nós nos quais o item em questão ocorreu, utilizando para isso os ponteiros *próximo* da própria tabela header e os ponteiros

próximos de cada nó. Já para calcular o *sup_a* de um itemset é realizada a intersecção dos conjuntos de agrupamentos dos itens de um mesmo ramo.

Por exemplo, o item *Tipo=Júnior* ocorre em quatro nós, sendo que seus agrupamentos são: {*Ct3*}, {*Ct3, Ct7*}, {*Ct3*} e {*Ct4*}. Para preencher seus agrupamentos na tabela header é feita a união dos agrupamentos, ou seja, {*Ct3*} ∪ {*Ct3, Ct7*} ∪ {*Ct3*} ∪ {*Ct4*} = {*Ct3, Ct4, Ct7*}. Para calcular seu contador na tabela header, calcula-se o comprimento do conjunto de agrupamentos, isto é, |{*Ct3, Ct4, Ct7*}|=3.

Para calcular os agrupamentos em que o itemset *{data_aquisição=0-1, status=A, tipo=Júnior}* ocorre, faz-se a intersecção de seus agrupamentos. Dessa forma, *{Ct1, Ct3, Ct7}* ∩ *{Ct1, Ct3, Ct4}* ∩ *{Ct3, Ct4, Ct7}= {Ct3}*. A partir daí, basta calcular o comprimento do conjunto de agrupamentos resultante para obter o número de ocorrências desse itemset, e calcular *sup_a.* Nesse caso, *|{Ct3}|=1*, portanto *sup_a=1/4=25%.*

Um exemplo de execução do algoritmo GFP-Growth pode ser encontrado no Apêndice B.

4.6. Resultados Experimentais

As regras encontradas pelo GFP-Growth diferem do FP-Growth pelo fato de que a mineração de dados é feita considerando os agrupamentos formados por uma entidade pré-definida, enquanto o FP-Growth considera as transações individualmente. Dessa forma, se os dois algoritmos forem aplicados sobre uma mesma base, regras idênticas podem ser encontradas, porém os valores das medidas de interesse suporte e confiança podem ser diferentes.

Para exemplificar essa situação, considere a base da figura 4.1. Deseja-se verificar se existe alguma relação entre a freqüência com que os clientes movimentam suas contas e o valor e a caracterização das ordens de pagamento vinculadas a essas contas. Para isso uma parcela dos dados das tabelas *Conta* e *Ordem de Pagamento,* extraídos de *http://lisp.vse.cz/challenge/index.html* foi selecionada e a operação *full outer join* foi realizada para deixar a tabela no formato necessário para aplicação do algoritmo GFP-Growth, conforme mostrado na tabela 4.7. A base pré-processada resultou em 7115 transações compreendendo 3758 agrupamentos, o que indica que 3758 contas possuíam ordens de pagamento vinculadas.

Um exemplo de regra encontrada pelo GFP-Growth é a seguinte: *quantia=5-10→ freqüência=mensal*, *sup_a=29%, conf_a=90%, peso(quantia=5-10)=100.0%* e

peso(freqüência=mensal)=83.8%. O que indica que as contas que possuem ordens de pagamento no valor de R$5.000 a R$10.000 tendem a ser movimentadas mensalmente.

Tabela 4.7 Contas e ordens de pagamento relacionadas

IdConta	Freqüência	Quantia	Caracterização
492	mensal	null	Null
493	mensal	null	Null
494	mensal	0-1	pay_other
494	mensal	2-3	pay_leasing
494	mensal	5-10	pay_household
496	mensal	3-5	pay_household
497	mensal	3-5	pay_household
498	mensal	null	Null
500	mensal	0-1	pay_other
500	mensal	2-3	pay_leasing
500	mensal	3-5	pay_household

Além disso, pode-se observar que esse padrão foi encontrado em 29% das contas da base e 90% das contas que possuem ordens de pagamento nos valores de R$5.000 a R$10.000 são movimentadas mensalmente. Os valores de peso indicam que os valores *quantia=5-10* foram usados em sua totalidade, enquanto que foram usados 83,8% dos valores *freqüência=mensal*. Isso ocorre porque algumas contas não possuem ordens de pagamento vinculadas, com isso seus blocos não formam agrupamento.

Se usarmos a mesma base para aplicação do FP-Growth, a mesma regra será encontrada da seguinte forma: *quantia=5-10→ freqüência=mensal, suporte = 16.4%, confiança = 89.69%*. Os valores diferentes para suporte e confiança ocorrem porque o FP-Growth não considera os agrupamentos formados pelas contas, e conseqüentemente não desconsidera as duplicações de dados nem descarta os blocos que não formam segmentos. Ou seja, esse algoritmo minera regras considerando as transações, o que indica que a regra acima foi encontrada em 16.4% das transações totais da base, e que o item *freqüência=mensal* ocorreu em 89.69% das transações nas quais *quantia=5-10* também ocorreu.

Como o FP-Growth não considera que os dados estão agrupados por contas, é necessário que a base seja preparada de uma forma diferente. Para isso, é necessário eliminar os valores nulos e o identificador das contas, que não deve aparecer nas regras. A tabela final compreende 6373 transações.

Ao aplicar o algoritmo FP-Growth nessa nova base, a regra analisada será encontrada da seguinte forma *quantia=5-10 → freqüência=mensal, suporte = 18.3% e confiança = 89.69%.* Nesse caso, os valores de suporte e confiança aumentaram um pouco devido à diminuição do número de tuplas considerado.

Considerando as bases analisadas pelo FP-Growth e pelo GFP-Growth pode-se dizer que as mesmas são diferentes. A primeira é resultado de uma operação de *inner join* sobre as tabelas originais, o que leva à eliminação de uma parcela dos dados que não possua dados relacionados em outra tabela. Outro ponto importante é que o FP-Growth não leva em conta a entidade a ser analisada e, portanto não possui seu identificador.

A base analisada pelo GFP-Growth possui o identificador da entidade analisada e o utiliza no processo de mineração. Essa base é resultado de uma operação de *full outer join*, o que origina valores nulos, duplicações e não elimina nenhum os dados que não possuem relação com as outras tabelas. Essa base costuma ser bem maior que a base analisada pelo FP-Growth.

Quando os dados das várias tabelas analisadas possuem grande relação entre si, a tabela usada pelo FP-Growth descartará uma pequena parcela dos dados e a tabela usada pelo GFP-Growth possuirá uma pequena quantidade de valores nulos. Isso faz com que essas bases de dados não apresentem grandes diferenças. Quanto menor a relação entre os dados presentes nas tabelas analisadas, maior será a diferença encontrada entre as bases analisadas pelos dois algoritmos considerados.

Quanto mais próximas forem as bases consideradas, menor será a diferença entre as medidas suporte e confiança encontrados por ambos os algoritmos. Nesse caso, pode-se observar que uma conta pode possuir nenhuma ou várias ordens de pagamento vinculadas. Isso torna a base analisada pelo GFP-Growth muito diferente da base analisada pelo FP-Growth. Se as contas possuíssem apenas uma ordem de pagamento vinculada, os valores de suporte e confiança encontrados seriam os mesmos encontrados pelo FP-Growth.

4.6.1. Comparações com Outros Algoritmos

O algoritmo GFP-Growth foi submetido a testes juntamente com os algoritmos FP-Growth e Connection, com o objetivo de comparar seus tempos de execução e numero de regras geradas. Foram selecionados dois conjuntos de dados extraídos da base de dados de informações bancárias, obtidas em *http://lisp.vse.cz/challenge/index.html.*

O primeiro conjunto refere-se a uma parcela dos dados das tabelas *Conta* e *Ordem de Pagamento*. As informações desse conjunto envolvem a freqüência de movimentação da conta e sua data de criação. Alem disso, são tratados a quantia da ordem de pagamento e sua caracterização. Esse conjunto compreende 4500 contas e 6373 ordens de pagamento.

O segundo conjunto refere-se a uma parcela dos dados das tabelas *Conta* e *Empréstimo*. As informações sobre as contas são as mesmas do primeiro conjunto, enquanto que os

empréstimos envolvem informações sobre sua duração, status e quantia requerida, compreendendo 682 empréstimos.

Os testes foram realizados utilizando um processador Intel Pentium M 1.73GHz com 2MB de cache L2, 1GB de memória *RAM DDR2-SDRAM* de 532MHz, com sistema operacional Windows *XP/2002 Home*.

4.6.1.1. Comparação com o FP-Growth

Com o objetivo de comparar seus tempos de execução, os algoritmos GFP-Growth e FP-Growth foram aplicados sobre a tabela resultante da junção das tabelas *Conta* e *Ordem de Serviços*. É importante ressaltar que os dois algoritmos tratam problemas diferentes e por isso é esperada uma diferença em seus tempos de execução. Além disso, essa base não produz resultados confiáveis para o FP-Growth, como já foi explicado anteriormente, portanto a análise tem como objetivo apenas a comparação do tempo de execução dos dois algoritmos sobre uma base de mesmo tamanho para mostrar que o algoritmo GFP-Growth pode ser aplicado na prática. Para isso, o valor das medidas de interesse *sup_a* e suporte foram variados, e os tempos de execução dos algoritmos foram medidos e pautados em um gráfico, como se pode observar na figura 4.5.

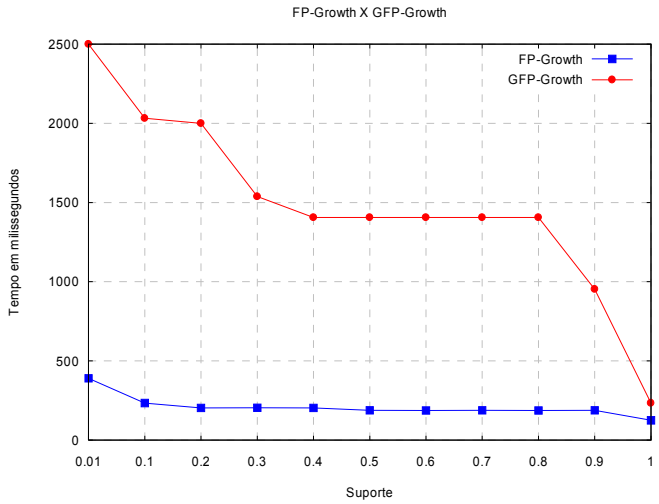

Figura 4.5 Tempo de execução dos algoritmos

Nesse gráfico pode-se observar que o tempo de execução do algoritmo GFP-Growth é sempre superior ao tempo do FP-Growth, no entanto a diferença entre os tempos diminui à medida que se aumenta o suporte. Isso se deve ao fato de que para suportes menores, existe

uma grande quantidade de agrupamentos a serem considerados, o que toma uma parcela considerável do tempo de execução do GFP-Growth.

A fase de identificação dos agrupamentos é realizada no início do algoritmo independente dos valores das medidas de interesse utilizadas. Na base utilizada para a análise em questão, a fase de identificação dos segmentos levou 140 milissegundos, tanto para valores de suporte altos como para valores baixos. Esse tempo é maior que o tempo total de execução do FP-Growth com suporte de 100%, que corresponde a 125 milissegundos.

O conjunto de dados analisado compreende 7115 tuplas, sendo que 742 tuplas possuem valores nulos. Nesse caso, foram encontrados 3758 agrupamentos, o que fez com que o GFP-Growth gerasse regras a partir de praticamente 90% da base de dados analisada.

Ao considerarmos a base contendo informações sobre contas e empréstimos, considerando que poucas contas possuem empréstimos relacionados, o resultado será diferente. Nesse conjunto, 10% da base de dados analisada contribui para a geração de regras, diminuindo o tempo de execução do GFP-Growth e aproximando-o do tempo do FP-Growth, como se pode verificar na figura 4.6.

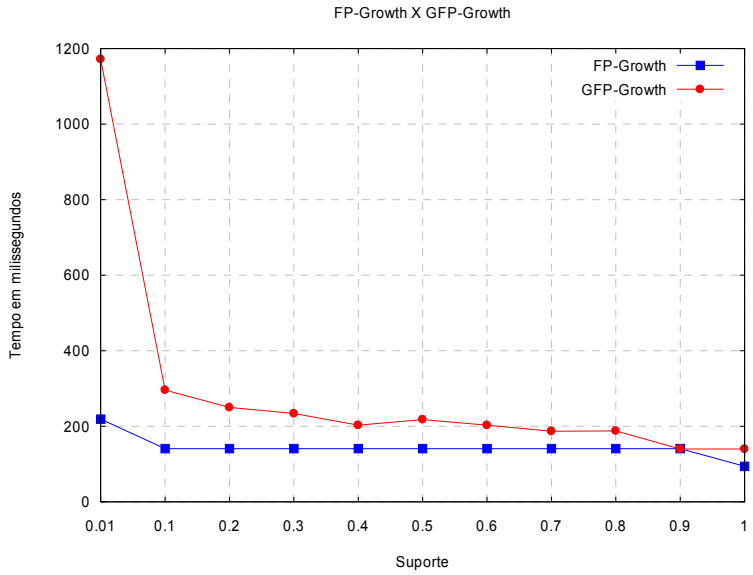

Figura 4.6 Tempo de execução dos algoritmos

O fato do algoritmo GFP-Growth apresentar um tempo de execução maior que o algoritmo GFP-Growth em ambas as bases de dados analisadas, deve-se ao fato de que as bases mineradas não são analisadas da mesma forma, já que não representam o mesmo contexto, e conseqüentemente as regras encontradas são diferentes. Assim, a aplicação dos dois algoritmos considerando o mesmo valor de suporte pode gerar um número diferente de regras e as regras encontradas possuem valores de medidas de interesse em relação aos

agrupamentos encontrados, ao contrário do FP-Growth, que considera todas as transações da base.

Na tabela 4.8 é representado o número de regras obtido a partir da análise das bases de dados consideradas para cada algoritmo variando seu suporte. Nessa analise a tabela minerada pelo algoritmo FP-Growth é resultado da junção das tabelas envolvidas, de forma que os valores *null* são eliminados.

Tabela 4.8 Número de regras encontradas

Suporte	Contas X Ordem de Pagamento		Contas X Empréstimo	
	GFP-Growth	FP-Growth	GFP-Growth	FP-Growth
1%	757	581	1698	1698
10%	79	52	11	11
20%	40	8	19	19
30%	10	2	2	2
40%	2	2	2	2
50%	2	2	0	0
60%	2	0	0	0
70%	2	0	0	0
80%	2	0	0	0
90%	0	0	0	0
100%	0	0	0	0

Na tabela pode-se observar que para a primeira base de dados analisada o GFP-Growth encontra um número maior de regras que o FP-Growth para suportes até 30% e a maior discrepância entre os algoritmos é observada para o suporte de 20%, para o qual o GFP-Growth encontra 24 regras, enquanto o FP-Growth encontra 4 regras. Esse número maior de regras encontradas também justifica a diferença nos tempos de execução dos dois algoritmos.

Além disso, o GFP-Growth consegue obter regras para valores de *sup_a* mais elevados que o suporte do FP-Growth. Por exemplo, o maior suporte obtido pelas regras encontradas pelo FP-Growth corresponde a 51%, enquanto que pelo GFP-Growth corresponde a 85%.

Ao analisarmos a segunda base de dados nota-se que o número de regras encontradas é o mesmo. Porém, isso não significa que o resultado da aplicação desses dois algoritmos é igual, pois as medidas de interesse das regras devem ser igualmente consideradas, conforme discutido no início desta seção.

4.6.1.2. Comparação com o Algoritmo Connection

Dando continuidade à análise do algoritmo GFP-Growth, foi realizada uma comparação com o algoritmo Connection, sendo que seus tempos de execução e número de regras geradas foram analisados. Para comparar seus tempos de execução, os algoritmos foram aplicados

inicialmente sobre uma base de dados contendo informações sobre contas e ordens de pagamentos vinculadas, e posteriormente sobre uma base de dados com informações sobre contas e empréstimos vinculados a essas contas.

O conjunto de entrada do algoritmo GFP-Growth é formado pela tabela resultante da junção das tabelas *Contas* e *Ordens de Pagamento* da figura 4.1, e representa a mesma tabela usada na comparação da seção anterior. Já o conjunto de entrada do algoritmo Connection é formado pelas tabelas *Contas* e *Ordens de Pagamento*, com 4500 e 6373 tuplas respectivamente.

Dessa forma, os algoritmos foram aplicados sobre seus conjuntos de entrada e seus tempos de execução foram pautados no gráfico mostrado na figura 4.7. Neste gráfico pode-se observar que o tempo de execução do algoritmo GFP-Growth é superior ao tempo do Connection para o maior número de medidas de suporte.

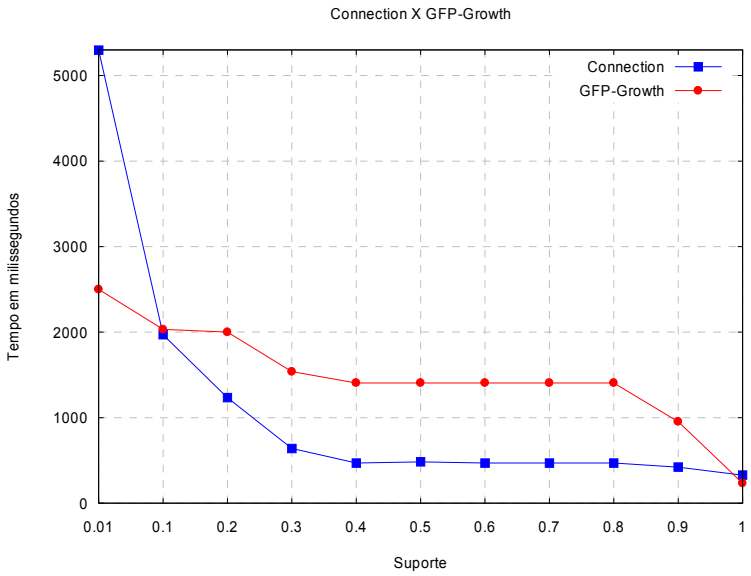

Figura 4.7 Tempo de execução dos algoritmos

Isso ocorre porque nessa base de dados, a quantidade de itens eliminados no início do processo de mineração pelo GFP-Growth, por não contribuir para a geração de regras, foi pequena, conforme discutido na seção anterior. Nesse caso, o custo para varrer a tabela resultante da junção usada pelo GFP-Growth é maior do que o custo para relacionar duas tabelas separadas, realizado pelo Connection.

No entanto, é importante analisar o segundo conjunto de entrada considerado, conforme foi feito na seção anterior com o FP-Growth. Os algoritmos GFP-Growth e Connection foram aplicados ao conjunto de dados com informações sobre contas e empréstimos, e seus tempos de execução podem ser observados no gráfico mostrado na figura 4.8.

Nesse gráfico pode-se observar que o tempo de execução do algoritmo GFP-Growth é inferior ao tempo do algoritmo Connection. Isso ocorre porque nesse caso a eliminação dos blocos que não formam agrupamentos diminui consideravelmente o espaço de busca, fazendo com que o custo de análise dos agrupamentos seja menor que o custo de relacionamento das tabelas envolvidas no Connection.

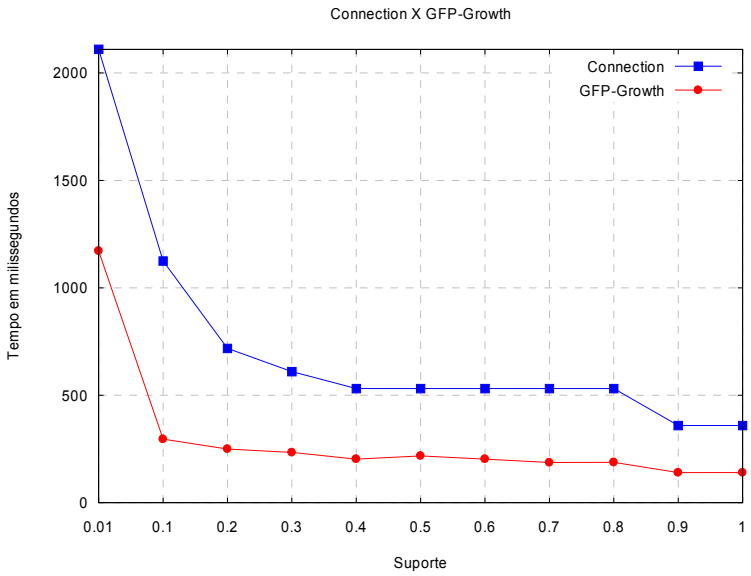

Figura 4.8 Tempo de execução dos algoritmos

Também foram feitas comparações quanto ao número de regras geradas pelos algoritmos GFP-Growth e Connection. As regras geradas por ambos os algoritmos possuem algumas diferenças quanto à sua estrutura e quanto à origem dos itens.

As regras geradas pelo GFP-Growth podem envolver itens de uma única tabela ou de varias tabelas analisadas. Quanto à sua estrutura, o conseqüente da regra sempre possui 1 item, conforme ocorre com o FP-Growth. O algoritmo Connection sempre gera regras que envolvem itens de tabelas diferentes, sendo que itens da mesma tabela não ocorrem parte no conseqüente e parte no antecedente da regra. Alem disso, as regras podem conter *itemsets* em seu conseqüente.

Na tabela 4.9 é representado o número de regras obtido a partir da análise das bases de dados variando o suporte. Nessa análise as tabelas mineradas pelos algoritmos são diferentes, pois o Connection minera duas tabelas separadas, enquanto o GFP-Growth minera apenas uma contendo informações sobre as duas tabelas em questão.

Na tabela pode-se observar que o GFP-Growth encontra um número maior de regras que o Connection para suportes até 30% e a maior discrepância entre os algoritmos é observada

para o suporte de 1%. Uma das razões para esse número diferente de regras é o fato do GFP-Growth encontrar regras que envolvem itens de uma única tabela.

Tabela 4.9 Número de regras geradas

Suporte	Contas X Ordem de Pagamento		Contas X Empréstimo	
	Connection	GFP-Growth	Connection	GFP-Growth
1%	606	757	756	1698
10%	50	79	58	111
20%	26	40	14	19
30%	4	10	2	2
40%	2	2	2	2
50%	2	2	0	0
60%	2	2	0	0
70%	2	2	0	0
80%	2	2	0	0
90%	0	0	0	0
100%	0	0	0	0

4.6.2. Análise do Peso

Para avaliar o efeito da medida de interesse peso, o algoritmo GFP-Growth foi aplicado a uma base de dados e o tempo de execução e o número de regras geradas são mostrados nas figuras 4.9 e 4.10.

A base utilizada compreende informações sobre freqüência de movimentação de contas, idade e sexo dos clientes dessas contas, além da quantia e status dos empréstimos vinculados a essas contas. Essa tabela compreende 4500 tuplas. A base analisada tem o formato da tabela 4.10.

Tabela 4.10 Contas e empréstimos relacionados

IdConta	Sexo	Idade	Freqüência	Quantia	Status
11328	F	45-60	mensal	180-600	C
11333	M	45-60	mensal	null	null
11349	F	45-60	semanal	180-600	C
11349	M	45-60	semanal	180-600	C
11359	M	30-45	mensal	0-80	A
11362	F	30-45	mensal	80-180	A
11382	F	45-60	mensal	null	null

Nessa tabela, pode-se observar que algumas contas possuem campos nulos em seus agrupamentos, o que indica que essas contas não possuem empréstimos vinculados, já os valores para a freqüência de movimentação, idade e sexo dos clientes estão sempre presentes. Nesse caso, os valores da medida peso para os itens dos campos status e quantia serão sempre 100%, visto que seus valores nunca ocorrerão em um bloco que não forma agrupamento. Já os

valores da freqüência de movimentação, idade e sexo dos clientes titulares das contas terão peso diferente de 100%, pois ocorrem tanto em agrupamentos quanto em blocos que não formam agrupamentos.

A figura 4.9 mostra que o número de regras obtidas pelo GFP-Growth diminui à medida que se aumenta o valor da medida de interesse peso. Isso ocorre porque um itemset freqüente precisa satisfazer o mínimo suporte e peso, assim sendo não basta o itemset satisfazer apenas o mínimo suporte pré-estabelecido, pois ainda é necessário que o mesmo possua alguma relação com os dados das outras tabelas que estão sendo analisadas.

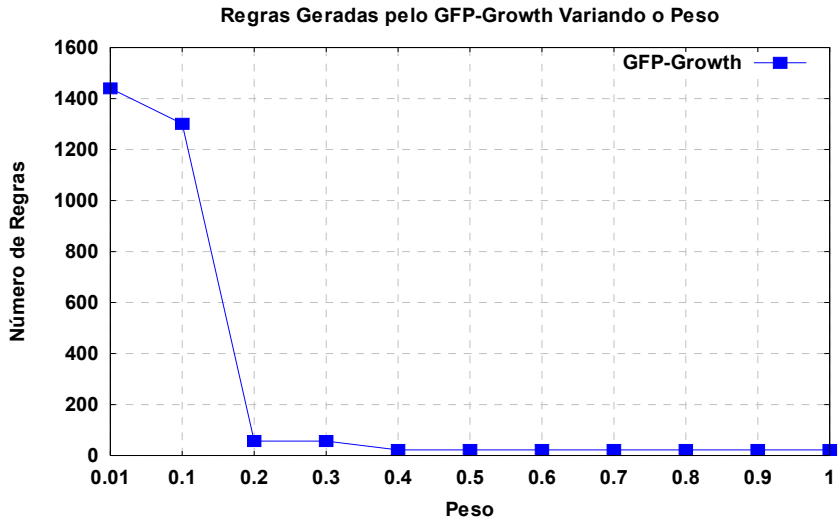

Figura 4.9 Número de regras obtidas variando o peso

A partir da figura 4.9 obtém-se 1440 regras com pelo menos 1% de utilização das ocorrências de seus itens no processo de mineração. Ao compararmos com o valor de peso 20%, obtém-se 57 regras cujos itens estavam relacionados às outras tabelas analisadas em 20% de suas ocorrências.

Um exemplo de regra encontrada com peso maior que 20% é: *quantia=180-600 freqüência=semanal → status=C, sup_a=3.6% conf_a=67.5%, peso(quantia=180-600) = 100.0%, peso(freqüência=semanal) = 34.2% e peso(status=C) = 100.0%.* Já a regra *idade=60-80 -> quantia=0-80, sup_a=3.6%, conf_a=40.9%, peso(quantia=0-80)=100.0% e peso(idade=60-80)=4.8%*, não é encontrada para peso maior que 20%, pois um item dessa regra não satisfaz esse peso mínimo.

Como a medida peso limita a quantidade de regras encontradas, isso influi no tempo de execução do algoritmo. Na figura 4.10 pode-se observar uma curva semelhante à mostrada na figura 4.9, o que indica que o tempo de execução do algoritmo diminui na mesma proporção que o número de regras.

Além disso, pode-se observar que o tempo se mantém constante quando o número de regras encontradas também é constante, o que ocorre para os valores de peso de 20% a 30%, para os quais são encontradas 57 regras em 344 milissegundos, e para os valores de 40% a 100%, em que são encontradas 22 regras em milissegundos.

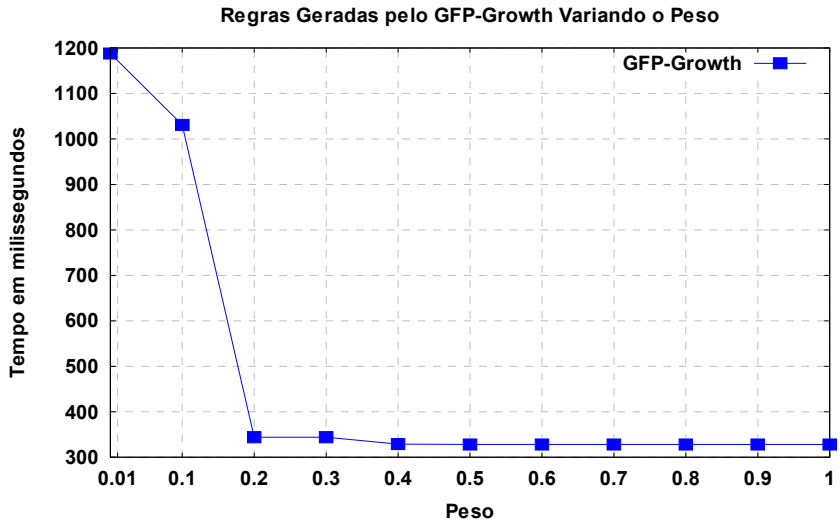

Figura 4.10 Tempo de execução variando o peso

A maior variação do número de regras encontradas ocorreu quando o peso variou de 10% para 20%, sendo que o número de regras que era 1330 passou para 57, conforme pode ser observado na figura 4.9. Nesse caso, pode-se observar que a curva mais acentuada para esses valores na figura 4.9 ocorre do mesmo modo na figura 4.10.

4.7. Considerações Finais

Este capítulo apresentou o algoritmo GFP-Growth, desenvolvido como resultado deste trabalho de mestrado. Esse algoritmo foi proposto com o intuito de aumentar o número de regras descobertas durante o processo de mineração, permitindo descobrir regras que envolvam itens de várias tabelas, além daquelas que envolvam itens de uma única tabela.

Os testes realizados comprovam a descoberta de regras adicionais, além de apontar um desempenho melhor que o apresentado pelo algoritmo Connection em determinadas situações.

O próximo capítulo apresenta as contribuições deste trabalho e sugestões de trabalhos futuros, incluindo algumas possibilidades para aperfeiçoamento do algoritmo GFP-Growth.

5. Conclusões

5.1. Considerações Iniciais

Este trabalho apresentou um novo algoritmo para mineração multi-relacional de regras de associação que pode ser aplicado em análises envolvendo múltiplas tabelas relacionadas de um banco de dados.

Foram apresentados os conceitos básicos sobre mineração de dados e a tarefa de associação foi detalhada. Além disso, foram explorados alguns algoritmos de mineração de regras de associação tradicionais e multi-relacionais, que foram importantes para direcionar a realização deste trabalho.

5.2. Contribuições

A principal contribuição deste trabalho foi a definição e implementação do algoritmo GFP-Growth que tornou possível a mineração multi-relacional de tabelas relacionadas, a partir da tabela resultante da junção das mesmas. Ao contrário das técnicas tradicionais de mineração, o GFP-Growth trata a redundância nos dados e as possíveis inconsistências nas medidas de interesse introduzidas pela operação de junção.

Considerando os agrupamentos presentes nos dados, o algoritmo possibilita descobrir regras envolvendo uma determinada entidade, ao invés de analisar todas as tuplas disponíveis como se as mesmas fossem independentes, como é feito pela maioria das técnicas de mineração encontradas.

As medidas de interesse utilizadas pelo algoritmo GFP-Growth foram ajustadas para reconhecer as entidades que são relevantes em várias tabelas, e dessa forma puderam retratar corretamente o comportamento dos itens da base de dados analisada.

5.3. Trabalhos Futuros

5.3.1. Melhorar o desempenho

A partir dos testes realizados constatou-se que o desempenho do GFP-Growth é inferior ao algoritmo FP-Growth pelas razões já explicitadas no capítulo 4, o que pode ser considerado uma conseqüência da semântica embutida nos dados utilizados pelo GFP-Growth. Apesar da análise realizada pelos dois algoritmos envolver conjuntos de entrada e de saída diferentes e servirem para diferentes propósitos, seria interessante melhorar o desempenho do GFP-

Growth. Alguns pontos que poderiam ser levados em conta são a minimização de estruturas temporárias e a substituição de algumas estruturas de dados.

A tabela minerada pelo GFP-Growth pode ter um tamanho elevado devido à operação de junção, por isso pode-se analisar uma outra forma de diminuir seu tamanho para reduzir o espaço de busca do algoritmo. Uma alternativa seria adotar um formato dos dados semelhante ao mostrado na tabela 5.1, que equivale à tabela 4.2 sem as duplicações de dados.

Tabela 5.1 Dados agrupados por contas

IdConta	Cartão	Empréstimo
Ct1	{(classic, 0-1)}	{(0-80, A)}
Ct2	{(classic, 1-3)}	{ }
Ct3	{(júnior, 0-1)}	{(80-180, B) (0-80, A) (80-180, A)}
Ct4	{(júnior, 1-3) (gold, 1-3)}	{(0-80, A)}
Ct5	{(gold, 3-5)}	{ }
Ct6	{ }	{(180-600, C) (0-80, A)}
Ct7	{(júnior, 0-1)}	{(80-180, D) (180-600, C)}
Ct8	{(gold, 1-3)}	{ }

5.3.2. Desenvolver uma Ferramenta para Mineração de Dados

Considerando o algoritmo desenvolvido neste trabalho e os algoritmos analisados no levantamento bibliográfico, nota-se uma grande influência do algoritmo FP-Growth. Assim, é possível classificar esses algoritmos derivados do FP-Growth em uma "*família de algoritmos*", levando em conta que cada algoritmo comporta-se melhor em determinada situação.

Para que o usuário possa se beneficiar das vantagens desses algoritmos, seria interessante desenvolver uma ferramenta de testes que disponibilize uma "*família de algoritmos*" ao usuário e o mesmo possa escolher com facilidade qual algoritmo deseja utilizar.

O usuário poderia visualizar o resultado obtido pela aplicação de vários algoritmos sobre a base e escolher qual deles satisfaz seu problema. Outra opção seria usar os vários resultados para uma análise mais abrangente da base de dados em questão.

5.3.3. Estender a Análise

A estratégia desenvolvida neste trabalho é direcionada para a tarefa de associação e considera apenas dados discretos. Seria interessante estender essa análise para outras tarefas de mineração, como classificação e padrões seqüenciais. Isso permitiria que fossem aplicados

sobre um único conjunto de entrada, algoritmos que produzissem diferentes saídas, permitindo ao usuário escolher qual tarefa permitiria uma análise mais apurada de seu problema.

O algoritmo em questão necessita de uma etapa preliminar de discretização dos dados numéricos, o que é feito pelo usuário. A escolha dos intervalos de valores é subjetiva, podendo evitar a descoberta de alguns padrões interessantes caso o usuário não conheça o domínio do problema. Estender o algoritmo para o tratamento de dados quantitativos permitiria uma análise mais consistente dos dados.

5.3.4. Propor Novas Medidas de Interesse

Neste trabalho foram usadas medidas de interesse anteriormente propostas para tratar o problema de mineração multi-relacional de regras de associação. A medida peso permite dimensionar a quantidade de dados que é descartada quando se relaciona várias tabelas, no entanto seria interessante a adoção de novas medidas que possibilitassem analisar outros parâmetros importantes envolvendo múltiplas tabelas.

5.3.5. Realizar Baterias de Testes

Os testes realizados neste trabalho tiveram como objetivo validar o uso do algoritmo e comparar se o mesmo mantinha um padrão de comportamento semelhante ao FP-Growth e ao Connection. Para isso foram considerados o tempo de execução e o número de regras obtidas.

No entanto, seria interessante realizar análises considerando outros parâmetros ou ainda outros algoritmos multi-relacionais. Essas análises poderiam indicar os pontos falhos do algoritmo e direcionar o desenvolvimento de melhorias ou até mesmo de uma nova estratégia.

5.3.6. Propor uma Nova Estratégia de Mineração

Em um banco de dados podem existir tabelas que não estão diretamente relacionadas pela modelagem, mas apresentam uma relação semântica entre si. Oferecer recursos para a extração de padrões em tabelas semanticamente relacionadas pode trazer vários benefícios, e para isso é necessário o desenvolvimento de novas abordagens para a mineração dos dados e para a visualização dos padrões encontrados.

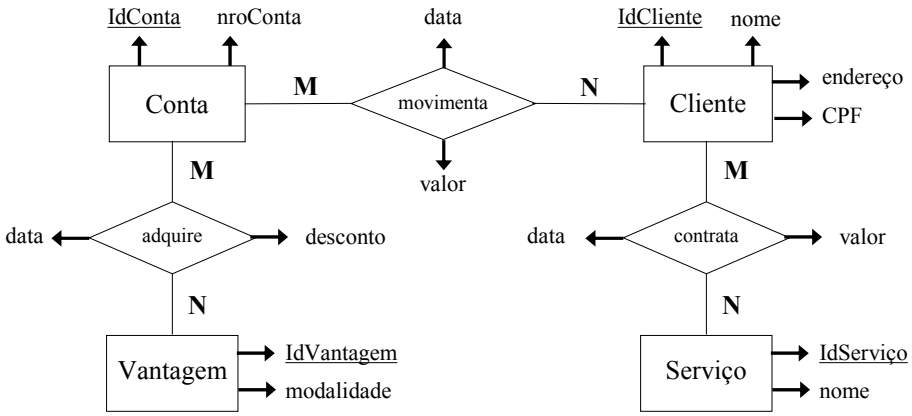

Figura 5.1 Informações bancárias

Na figura 5.1 uma análise interessante seria descobrir se existe alguma relação entre as vantagens oferecidas aos clientes e os serviços contratados por esses clientes. As tabelas *Vantagem* e *Serviço* não compartilham uma chave (primária ou estrangeira), portanto não podem ser mineradas com o GFP-Growth. No entanto, existe uma relação semântica entre essas tabelas.

Desse modo, uma melhoria no processo de mineração multi-relacional seria o desenvolvimento de técnicas para minerar regras em tabelas cuja semântica envolvida favoreça a sua análise em conjunto.

6. Referências Bibliográficas

AGRAWAL, R., EVFIMIEVSK, A. e SRIKANT, R. **Information sharing across private databases**. In: ACM SIGMOD international conference on Management of data, 2003, San Diego, USA. **Anais.** San Diego, USA, 2003. p. 86-97.

AGRAWAL, R., IMIELINSKI, T. e SWAMI, A. **Mining association rules between sets of items in large databases**. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data, 1993, Washington, D.C., USA. **Anais.** Washington, D.C., USA, 1993. p. 207-216.

AGRAWAL, R. e RAMAKRISHNAN, S. **Privacy-preserving Data Mining**. In: Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of Data, 2000, Dallas, Texas, USA. **Anais.** Dallas, Texas, USA, 2000.

AGRAWAL, R. e SRIKANT, R. **Fast algorithms for mining association rules**. In: Proc. of the Int'l Conf. on Very Large Databases, 1994, Santiago de Chile, Chile. **Anais.** Santiago de Chile, Chile, 1994.

_____ **Mining Sequential Patterns**. In: Eleventh International Conference on Data Engineering, 1995, Taipei, Taiwan. **Anais.** Taipei, Taiwan, 1995. p. 3-14.

ATRAMENTOV, A., LEIVA, H. e HONAVAR, V. **A Multi-Relational Decision Tree Learning Algorithm - Implementation and Experiments**. In: Proc. of the 13th International Conference on Inductive Logic Programming (ILP 2003), 2003, Szeged, Hungary. **Anais.** Szeged, Hungary, 2003. p. 38 - 56.

BLOCKEEL, H. e SEBAG, M. **Scalability and efficiency in multi-relational data mining**. ACM SIGKDD Explorations Newsletter, v. 5, n. 1, p. 17-30, 2003.

CHAUDHURI, S. e DAYAL, U. **An Overview of Data Warehousing and OLAP Technology**. In: ACM SIGMOD, 1997, **Anais.** 1997.

CHEUNG, D. W.*, et al.* **A fast distributed algorithm for mining association rules**. In: Fourth International Conference on Parallel and Distributed Information Systems, 1996, Miami Beach, Florida, USA. **Anais.** Miami Beach, Florida, USA, 1996. p. 31-42.

CLARE, A., WILLIAMS, H. E. e LESTER, N. **Scalable Multi-Relational Association Mining**. In: Proc. of the 4th IEEE International Conference on Data Mining (ICDM'04), 2004, Brighton, UK. **Anais.** Brighton, UK, 2004. p. 355-358.

DEHASPE, L. e TOIVONEN, H. **Discovery of frequent Datalog patterns**. In: Data Mining and Knowledge Discovery, 1999, **Anais.** 1999. p. 7-36.

_____ **Discovery of Relational Association Rules**. New York, NY, USA: Springer-Verlag New York, Inc, 2001.

DESHAPE, L. e RAEDT, L. **Mining association rules in multiple relations**. In: Proc. of the 7th International Workshop on Inductive Logic Programming, 1997, Prague, Czech Republic. **Anais.** Prague, Czech Republic, 1997. p. 125-132.

DING, Q. e PARIKH, B. **A Model for Multi-relational Data Mining on Demand Forecasting**. In: 13th International Conference on Intelligent and Adaptive Systems and Software Engineering, 2004, Nice, France. **Anais.** Nice, France, 2004. p. 1-5.

DOMINGOS, P. **Prospects and Challenges for Multi-Relational Data Mining**. ACM SIGKDD Explorations Newsletter, v. 5, n. 1, p. 80-83, 2003.

DžEROSKI, S. O. **Multi-relational data mining: an introduction**. ACM SIGKDD Explorations Newsletter, v. 5, n. 1, p. 1 - 16, 2003.

DžEROSKI, S. O. e ŽENKO, B. **A Report on the Summer School on Relational Data Mining**. ACM SIGKDD Explorations Newsletter, v. 5, n. 1, p. 100-101, 2002.

DžEROSKI, S. O. e RAEDT, L. **Multi-relational Data Mining: a Workshop Report**. ACM SIGKDD Explorations Newsletter, v. 4, n. 2, p. 122-124, 2002.

DžEROSKI, S. O., RAEDT, L. e WROBEL, S. **Multi-relational Data Mining 2003: Workshop Report**. ACM SIGKDD Explorations Newsletter, v. 5, n. 2, p. 200-202, 2003.

FAYYAD, U., PIATETSKY-SHAPIRO, G. e SMYTH, P. **From Data Mining to Knowledge Discovery in Databases**. 1996.

FREITAS, A. A. **Understanding the crucial differences between classification and discovery of association rules: a position paper**. ACM SIGKDD Explorations Newsletter, v. 2, n. 1, p. 65 - 69, June 2000, 2000.

HAN, J. e KAMBER, M. **Data Mining - Concepts and Techniques**. 1a edição. Nova York: Morgan Kaufmann, 2001.

HAN, J., PEI, J. e YIN, Y. **Mining frequent patterns without candidate generation**. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data, 2000, Dallas, Texas, USA. **Anais.** Dallas, Texas, USA, 2000.

HO, T. B.*, et al.* **Extracting Knowledge from Hepatitis Data with Temporal Abstraction**. In: Proc. of the ECML/PKDD 2002 Discovery Challenge, 2002, Helsinki, Finland. **Anais.** Helsinki, Finland, 2002. p. 19-23.

_____ **Mining Hepatitis Data with Temporal Abstraction**. In: Proc. of the ACM International Conference on Knowledge Discovery and Data Mining KDD-03, 2003, Washington, USA. **Anais.** Washington, USA, 2003. p. 369-377.

IMIELINSKI, T. e MANNILA, H. **A database perspective on knowledge discovery**. Communications of the ACM, v. 39, n. 11, p. 58 - 64, November 1996, 1996.

IVÁNCSY, R., KOVÁCS, F. e VAJK, I. **An Analysis of Association Rule Mining Algorithms**. In: Proc. of the Fourth International ICSC Symposium on Engineering of Intelligent Systems (EIS 2004), 2004, Madeira, Portugal. **Anais.** Madeira, Portugal, 2004. p. 526-532.

JENSEN, V. e SOPARKAR, N. **Frequent Itemset Counting Across Multiple Tables**. In: Proc. of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2000, Kyoto, Japan. **Anais.** Kyoto, Japan, 2000. p. 49-61.

KANODIA, J. **Structural Advances for Pattern Discovery in Multi-Relational Databases**. 2005. Dissertação de Mestrado – Departamento de Ciência da Computação, Rochester Institute of Technology, Rochester, NY, 2005.

KANTARCIOGLU, M. e CLIFTON, C. **Privacy-preserving Distributed Mining of Association Rules on Horizontally Partitioned Data**. In: The ACM SIGMOD Workshop in Research Issues on Data Mining and Knowledge Discovery, 2002, Madison, Wisconsin, USA. **Anais.** Madison, Wisconsin, USA, 2002.

KLEMETTINEN, M.*, et al.* **Finding Interesting Rules from Large Sets of Discovered Association Rules**. In: Proc. of the Third International Conference on Information and Knowledge Management (CIKM'94), 1994, Gaithersburg, Maryland, USA. **Anais.** Gaithersburg, Maryland, USA, 1994.

KNOBBE, A. J., SIEBES, A. e WALLEN, D. V. D. **Multi-relational Decision Tree Induction**. In: Proc. of the 3rd European Conference on Principles of Data Mining and Knowledge Discovery (PKDD), 1999, Prague, Czech Republic. **Anais.** Prague, Czech Republic, 1999. p. 378-383.

LENT, B., SWAMI, A. e WIDOM, J. **Clustering Association Rules**. In: 13th International Conference on Data Engineering (ICDE'97), 1997, Birmingham, U.K. **Anais.** Birmingham, U.K., 1997. p. 220-231.

MATSUDA, T.*, et al.* **Preliminary Analysis of Hepatitis Data by Beam-wise Graph-Based Induction**. In: Proc. of the ECML/PKDD 2002 Discovery Challenge, 2002, Helsinki, Finland. **Anais.** Helsinki, Finland, 2002.

MEO, R., PSAILA, G. e CERI, S. **A New SQL-like Operator for Mining Association Rules**. In: Proceedings of 22th International Conference on Very Large Data Bases, 1996, Bombay, India. **Anais.** Bombay, India, 1996. p. 122-133.

NESTOROV, S. e JUKIC, N. **Ad-Hoc Association-Rule Mining within the Data Warehouse**. In: Proc. of 36th Annual Hawaii International Conference on System Sciences (HICSS'03), 2003, Big Island, Hawaii. **Anais.** Big Island, Hawaii, 2003. p. 232a.

NG, E., FU, A. e WANG, K. **Mining Association Rules from Stars**. In: 2002 IEEE International Conference on Data Mining (ICDM'02), 2002, Maebashi City, Japan. **Anais.** Maebashi City, Japan, 2002. p. 322-329.

OHARA, K.*, et al.* **Analysis of Hepatitis Dataset by Decision Tree Graph-Based Induction**. In: Proc. of the ECML/PKDD2004 Discovery Challenge, 2004, Pisa, Italy. **Anais.** Pisa, Italy, 2004. p. 173-184.

OHSAKI, M.*, et al.* **A rule discovery support system for sequential medical data - in the case study of a chronic hepatitis dataset**. In: Proc. of the ECML/PKDD 2003 Discovery

Challenge, 2003, Cavtat-Dubrovnik, Croatia. **Anais.** Cavtat-Dubrovnik, Croatia, 2003. p. 154-165.

ORDONEZ, C. **A Model for Association Rules Based on Clustering**. In: Proc. of the 2005 ACM Symposium on Applied Computing, 2005, Santa Fe, New Mexico, USA. **Anais.** Santa Fe, New Mexico, USA, 2005. p. 545 - 546.

PIZZI, L. C., RIBEIRO, M. X. e VIEIRA, M. T. P. **Analysis of Hepatitis Dataset using Multirelational Association Rules**. In: Proc. of the ECML/PKDD 2005 Discovery Challenge, 2005, Porto, Portugal. **Anais.** Porto, Portugal, 2005.

PIZZI, L. C. e VIEIRA, M. T. P. **Mineração Multi-Relacional de um Banco de Dados com Informações sobre Hepatite**. São Carlos: UFSCar; Novembro, 2006.

RIBEIRO, M. **Mineração de Dados em Múltiplas Tabelas Fato de Data Warehouse**. 2004. Dissertação de Mestrado – Departamento de Computação, Universidade Federal de São Carlos, São Carlos, SP, 2004.

RIBEIRO, M. X. e VIEIRA, M. T. P. **A New Approach for Mining Association Rules in Data Warehouses**. In: 6th International Conference On Flexible Query Answering Systems, 2004, Lyon, France. **Anais.** Lyon, France, 2004.

RIBEIRO, M. X., VIEIRA, M. T. P. e TRAINA, A. J. M. **Mineração de Regras de Associação Usando Agrupamentos**. In: I Workshop sobre Algoritmos de Mineração de Dados (WAMD'2005), 2005, Uberlândia, MG, Brasil. **Anais.** Uberlândia, MG, Brasil, 2005.

SARAWAGI, S., THOMAS, S. e AGRAWAL, R. **Integrating association rule mining with relational database systems: alternatives and implications**. In: the 1998 ACM SIGMOD international conference on Management of data, 1998, Seattle, Washington, United States. **Anais.** Seattle, Washington, United States, 1998. p. 343-354.

SAVASERE, A., OMIECINSKI, E. e NAVATHE, S. **An Efficient Algorithm for Mining Association Rules in Large Databases**. In: Proc. of the 21st Conf. on Very Large Databases (VLDB'95), 1995, **Anais.** 1995. p. 432-444.

SILBERSCHATZ, A. e TUZHILIN, A. **What Makes Patterns Interesting in Knowledge Discovery Systems**. IEEE Transactions on Knowledge and Data Engineering, p. 970–974, 1996.

TEREDESAI, A., KANODIA, J. e GABORSKI, R. **CoMMA: A Framework for Integrated Multimedia Mining using Multi-relational Associations**. In: Proc. of the 7th ACM SIGKDD Workshop on Multimedia Data Mining, 2004, Seattle, WA, USA. **Anais.** Seattle, WA, USA, 2004.

WATANABE, T.*, et al.* **Application of prototypeline to chronic hepatitis data**. In: Proc. of the ECML/PKDD-2003 Discovery Challenge, 2003, Cavtat-Dubrovnik, Croatia. **Anais.** Cavtat-Dubrovnik, Croatia, 2003. p. 166-177.

YIN, X., HAN, J. e YANG, J. **Efficient Multi-relational Classification by Tuple ID Propagation**. In: Proc. of the 2005 European Conf. on Principles and Practice of Knowledge

Discovery in Databases (PKDD'05), 2005, Porto, Portugal. **Anais.** Porto, Portugal, 2005. p. 122-134.

YOSHIDA, T.*, et al.* **Preliminary Analysis of Interferon Therapy by Graph-Based Induction**. In: Proc. of the Third International Workshop on Active Mining, 2004, Kanazawa, JAPAN. **Anais.** Kanazawa, JAPAN, 2004. p. 31-40.

ZAKI, M.*, et al.* **New algorithms for fast discovery of association rules**. In: Proc. of the ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining, 1997, Newport Beach, CA, USA. **Anais.** Newport Beach, CA, USA, 1997. p. 283-286.

7. Apêndice A - Mineração de Dados

7.1. Considerações Iniciais

Segundo Han e Kamber (HAN; KAMBER, 2001), a mineração de dados pode ser vista como resultado da evolução da tecnologia da informação. Esse processo evolutivo envolve desde a criação dos sistemas gerenciadores de bancos de dados (SGBD's) e seus mecanismos para armazenamento e recuperação de dados, até o desenvolvimento de técnicas de análise de dados, incluindo *data warehouses* (CHAUDHURI; DAYAL, 1997) e mineração de dados.

Neste apêndice é apresentado o processo de descoberta de conhecimento (KDD), concentrando-se na etapa de mineração de dados, além de algumas tarefas de mineração de dados. Alguns conceitos sobre *data warehouses* são apresentados devido à estreita relação destes com o processo de KDD.

7.2. Definição e caracterização

A mineração de dados consiste na extração de conhecimento a partir de grandes quantidades de dados. A motivação para o desenvolvimento de tarefas de mineração de dados é resultado da grande quantidade de dados disponíveis para análise e da carência de mecanismos para tratamento desses dados.

A mineração de dados é considerada a principal fase do processo de descoberta de conhecimento (KDD), em que técnicas de mineração são aplicadas aos dados para obtenção dos padrões desejados. De acordo com Roden, Burl e Fowlkes (1999 apud RIBEIRO, 2004), a mineração de dados consiste em um campo multidisciplinar, envolvendo conceitos de banco de dados, processamento de imagens, estatística e inteligência artificial.

7.3. Processo de descoberta de conhecimento

Segundo Fayyad, Piatetsdy-Shapiro e Smyth (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996), a descoberta de conhecimento é um processo não trivial de identificação de padrões embutidos nos dados que sejam válidos, novos, potencialmente úteis e compreensíveis.

Em (HAN; KAMBER, 2001) o processo de KDD é dividido em várias etapas conforme mostra a Figura 7.1. Essas etapas são:

- Limpeza;
- Integração;
- Seleção;
- Transformação;
- Mineração dos dados;
- Avaliação dos padrões;
- Apresentação.

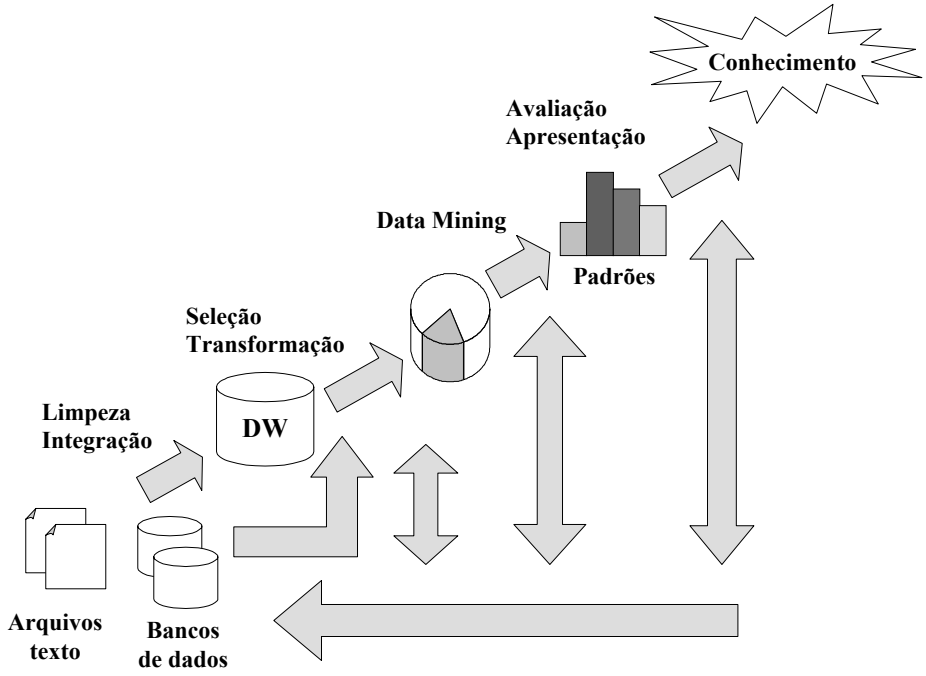

Figura 7.1 - Processo de descoberta de conhecimento (HAN; KAMBER, 2001)

Inicialmente é feita a limpeza dos dados selecionados, o que consiste no tratamento dos dados inconsistentes. O objetivo dessa etapa é impedir a utilização de campos cujo significado seja desconhecido pelo processo de mineração, como campos não preenchidos ou com valores inválidos, os quais podem ser eliminados ou preenchidos com um valor padrão, por exemplo.

Considerando que a etapa anterior realiza operações sobre os dados na própria fonte de origem dos mesmos, a próxima etapa seria a integração dos dados provenientes de múltiplas fontes, que pode ser feita com o auxílio de um DW (*data warehouse*). As fontes de origem dos dados podem ser bases de dados de diferentes formatos ou arquivos de texto. Uma das atividades dessa etapa é a uniformização do formato dos dados, como no caso da integração de dados originários de uma base relacional e de outra orientada a objetos. Outra atividade seria a análise de campos de diferentes origens que possuem nomes diferentes, mas possuem o

mesmo significado, o que pode causar redundância durante a integração. Além disso, é necessário analisar campos com nomes similares, mas que não se referem ao mesmo assunto, o que pode causar inconsistência.

Em uma próxima etapa é feita a seleção dos dados relevantes para análise. Quando a quantidade de dados disponível para análise é muito grande, pode não ser viável efetuar a busca por padrões em toda a base de dados. Dessa forma, é preciso limitar essa quantidade de dados selecionando apenas uma parcela destes, o que pode ser feito de várias formas. Um exemplo seria escolher aleatoriamente algumas tuplas do banco de dados ou selecionar apenas os atributos mais significativos das tabelas.

As duas fases iniciais do processo de KDD, que compreendem a limpeza e integração dos dados, também são realizadas no processo de *data warehousing* (HAN; KAMBER, 2001). Dessa forma, a seleção dos dados pode ser realizada diretamente sobre um DW, fazendo com que o processo de KDD se beneficie do pré-processamento ao qual esse DW foi submetido.

A próxima fase é a de mineração de dados, onde os algoritmos de mineração são aplicados sobre os dados selecionados na etapa anterior. No entanto, ainda é necessário realizar um pré-processamento sobre os mesmos, transformando-os para o formato de entrada do algoritmo a ser aplicado.

A aplicação dos algoritmos de mineração pode gerar uma grande quantidade de padrões, sendo necessário estabelecer medidas que limitem o número de padrões gerados. Dessa forma, tem início uma nova etapa cujo objetivo é a avaliação dos padrões, eliminando aqueles que são menos interessantes ou que não representam conhecimento. Uma forma de avaliar o interesse de um padrão encontrado é o uso de medidas de interesse, sendo que um padrão é interessante caso satisfaça um limite mínimo pré-estabelecido de determinadas medidas de interesse.

Finalmente, é feita a apresentação do conhecimento obtido através dos padrões. A apresentação deve ser feita de uma forma que facilite a análise dos resultados, pois normalmente seu uso se restringe ao nível estratégico das organizações. As formas de apresentação mais utilizadas são relatórios e gráficos.

7.4. Data Warehouse

Um *data warehouse* (DW) consiste em um repositório de dados cuja estrutura está voltada para o suporte a decisões. É uma base de dados integrada, não volátil, variável com o

tempo e mantida separadamente da base de dados operacional correspondente (CHAUDHURI; DAYAL, 1997).

Data warehouses diferem das bases de dados transacionais em seu propósito, estrutura, funcionamento e desempenho (ELMASRI; NAVATHE, 2000). Enquanto as bases de dados transacionais armazenam registros individuais detalhados, em um *data warehouse* são armazenados dados históricos, sumarizados e consolidados (CHAUDHURI; DAYAL, 1997).

A modelagem de um *data warehouse* é orientada a assunto, na qual são contempladas apenas as informações importantes para o processo de tomada de decisões. Além disso, os *data warehouses* são repositórios que integram dados de diversas fontes heterogêneas, o que exige a aplicação de técnicas de limpeza e integração dos dados para garantir a consistência dos mesmos.

Além da tomada de decisões, *data warehouses* podem ser usados como uma etapa de pré-processamento no processo de descoberta de conhecimento (KDD), o qual se beneficia das etapas de limpeza e integração dos dados.

7.4.1. Modelagem Multidimensional

O modelo multidimensional é usado para modelar *data warehouses* devido ao seu melhor desempenho na execução de consultas e na análise dos dados em relação ao modelo relacional (CHAUDHURI; DAYAL, 1997).

A estrutura multidimensional facilita a manipulação dos dados, já que os organiza em torno de um tema central ou assunto. Esse assunto é representado pela tabela fato e consiste na medida que se deseja analisar. A tabela fato possui várias chaves que determinam suas dimensões, representadas por tabelas menores, as tabelas dimensões. As dimensões representam as perspectivas ou entidades que se deseja armazenar e podem ser consideradas características dos fatos. Segundo Elmasri e Navathe (ELMASRI; NAVATHE, 2000), a modelagem multidimensional pode ser feita usando o esquema estrela, constelação de fatos ou flocos de neve.

O esquema estrela trata apenas um assunto de interesse, o qual é representado por uma tabela fato. As características que permitem analisar esse assunto são representadas por tabelas dimensão. Graficamente, a tabela fato é representada no centro e suas dimensões ao redor da mesma.

Como exemplo, considere o caso de uma empresa que deseja analisar suas vendas. Para isso, ela precisa relacionar determinada venda com o cliente que efetuou a compra, o produto

adquirido e a data em que a venda ocorreu. A Figura 7.2 mostra o esquema estrela desse exemplo.

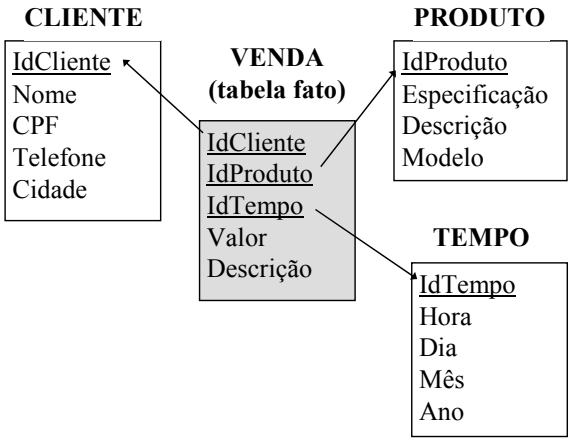

Figura 7.2 - Exemplo de esquema estrela

O esquema constelação de fatos consiste em uma extensão da modelagem do esquema estrela, sendo capaz de representar a existência de mais de um assunto a ser analisado. Nesse caso, existem várias tabelas fato, uma para cada assunto, as quais são indiretamente relacionadas através do compartilhamento de tabelas dimensões.

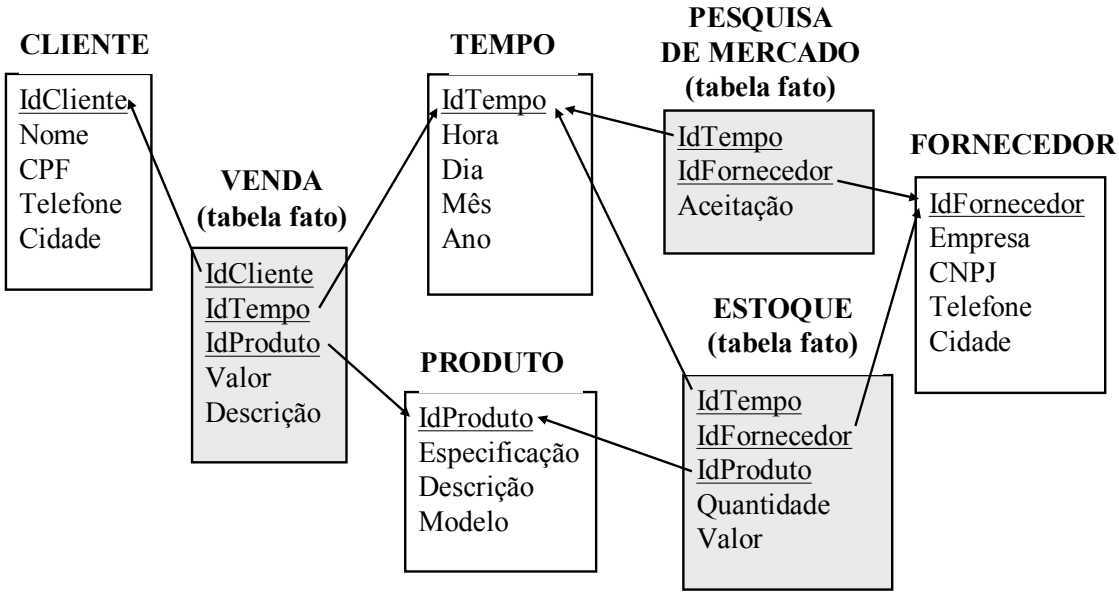

Figura 7.3 - Exemplo de constelação de fatos

Considere o exemplo de uma empresa que deseja analisar suas vendas e sua reposição de estoque. Para isso, ela precisa relacionar as vendas aos clientes que efetuaram a compra, aos produtos que foram adquiridos e às datas em que ocorreram essas vendas. Além disso,

para analisar o estoque, é necessário considerar os produtos que foram comprados, o fornecedor dos mesmos e a data em que ocorreu essa compra. A Figura 7.3 mostra a constelação de fatos para esse exemplo.

O esquema flocos de neve consiste em uma variação do esquema estrela, no qual algumas dimensões são normalizadas. Nesse caso, evita-se a redundância, porém perde-se em desempenho, pois é necessário um maior número de junções para responder às requisições.

A Figura 7.4 mostra a modelagem flocos de neve para o esquema estrela da Figura 7.2. Nesse caso, foram criadas 3 novas tabelas: cidade, modelo e mês.

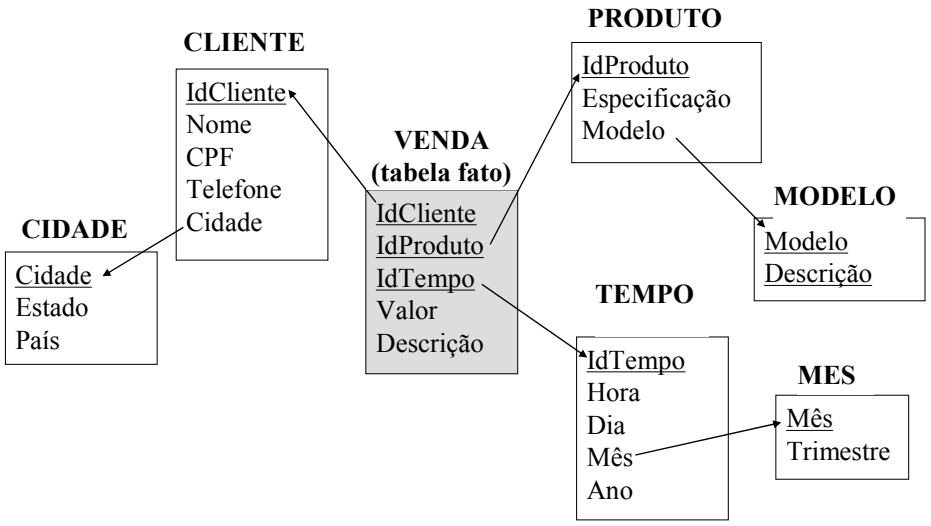

Figura 7.4 - Exemplo de flocos de neve

7.5. Tarefas de Mineração

Uma tarefa de mineração de dados define a técnica usada para buscar padrões e o tipo de padrão desejado. Como uma grande quantidade de padrões irrelevantes pode ser gerada, as tarefas de mineração podem considerar apenas uma fração do banco de dados como seu espaço de busca ou estabelecer medidas de interesse para restringir certos padrões encontrados. Além disso, pode ser levado em conta o conhecimento prévio que se tem sobre o domínio do problema para orientar a busca, o que favorece a descoberta de padrões interessantes.

As tarefas de mineração podem ser classificadas em várias categorias, de acordo com o tipo de padrão encontrado. As principais tarefas são:

- Associação
- Classificação

- Agrupamento
- Padrões seqüenciais

7.5.1. Tarefa de Associação

Uma tarefa de associação busca por padrões que demonstrem o relacionamento entre conjuntos de itens. Os padrões encontrados são representados na forma de uma implicação $X \rightarrow Y$, onde X e Y representam conjuntos de itens e o padrão encontrado sugere que a presença de X está relacionada à presença de Y. Esse relacionamento pode ser direto, quando a presença de X aumenta a possibilidade da presença de Y, ou inverso, quando a presença de X diminui a possibilidade da presença de Y.

Os dados da Tabela 7.1 referem-se a uma cesta de compras e serão usados para exemplificar a regra de associação. Considerando a regra *manteiga → pão*, nota-se que dentre as 5 transações existentes, pão e manteiga ocorrem em 2 transações. Além disso, dentre as 4 transações em que manteiga ocorre, pão ocorre em 2 dessas transações. Dessa forma, pode-se dizer que a regra possui freqüência de 40% e credibilidade de 50%, o que demonstra que essa regra pode revelar um padrão de comportamento dos clientes: "Clientes que compram manteiga tendem a comprar pão". No capítulo 2 desta monografia são apresentados mais detalhes sobre regras de associação.

Tabela 7.1 - Exemplo de cesta de compras

IdTransação	Item
100	Pão, leite, manteiga.
200	Pão, requeijão, leite.
300	Manteiga, farinha, leite.
400	Manteiga, pão, refrigerante.
500	Bolacha, leite, manteiga.

7.5.2. Tarefa de Classificação

A tarefa de classificação agrupa dados em hierarquia de classes de acordo com os valores de seus atributos. Para isso é utilizado um atributo alvo, cujo valor determina a classe à qual pertence. A classificação é realizada a partir de dois passos principais: construção de um modelo de classificação e posterior aplicação desse modelo para classificar os dados desejados.

O modelo pode ser representado através de regras de classificação, árvores de decisão ou fórmulas matemáticas e é construído a partir da aplicação do algoritmo de classificação sobre um conjunto treinamento, cuja classificação é conhecida.

Posteriormente as regras obtidas são usadas para classificar a base de dados. Inicialmente é realizado um teste para verificar a exatidão dessas regras usando para isso uma amostra de dados selecionada aleatoriamente. Dessa forma, as regras serão usadas para classificar dados cujo atributo alvo é desconhecido, somente se o resultado do teste for satisfatório. Para uma melhor compreensão da tarefa de classificação pode-se consultar (FREITAS, 2000), que compara as tarefas de classificação e associação.

A seguir é mostrado um exemplo sobre a classificação de clientes de uma operadora de celular. A Tabela 7.2 representa o conjunto treinamento, no qual o atributo Categoria é o atributo alvo, cujo valor é conhecido. As regras obtidas a partir do conjunto treinamento são mostradas na Figura 7.5.

Tabela 7.2 - Dados relativos à classificação de clientes de uma operadora de celular

Aparelho	Plano	Categoria
com_câmera	300	ouro
com_câmera	150	ouro
com_câmera	50	ouro
Colorido	300	ouro
Colorido	150	prata
Colorido	50	bronze
pretobranco	300	bronze
pretobranco	150	bronze

SE (Aparelho = “com_câmera”) ENTÃO Categoria = “ouro”
SE (Aparelho = “colorido”) E (Plano = 300) ENTÃO Categoria = “ouro”
SE (Aparelho = “colorido”) E (Plano = 150) ENTÃO Categoria = “prata”
SE (Aparelho = “colorido”) E (Plano = 50) ENTÃO Categoria = “bronze”
SE (Aparelho = “pretobranco”) ENTÃO Categoria = “bronze”

Figura 7.5 Regras de classificação geradas pelo conjunto treinamento

Alguns algoritmos de classificação retornam, como resultado, uma árvore de decisão. Uma árvore de decisão é uma estrutura eficiente, intuitiva e pode ser facilmente convertida em regras de classificação. Em sua estrutura, os nós consistem em testes realizados sobre os dados, os ramos representam o resultado desses testes e as folhas constituem o atributo alvo.

Durante o processo de indução da árvore, alguns nós criados podem refletir ruídos nos dados, o que requer um mecanismo de poda. Além disso, pode ser realizada uma análise da

influência dos atributos nas regras para determinar quais níveis da árvore serão ocupados pelos mesmos. A Figura 7.6 mostra a árvore gerada a partir do conjunto treinamento da Tabela 7.2.

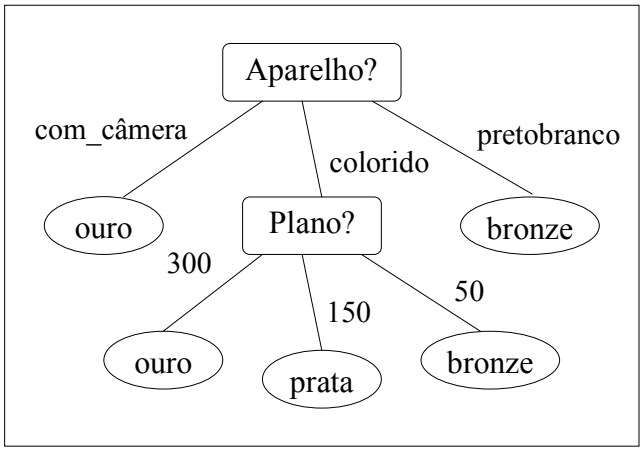

Figura 7.6 - Árvore de decisão gerada
a partir do conjunto treinamento da Tabela 7.2

7.5.3. Tarefa de Agrupamento

A tarefa de agrupamento, também conhecida como clusterização, consiste em agrupar itens semelhantes em clusters. A semelhança encontrada entre itens de um mesmo cluster deve ser sempre maior que a encontrada entre itens de clusters distintos. A diferença entre o agrupamento e classificação consiste no fato de que no agrupamento não é conhecido um rótulo que define o comportamento dos dados do cluster, enquanto na classificação existe um atributo alvo que define a classe.

Na Figura 7.7 é mostrado um conjunto de dados no qual será aplicado um algoritmo de agrupamento. Nesse exemplo, considera-se que o valor dos atributos analisados são representados como pontos e um cluster é representado por uma região do gráfico com grande concentração de pontos. Como clusters agrupam itens semelhantes, pode-se considerar que pontos próximos são semelhantes, ou seja, a semelhança entre itens é medida pela distância entre seus pontos e quanto menor a distância, maior a semelhança.

Segundo Han e Kamber (HAN; KAMBER, 2001), os métodos utilizados nos algoritmos de agrupamento podem ser classificados em métodos de particionamento, métodos hierárquicos, métodos baseados em modelos e métodos baseados em grid. A Figura 7.8 mostra o resultado da aplicação de um algoritmo de agrupamento por particionamento nos dados da Figura 7.7. Foram obtidos 3 clusters, o que identifica 3 grupos de itens com comportamento semelhante.

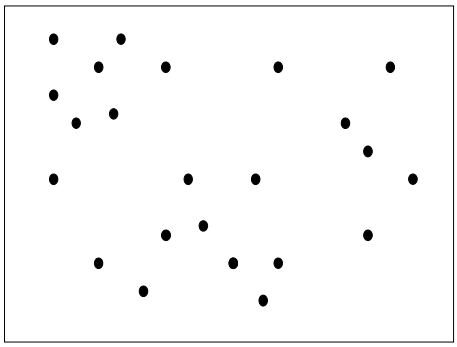

Figura 7.7 - Conjunto de dados para aplicação de algoritmo de agrupamento

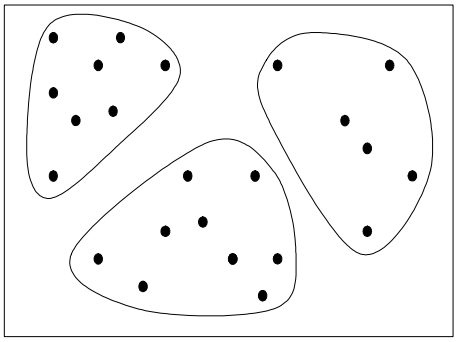

Figura 7.8 - Resultado da aplicação de um algoritmo de agrupamento

7.5.4. Padrões Seqüenciais

Essa tarefa consiste na busca por seqüências de itens e foi introduzida em (AGRAWAL; SRIKANT, 1995). Deseja-se descobrir uma seqüência de itens que ocorre sempre na mesma ordem em determinada base de dados e que satisfaça uma freqüência mínima pré-estabelecida.

Tabela 7.3 - Compras realizadas em uma loja de equipamentos de informática

IdCliente	Data_Hora	Cesta de compras
Cl1	05/01/04 20:00	PC, jogo, CD-R
Cl1	10/04/04 15:00	impressora, rack, disquete
Cl1	20/07/04 21:00	scanner, placa de vídeo
Cl2	06/01/04 17:00	PC
Cl2	10/07/04 12:00	scanner, impressora
Cl3	05/01/04 10:00	jogo, CD-R
Cl3	20/05/04 15:00	placa de vídeo, disquete
Cl4	26/03/04 11:00	PC, rack
Cl5	10/04/04 09:00	PC, rack, disquete
Cl5	17/07/04 11:00	CD-R, impressora, jogo
Cl5	10/09/04 18:00	placa de vídeo, scanner

A tabela 7.3 mostra as compras realizadas em uma loja de equipamentos de informática. A partir desses dados, pode-se descobrir se existe uma seqüência de compras que se repete

para vários clientes. Dessa forma, as compras são agrupadas para cada cliente, em ordem crescente da data da compra, conforme mostrado na tabela 7.4.

Tabela 7.4 - Seqüência de compras de cada cliente

IdCliente	Seqüência de compras
Cl1	< (PC, jogo, CD-R) (impressora, rack, disquete) (scanner, placa de vídeo)>
Cl2	< (PC) (scanner, impressora)>
Cl3	< (jogo, CD-R) (placa de vídeo, disquete) >
Cl4	< (PC, rack) >
Cl5	< (PC, rack, disquete) (CD-R, impressora, jogo) (placa de vídeo, scanner)>

O próximo passo é a determinação de quais dessas seqüências de itens comprados satisfazem a freqüência mínima pré-estabelecida pelo usuário. A Tabela 7.5 exibe as seqüências freqüentes, supondo uma freqüência mínima de 50%.

Tabela 7.5 - Padrões seqüenciais freqüentes

Seqüências com freqüência maior ou igual a 50%
< (PC) (impressora) >
< (PC) (scanner) >
< (jogo, CD-R) (placa de vídeo) >

Nesse caso, pode-se concluir que é comum um cliente comprar um PC e posteriormente comprar um scanner ou uma impressora, ou então que um cliente que compra jogo e CD-R, provavelmente comprará uma placa de vídeo posteriormente. No entanto, isso não impede que esse cliente possa adquirir outros itens entre a compra do PC e do scanner, por exemplo. Isso pode ser visualizado no registro do cliente *Cl1*, que compra um PC na primeira compra e um scanner na terceira compra.

7.6. Considerações Finais

Neste apêndice foi apresentada uma introdução ao processo de mineração de dados. Além disso, foram apresentados o processo de descoberta de conhecimento e suas etapas, dentre as quais a mineração de dados é destacada como uma das mais importantes. Foram também introduzidos alguns conceitos sobre data warehouses e modelagem multi-dimensional. Finalmente, foram apresentadas algumas tarefas de mineração: associação, agrupamento, classificação e padrões seqüenciais.

8. Apêndice B - Exemplo de Execução do Algoritmo GFP-Growth

8.1.Considerações Iniciais

Neste apêndice será detalhado o funcionamento do algoritmo GFP-Growth, através da execução passo a passo de um exemplo. Serão apresentadas as fases do algoritmo juntamente com as estruturas de dados geradas durante o processo.

8.2. Apresentação do Algoritmo GFP-Growth

Para facilitar a apresentação do algoritmo GFP-Growth, o mesmo será dividido em 4 principais métodos: contagem dos itemsets, construção da *GFP-Tree*, mineração da *GFP-Tree* e geração das regras. Esses métodos e seus submétodos são chamados na seguinte ordem:

countItemOccurrences
constructGFP-Tree
    processRow
        insert
gfp_Growth
    combine
    buildConditionalGFP-Tree
        processPattern
            insert
generateRules

O método *countItemOccurrences* é responsável pela contagem das ocorrências dos itens em agrupamentos e blocos, permitindo o cálculo do peso e *sup_a*. Já o método *constructGFP-Tree* é responsável pela criação da *GFP-Tree* inicial, havendo nesse método dois importantes submétodos: *processRow* e *insert*.

O submétodo *processRow* é responsável pela inserção de uma tupla do banco na *GFP-Tree*, verificando quais itens participarão efetivamente da árvore e classificando-os em ordem decrescente de *sup_a*. Já o submétodo *insert* navega na árvore em busca do local de inserção do item e atualiza as medidas do nó e da tabela header.

O método GFP-Growth é responsável pela mineração da *GFP-Tree* inicial. Basicamente, seu funcionamento consiste em gerar combinações entre os itens de um ramo, caso a árvore tenha apenas esse ramo, e em caso contrário gerar *GFP-Tree*'s condicionais e minerá-las recursivamente até que as mesmas tenham um único ramo para gerar combinações de itens. Para realizar tal tarefa, esse método utiliza dois submétodos: o submétodo *combine*,

que realiza as combinações entre os itens de um ramo e armazena essa combinação, e o submétodo *buildConditionalGFP-Tree,* que gera as *GFP-Tree*'s condicionais.

O submétodo *buildConditionalGFP-Tree* verifica para cada nó da *GFP-Tree* seus nós ascendentes, os quais formam o prefixo do nó, e insere os itens desse prefixo na *GFP-Tree* condicional do item em questão. Para isso, é utilizado o submétodo *processPattern,* que é semelhante ao submétodo *processRow,* sendo responsável pelo tratamento do conjunto de itens a ser inserido em uma *GFP-Tree*. No entanto, nesse caso as informações a serem tratadas sobre os itens são diferentes, já que envolvem posições do mesmo em tabelas *hash* e ponteiros dos nós. O submétodo *insert* é semelhante ao citado no método *constructGFP-Tree* .

Finalmente, o método *generateRules* é responsável pela geração das regras, que é feita da seguinte forma: para cada combinação de itens gerada no método GFP-Growth, seus itens são combinados entre si para gerar diferentes regras. Para cada regra gerada, calcula-se sua *conf_a*, e se a mesma satisfizer a mínima *conf_a* pré-estabelecida, então essa regra é retornada ao usuário.

1.1. Exemplo Considerado

A base de dados mostrada na figura 8.1 contém informações sobre pacientes e será usada para exemplificar a execução do GFP-Growth. Essa figura disponibiliza dados pessoais dos pacientes, informações se os pacientes são fumantes ou alérgicos, além de seus prontuários, que indicam seus sintomas e respectivos diagnósticos.

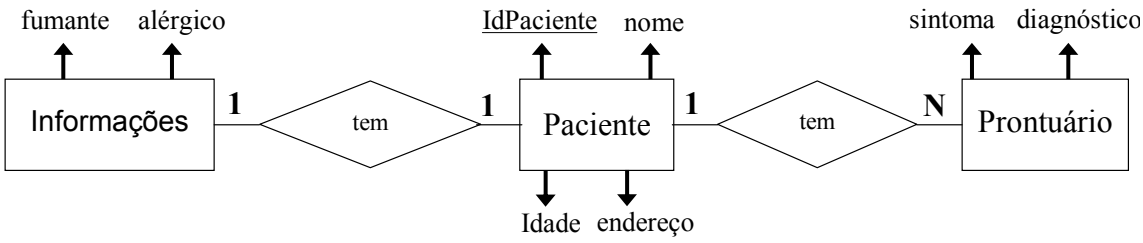

Figura 8.1 Base contendo informações sobre pacientes

Nesse caso, deseja-se analisar se existe alguma relação entre os sintomas e diagnósticos dos pacientes e o fato de os mesmos serem fumantes ou alérgicos. Para isso, será necessário minerar em conjunto as tabelas *Informações* e *Prontuário*, as quais são relacionadas a partir do identificador do paciente, que ambas as tabelas compartilham.

8.4. Preparação dos Dados

As tabelas 8.1 e 8.2 concentram os dados referentes às tabelas Informações e Prontuário, da figura 8.1. Para obter o conjunto de entrada do algoritmo GFP-Growth, é realizada a operação de junção externa entre as tabelas 8.1 e 8.2, de forma que o identificador do paciente não seja nulo. A tabela resultante é mostrada na tabela 8.3.

Tabela 8.1 Tabela Informações

IdPaciente	Fumante	Alérgico
Pac1	fuma=sim	alérgico=não
Pac2	fuma=sim	alérgico=sim
Pac3	fuma=não	alérgico=sim
Pac4	fuma=sim	alérgico=não
Pac5	fuma=não	alérgico=não
Pac7	fuma=não	alérgico=sim
Pac8	fuma=não	alérgico=não

Tabela 8.2 Tabela Prontuário

IdPaciente	Sintoma	Diagnóstico
Pac1	febre	resfriado
Pac3	inflamação	bactéria
Pac3	coriza	vírus
Pac3	dor	fratura
Pac4	hipertensão	distúrbio hormonal
Pac6	febre	resfriado
Pac6	hipertensão	distúrbio hormonal
Pac7	inflamação	bactéria
Pac7	coriza	vírus

O algoritmo GFP-Growth será aplicado à tabela 8.3 considerando os seguintes valores mínimos para as medidas de interesse: mínimo *sup_a* = 50%, mínima *conf_a* = 50% e mínimo peso = 50%.

Tabela 8.3 Conjunto de entrada para o GFP-Growth

IdPaciente	Fumante	Alérgico	Sintoma	Diagnóstico
Pac1	fuma=sim	alérgico=não	febre	resfriado
Pac2	fuma=sim	alérgico=sim	null	null
Pac3	fuma=não	alérgico=sim	inflamação	bactéria
Pac3	fuma=não	alérgico=sim	coriza	vírus
Pac3	fuma=não	alérgico=sim	dor	fratura
Pac4	fuma=sim	alérgico=não	hipertensão	distúrbio hormonal
Pac5	fuma=não	alérgico=não	null	null
Pac6	null	null	febre	resfriado
Pac6	null	null	hipertensão	distúrbio hormonal
Pac7	fuma=não	alérgico=sim	inflamação	bactéria
Pac7	fuma=não	alérgico=sim	coriza	vírus
Pac8	fuma=não	alérgico=não	null	null

8.5. Execução do Algoritmo

8.5.1. Método *countItemOccurrences*

Inicialmente é chamado o método *countItemOccurrences*, que é responsável pela contagem dos itens e tratamento dos valores obtidos. Alguns valores manipulados por essa função incluem número de blocos em que os itens ocorrem (blocos), o número de agrupamentos em que os itens ocorrem (agrup) e o peso dos itens (peso). O resultado obtido a partir da função *countItemOccurrences* para a tabela 8.3 pode ser observado na tabela 8.4. Nessa tabela também é mostrado o suporte dos itens.

8.5.2. Método *constructGFP-Tree*

Posteriormente, a função *constructGFP-Tree* é chamada. Nessa função, o peso e o suporte de cada item são verificados, com o auxílio das tabelas *hash* que contêm os agrupamentos e pesos dos itens, para verificar o número de itens freqüentes. A partir daí, uma *GFP-Tree* é criada para esses itens freqüentes, e para cada item é criada uma entrada na tabela header.

Para cada tupla do banco de dados, se a mesma formar agrupamento, seus itens serão utilizados na *GFP-Tree*, a partir da função ***processRow***. Nessa função, os itens freqüentes da linha são capturados, juntamente com seu agrupamento e classificados em ordem decrescente de *sup_a* para compor o conjunto L, que é o seguinte: *L = {alérgico=não, alérgico=sim, bactéria, coriza, fuma=não, fuma=sim, inflamação, vírus}*. Esse conjunto determina a ordem de inserção e é enviados para a função ***insert***, que irá inserir na *GFP-Tree* criada anteriormente.

Tabela 8.4 Valores obtidos a partir da contagem dos itens

Blocos	Agrupamentos	Peso	Suporte
blocos[Resfriado] = 2	agrup[Resfriado] = 1	peso[Resfriado] = 0.5	sup[Resfriado] = 0.25
blocos[Coriza] = 2	agrup[Coriza] = 2	peso[Coriza] = 1.0	sup[Coriza] = 0.5
blocos[Dor] = 1	agrup[Dor] = 1	peso[Dor] = 1.0	sup[Dor] = 0.25
blocos[Inflamação] = 2	agrup[Inflamação] = 2	peso[Inflamação] = 1.0	sup[Inflamação] = 0.5
blocos[fuma=não] = 4	agrup[fuma=não] = 2	peso[fuma=não] = 0.5	sup[fuma=não] = 0.5
blocos[Dist_hormonal] = 2	agrup[Dist_hormonal] = 1	peso[Dist_hormonal] = 0.5	sup[Dist_hormonal] = 0.25
blocos[alérgico=não] = 4	agrup[alérgico=não] = 2	peso[alérgico=não] = 0.5	sup[alérgico=não] = 0.5
blocos[Bactéria] = 2	agrup[Bactéria] = 2	peso[Bactéria] = 1.0	sup[Bactéria] = 0.5
blocos[Hipertensão] = 2	agrup[Hipertensão] = 1	peso[Hipertensão] = 0.5	sup[Hipertensão] = 0.25
blocos[Vírus] = 2	agrup[Vírus] = 2	peso[Vírus] = 1.0	sup[Vírus] = 0.5
blocos[Fratura] = 1	agrup[Fratura] = 1	peso[Fratura] = 1.0	sup[Fratura] = 0.25
blocos[Febre] = 2	agrup[Febre] = 1	peso[Febre] = 0.5	sup[Febre] = 0.25
blocos[fuma=sim] = 3	agrup[fuma=sim] = 2	peso[fuma=sim] = 0.666	sup[fuma=sim] = 0.5
blocos[alérgico=sim] = 3	agrup[alérgico=sim] = 2	peso[alérgico=sim] = 0.666	sup[alérgico=sim] = 0.5

Em ***insert*** para cada item considerado, é verificado se seu agrupamento já ocorreu na árvore antes e se o item já ocorreu nesse mesmo agrupamento. A partir dessas verificações, os contadores e os agrupamentos da header são atualizados.

O próximo passo é a inserção dos itens na *GFP-Tree*, de acordo com a ordem em que os mesmos aparecem no conjunto L. Para cada nó a partir da raiz é verificado se existe algum nó *filho* contendo o item em questão. Se o item não foi encontrado, será necessário criar um novo nó. Se o nó corrente já possui filhos, então agora a *GFP-Tree* terá vários ramos.

É criado um novo nó com as características do item, e o ponteiro *próximo* da header recebe esse nó criado. O nó criado será o nó *filho* do nó corrente, e a inserção continua a partir desse nó criado. Se o item foi encontrado na árvore, seu contador é incrementado e seu agrupamento é inserido no nó. A inserção continua a partir desse nó encontrado.

A tabela header e a *GFP-Tree* geradas para os dados da tabela 8.3 são mostradas na figura 8.2.

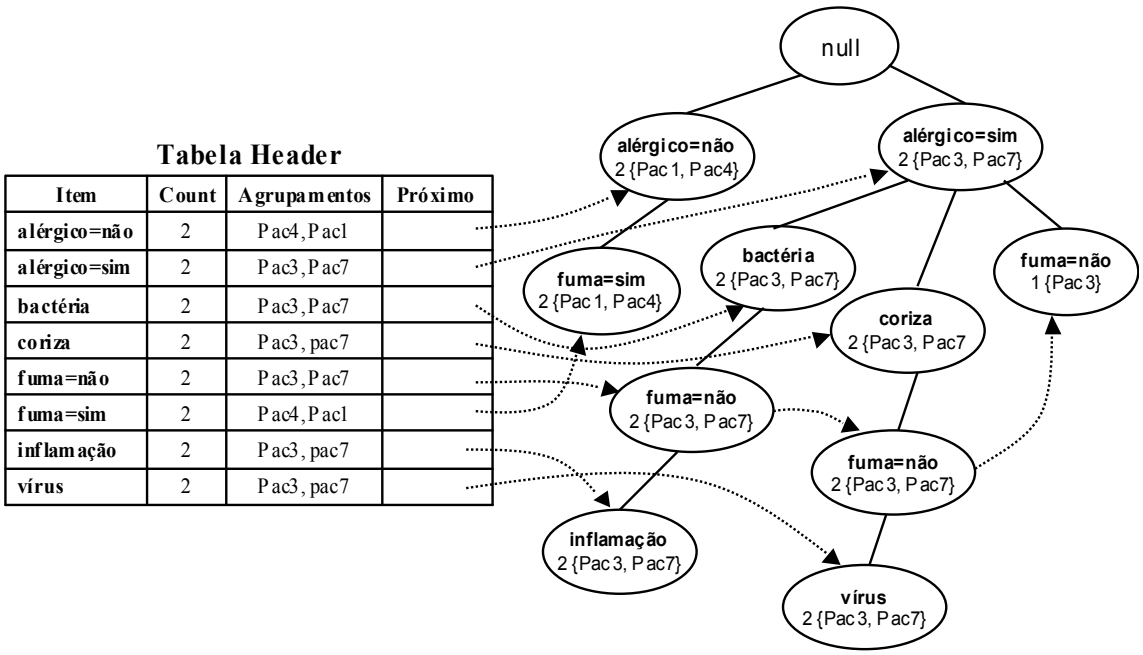

Figura 8.2 *GFP-Tree* gerada e sua tabela header

8.5.3. Método *gfp_Growth*

O próximo passo é a mineração da árvore gerada, a partir da função ***gfp_Growth***. A mineração é feita da seguinte forma: se a árvore possui apenas um ramo, os itens desse ramo são combinados para gerar os padrões freqüentes; caso tenha mais de um ramo, são criadas

GFP-Tree's condicionais, as quais são mineradas posteriormente, através da aplicação recursiva da função ***gfp_Growth***.

Se a árvore tem apenas um ramo, a combinação dos itens desse ramo é feita pela função ***combine***. Para gerar um padrão, cada item do ramo é adicionado a um itemset que foi passado como parâmetro à função em questão. Esse itemset representa uma combinação de itens e é armazenado para posteriormente gerar uma regra de associação. Recursivamente, os itens que ainda não foram utilizados são adicionados a esse itemset e o itemset resultante é armazenado.

Se a árvore tem mais de um ramo, ocorre o mesmo processo de adição do item ao itemset e seu armazenamento, porém nesse caso também é criada uma *GFP-Tree* condicional para o item considerado. O processo tem início nos nós folhas, e o itemset inicial do parâmetro é nulo. Caso uma *GFP-Tree* condicional gerada tenha mais de um ramo, recursivamente é criada uma *GFP-Tree* condicional para um 2-itemset, e assim sucessivamente, até que tenha apenas um ramo, para que a função ***combine*** possa ser aplicada. A criação da *GFP-Tree* condicional é discutida na seção a seguir.

8.5.4. Submétodo *buildConditionalGFP-Tree*

Para construir a *GFP-Tree* condicional é usada a função ***buildConditionalGFP-Tree***. Inicialmente, é obtida a entrada na header do item considerado e com isso pode-se obter o nó que representa o item na árvore (ponteiro *próximo* da header). É feita uma navegação na árvore, analisando os nós ascendentes para cada ocorrência do item, com o objetivo de capturar seu *sup_a*, o qual é armazenado.

Com esses valores, é possível verificar quais itens são freqüentes. Esses itens são classificados em ordem decrescente de suporte e uma *GFP-Tree* é criada para armazená-los. A árvore é percorrida novamente, analisando os prefixos do item analisado, para guardar cada item e sua entrada na header em um vetor. Esse vetor contém os itens que ocorrem junto em um ramo da árvore, e constitui um padrão encontrado, o qual é processado pela função ***processPattern***.

Em ***processPattern***, para cada item do ramo considerado, verifica-se quais itens são freqüentes, e é feita a classificação dos mesmos em ordem decrescente de suporte. Nessa ordem, os itens serão inseridos na *GFP-Tree*, pela função ***insert***.

Em ***insert***, os itens, juntamente com o contador do padrão e os grupos nos quais os mesmos ocorrem, são passados como parâmetro para a função. Inicialmente os itens são ordenados de acordo com sua entrada na header. Para inserir na árvore, para cada item, é

verificado se existe algum nó *filho* contendo o item em questão. Se o item não foi encontrado, será necessário criar um novo nó. Se o nó corrente já possui filhos, então agora a *GFP-Tree* terá vários ramos.

É criado um novo nó, e cada agrupamento desse item é inserido na tabela header e no nó, caso ainda não exista nos mesmos. Esse novo nó criado é atribuído ao ponteiro *próximo* da header e ao ponteiro *filho* do nó corrente, e a inserção continua a partir desse novo nó. Porém, se o item foi encontrado na árvore, seu contador é incrementado e cada agrupamento é inserido na header e no nó, caso ainda não exista. A inserção continua a partir desse nó encontrado.

A análise dos itens da tabela 8.3 ocorre da seguinte forma:

Em ***gfp_Growth***, como a árvore tem mais de um ramo, será criada a *GFP-Tree* condicional do item *vírus*, e sua combinação com o prefixo nulo é realizada e guardada, conforme se pode observar neste trecho gerado pelo código do GFP-Growth:

```
ITEM [8/8] -> vírus, agrup-> 2, agrupamentos-> Pac7, Pac3
Armazenou-> {vírus }/[0.5/2] (1)
```

Algumas informações sobre a *GFP-Tree* condicional a ser construída podem ser observadas no trecho abaixo, o qual foi gerado pelo código do GFP-Growth. A figura 8.3 mostra a *GFP-Tree* condicional gerada e sua tabela header a partir do sufixo *vírus*.

```
Sufixo-> vírus  Padrão-> <alérgico=sim, 1> <coriza, 3> <fuma=não, 4>
```

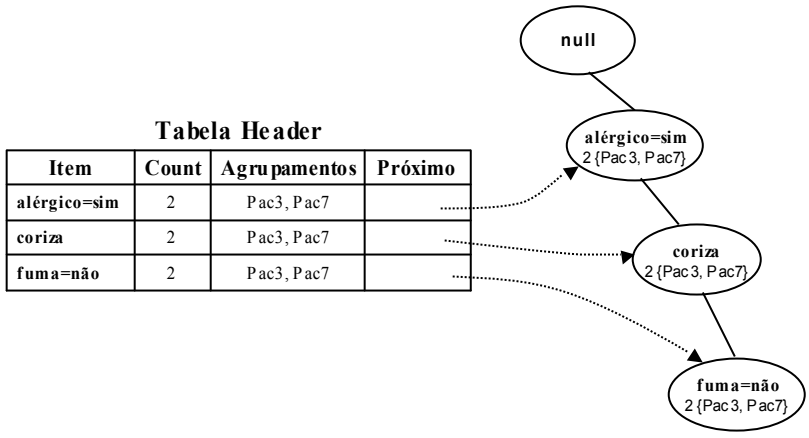

Figura 8.3 *GFP-Tree* condicional do sufixo *vírus*

A função ***gfp_Growth*** é aplicada recursivamente sobre a *GFP-Tree* condicional da figura 8.3. Como a árvore tem apenas um ramo, os itens da mesma são combinados com o

sufixo vírus. O resultado pode ser observado neste trecho gerado pelo código do GFP-Growth:

```
MINERAÇÃO DA GFP-TREE CONDICIONAL DO ITEMSET -> {vírus}
        Combinar-> {fuma=não, coriza, alérgico=sim, } com {vírus }/[0.5/2] (1)
          Gerou-> {vírus fuma=não }/[0.5/2] (2) agrup-> 2 sup-> 0.5
                Armazenou-> {vírus fuma=não }/[0.5/2] (2)
        Combinar-> coriza alérgico=sim  com-> {vírus fuma=não }/[0.5/2] (2)
                Armazenou-> {vírus fuma=não coriza }/[0.5/2] (3)
        Combinar-> alérgico=sim  com-> {vírus fuma=não coriza }/[0.5/2] (3)
                Armazenou-> {vírus fuma=não coriza alérgico=sim }/[0.5/2] (4)
                Armazenou-> {vírus fuma=não alérgico=sim }/[0.5/2] (3)
          Gerou-> {vírus coriza }/[0.5/2] (2) agrup-> 2 sup-> 0.5
                Armazenou-> {vírus coriza }/[0.5/2] (2)
        Combinar-> alérgico=sim  com-> {vírus coriza }/[0.5/2] (2)
                Armazenou-> {vírus coriza alérgico=sim }/[0.5/2] (3)
          Gerou-> {vírus alérgico=sim }/[0.5/2] (2) agrup-> 2 sup-> 0.5
                Armazenou-> {vírus alérgico=sim }/[0.5/2] (2)
```

A função ***gfp_Growth*** finaliza a recursão, e retorna à análise da *GFP-Tree* inicial da figura 8.2. Como a *GFP-Tree* tem mais de um ramo, será necessário construir mais *GFP-Tree*'s condicionais. O próximo item da tabela header a ser analisado é o nó folha *inflamação*. Esse item inicialmente é combinado com o sufixo *nulo* e o itemset gerado é armazenado, conforme se pode observar neste trecho gerado pelo código do GFP-Growth:

```
ITEM [7/8] -> inflamação, agrup-> 2, agrupamentos-> Pac7, Pac3
Armazenou-> {inflamação }/[0.5/2] (1)
```

Algumas informações sobre a *GFP-Tree* condicional a ser construída podem ser observadas no trecho abaixo gerado pelo código do GFP-Growth. A figura 8.4 mostra a *GFP-Tree* condicional gerada e sua tabela header a partir do sufixo *inflamação*.

```
Sufixo-> inflamação Padrão-> <alérgico=sim, 1> <bactéria, 2> <fuma=não, 4>
```

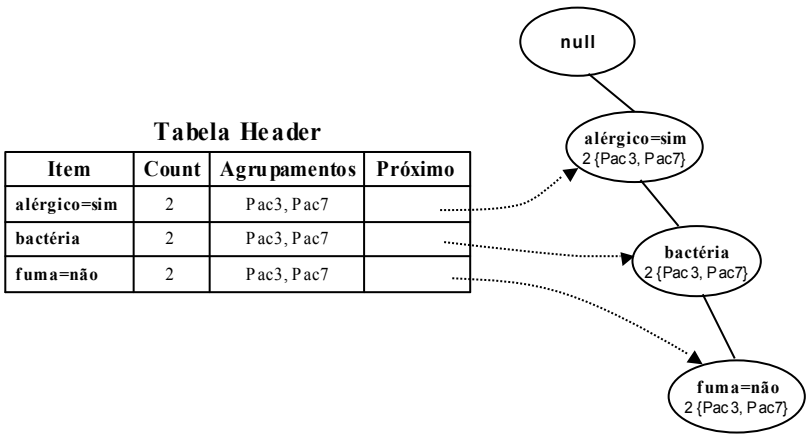

Figura 8.4 *GFP-Tree* condicional do sufixo *inflamação*

A função ***gfp_Growth*** é aplicada recursivamente sobre a *GFP-Tree* condicional da figura 8.4, e como a árvore tem apenas um ramo, é feita a combinação dos itens com o sufixo *inflamação*. O resultado pode ser observado neste trecho gerado pelo código do GFP-Growth:

```
MINERAÇÃO DA GFP-TREE CONDICIONAL DO ITEMSET-> {inflamação}
        Combinar-> {fuma=não, bactéria, alérgico=sim, } com {inflamação }/[0.5/2] (1)
          Gerou-> {inflamação fuma=não }/[0.5/2] (2) agrup-> 2 sup-> 0.5
                Armazenou-> {inflamação fuma=não }/[0.5/2] (2)
        Combinar-> bactéria alérgico=sim  com-> {inflamação fuma=não }/[0.5/2] (2)
                Armazenou-> {inflamação fuma=não bactéria }/[0.5/2] (3)
        Combinar-> alérgico=sim  com-> {inflamação fuma=não bactéria }/[0.5/2] (3)
                Armazenou-> {inflamação fuma=não bactéria alérgico=sim }/[0.5/2] (4)
                Armazenou-> {inflamação fuma=não alérgico=sim }/[0.5/2] (3)
          Gerou-> {inflamação bactéria }/[0.5/2] (2) agrup-> 2 sup-> 0.5
                Armazenou-> {inflamação bactéria }/[0.5/2] (2)
        Combinar-> alérgico=sim  com-> {inflamação bactéria }/[0.5/2] (2)
                Armazenou-> {inflamação bactéria alérgico=sim }/[0.5/2] (3)
          Gerou-> {inflamação alérgico=sim }/[0.5/2] (2) agrup-> 2 sup-> 0.5
                        Armazenou-> {inflamação alérgico=sim }/[0.5/2] (2)
```

A função ***gfp_Growth*** finaliza a recursão, e retorna à análise da *GFP-Tree* inicial (figura 8.2). Como a *GFP-Tree* tem mais de um ramo, será necessário construir mais *GFP-Tree*'s condicionais. O próximo item da header a ser analisado é o nó *fuma=sim*. Esse item é combinado com o sufixo nulo e o itemset gerado é armazenado, conforme se pode observar neste trecho gerado pelo código do GFP-Growth:

```
ITEM [6/8] -> fuma=sim, agrup-> 2, agrupamentos-> Pac4, Pac1
Armazenou-> {fuma=sim }/[0.5/2] (1)
```

Algumas informações sobre a *GFP-Tree* condicional a ser construída podem ser observadas no trecho abaixo gerado pelo código do GFP-Growth. A figura 8.5 mostra a *GFP-Tree* condicional gerada e sua tabela header a partir do sufixo *fuma=sim*.

```
Sufixo-> fuma=sim  Padrão-> <alérgico=não, 0>
```

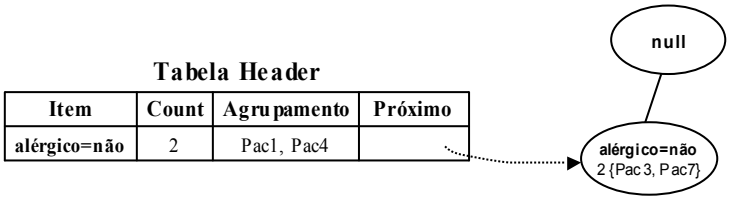

Figura 8.5 *GFP-Tree* condicional do sufixo *fuma=sim*

A função ***gfp_Growth*** é aplicada recursivamente sobre a *GFP-Tree* condicional da figura 8.5, e como a árvore tem apenas um ramo, é feita a combinação dos itens com o sufixo *fuma=sim*. O resultado pode ser observado neste trecho gerado pelo código do GFP-Growth:

```
MINERAÇÃO DA GFP-TREE CONDICIONAL DO ITEMSET->{fuma=sim}
      Combinar-> {alérgico=não, } com {fuma=sim }/[0.5/2] (1)
        Gerou-> {fuma=sim alérgico=não }/[0.5/2] (2) agrup-> 2 sup-> 0.5
              Armazenou-> {fuma=sim alérgico=não }/[0.5/2] (2)
```

A função ***gfp_Growth*** finaliza a recursão, e retorna à análise da *GFP-Tree* inicial (figura 8.2). Como a *GFP-Tree* tem mais de um ramo, será necessário construir mais *GFP-Tree*'s condicionais. O próximo item da header a ser analisado é o nó *fuma=não*. Esse item é combinado com o sufixo nulo e o itemset gerado é armazenado, conforme se pode observar neste trecho gerado pelo código do GFP-Growth:

```
ITEM [5/8] -> fuma=não, agrup-> 2, agrupamentos-> Pac7, Pac3
Armazenou-> {fuma=não }/[0.5/2] (1)
```

Analisando a GFP-Tree inicial da figura 8.2, pode-se notar que o item fuma=não ocorre em vários ramos. O mesmo pode ser observado nas informações sobre a GFP-Tree condicional a ser construída, mostradas no trecho abaixo gerado pelo código do GFP-Growth. A figura 8.6 mostra a GFP-Tree condicional gerada e sua tabela header a partir do sufixo fuma=não.

```
Sufixo-> fuma=não  Padrão-> <alérgico=sim, 1>
Sufixo-> fuma=não  Padrão-> <alérgico=sim, 1> <coriza, 3>
Sufixo-> fuma=não  Padrão-> <alérgico=sim, 1> <bactéria, 2>
```

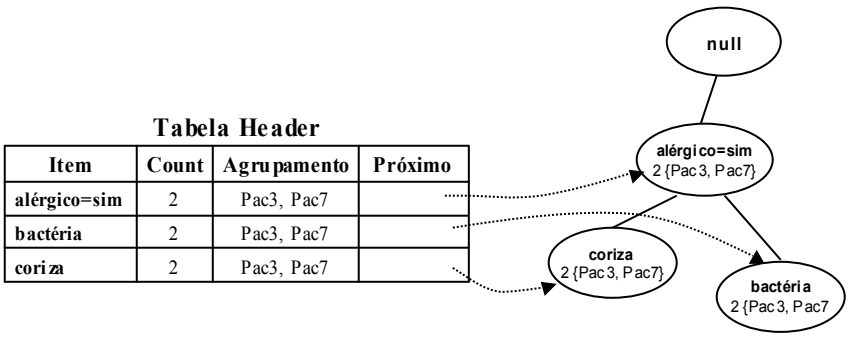

Figura 8.6 GFP-Tree condicional do sufixo fuma=não

A função ***gfp_Growth*** é aplicada recursivamente sobre a *GFP-Tree* condicional da figura 8.6. Como a árvore tem mais de um ramo, um novo sufixo será adotado para criação de

uma nova *GFP-Tree* condicional. O sufixo inicial é *fuma=não*, e será criada a *GFP-Tree* condicional do itemset {*fuma=não, coriza*}.

O item coriza é o 3º item da header da GFP-Tree condicional mostrada anteriormente. A função gfp_Growth tem como parâmetro o item fuma=não, cuja GFP-Tree condicional foi criada anteriormente. Esse parâmetro é combinado com o sufixo e é armazenado, conforme se pode observar neste trecho gerado pelo código do GFP-Growth:

```
ITEM [3/3] -> coriza, agrup-> 2, agrupamentos-> Pac7, Pac3
Armazenou-> {fuma=não coriza }/[0.5/2] (2)
```

Analisando a GFP-Tree condicional da figura 8.6 nota-se que o item coriza ocorre em apenas um ramo e possui apenas um nó ascendente. O mesmo pode ser observado nas informações sobre a GFP-Tree condicional a ser construída, mostradas no trecho abaixo gerado pelo código do GFP-Growth. A figura 8.7 mostra a GFP-Tree condicional gerada e sua tabela header a partir do sufixo {fuma=não, coriza}.

```
Sufixo-> coriza  Padrão-> <alérgico=sim, 0>
```

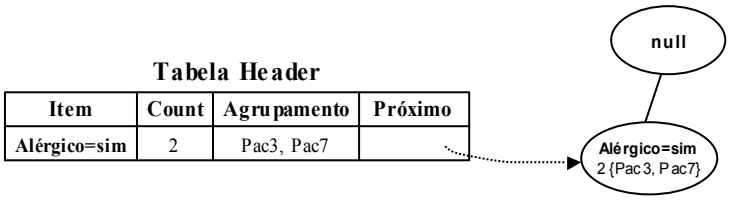

Figura 8.7 GFP-Tree condicional do sufixo {fuma=não, coriza}

A função gfp_Growth é aplicada recursivamente sobre a GFP-Tree condicional da figura 8.7, e como a árvore tem apenas um ramo, é feita a combinação dos itens com o sufixo {fuma=não, coriza}. O resultado pode ser observado neste trecho gerado pelo código do GFP-Growth:

```
MINERAÇÃO DA GFP-TREE CONDICIONAL DO ITEMSET -> {fuma=não coriza}
    Combinar-> {alérgico=sim, } com {fuma=não coriza }/[0.5/2] (2)
      Gerou-> {fuma=não coriza alérgico=sim }/[0.5/2] (3) agrup-> 2 sup-> 0.5
        Armazenou-> {fuma=não coriza alérgico=sim }/[0.5/2] (3)
```

A análise finaliza para o sufixo {fuma=não, coriza} e o algoritmo volta a processar a GFP-Tree da figura 8.6, sendo que o próximo item é bactéria, que é o 2º item da header. A função gfp_Growth tem como parâmetro o item fuma=não. Esse parâmetro é combinado com

o sufixo (bactéria) e é armazenado, conforme se pode observar neste trecho gerado pelo código do GFP-Growth:

```
ITEM [2/3] -> bactéria, agrup-> 2, agrupamentos-> Pac7, Pac3
Armazenou-> {fuma=não bactéria }/[0.5/2] (2)
```

Analisando a GFP-Tree condicional da figura 8.6 nota-se que o item bactéria ocorre em apenas um ramo e possui apenas um nó ascendente. O mesmo pode ser observado nas informações sobre a GFP-Tree condicional a ser construída, mostradas no trecho abaixo gerado pelo código do GFP-Growth. A figura 8.8 mostra a GFP-Tree condicional gerada e sua tabela header a partir do sufixo {fuma=não, bactéria}.

```
Sufixo-> bactéria  Padrão-> <alérgico=sim, 0>
```

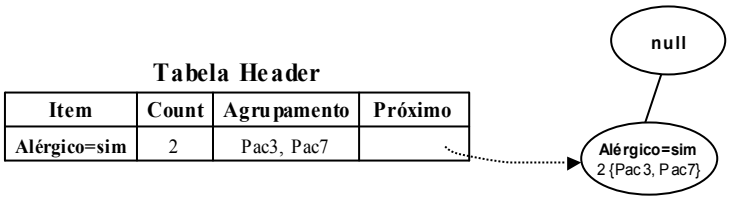

Figura 8.8 GFP-Tree condicional do sufixo {fuma=não, bactéria}

A função gfp_Growth é aplicada recursivamente sobre a GFP-Tree condicional da figura 8.8, e como a árvore tem apenas um ramo, é feita a combinação dos itens com o sufixo {fuma=não, bactéria}. O resultado pode ser observado neste trecho gerado pelo código do GFP-Growth:

```
MINERAÇÃO DA GFP-TREE CONDICIONAL DO ITEMSET -> {fuma=não bactéria}
        Combinar-> {alérgico=sim, } com {fuma=não bactéria }/[0.5/2] (2)
          Gerou-> {fuma=não bactéria alérgico=sim }/[0.5/2] (3) agrup-> 2 sup-> 0.5
                Armazenou-> {fuma=não bactéria alérgico=sim }/[0.5/2] (3)
```

A análise finaliza para o sufixo {fuma=não, bactéria} e o algoritmo volta a processar a GFP-Tree da figura 8.6, sendo que o próximo item é alérgico=sim, que é o 1º item da header. A função gfp_Growth tem como parâmetro o item fuma=não. Esse parâmetro é combinado com o sufixo (alérgico=sim) e é armazenado, conforme se pode observar neste trecho gerado pelo código do GFP-Growth:

```
ITEM [1/3] -> alérgico=sim, agrup-> 2, agrupamentos->    Pac7   Pac3
Armazenou-> {fuma=não alérgico=sim }/[0.5/2] (2)
```

O item alérgico=sim ocorre em um único nó da GFP-Tree condicional da figura 8.6, e percorrendo os nós ascendentes do mesmo, pode-se observar que existe apenas o nó raiz no mesmo ramo. Isso indica que a GFP-Tree do sufixo {fuma=não, alérgico=sim} é nula, o que faz com que a análise desse sufixo termine.

Nesse ponto, o algoritmo retorna à análise da *GFP-Tree* inicial da figura 8.2. O item em questão é *coriza*, e como a *GFP-Tree* inicial possui vários ramos, será necessário criar uma *GFP-Tree* condicional para esse item. O item é combinado com o sufixo nulo e essa combinação é armazenada, conforme se pode observar neste trecho gerado pelo código do GFP-Growth:

```
ITEM [4/8] -> coriza, agrup-> 2, agrupamentos-> Pac7, Pac3
Armazenou-> {coriza }/[0.5/2] (1)
```

O item *coriza* ocorre uma única vez na *GFP-Tree* inicial. Algumas informações sobre a *GFP-Tree* condicional de {*coriza*} podem ser observadas no trecho abaixo gerado pelo código do GFP-Growth. A *GFP-Tree* condicional gerada para o sufixo {*coriza*} pode ser observada na figura 8.9.

```
Sufixo-> coriza  Padrão-> <alérgico=sim, 1>
```

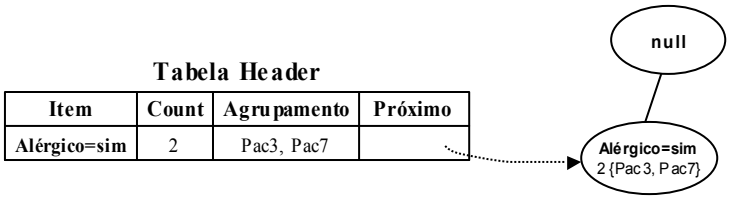

Figura 8.9 GFP-Tree condicional do sufixo {coriza}

A função ***gfp_Growth*** volta recursivamente a processar os itens da *GFP-Tree* condicional de {*coriza*}. Como a árvore tem apenas um ramo, não será necessário criar outra *GFP-Tree* condicional a partir dessa. O próximo passo é gerar combinações entre os itens do ramo e o sufixo. O resultado pode ser observado neste trecho gerado pelo código do GFP-Growth:

```
MINERAÇÃO DA GFP-TREE CONDICIONAL DO ITEMSET -> {coriza}
        Combinar-> {alérgico=sim, } com {coriza }/[0.5/2] (1)
          Gerou-> {coriza alérgico=sim }/[0.5/2] (2) agrup-> 2 sup-> 0.5
                Armazenou-> {coriza alérgico=sim }/[0.5/2] (2)
```

A função ***gfp_Growth*** volta recursivamente a processar os itens da *GFP-Tree* inicial da figura 8.2. O próximo item da header é *bactéria*, e como a árvore inicial tem vários ramos, será necessário criar uma *GFP-Tree* condicional para esse item. O item em questão é combinado com o sufixo nulo e essa combinação é armazenada, conforme se pode observar neste trecho gerado pelo código do GFP-Growth:

```
ITEM [3/8] -> bactéria, agrup-> 2, agrupamentos-> Pac7, Pac3
Armazenou-> {bactéria }/[0.5/2] (1)
```

O item *bactéria* ocorre uma única vez na *GFP-Tree* inicial. Algumas informações sobre a *GFP-Tree* condicional de {*bactéria*} podem ser observadas no trecho abaixo gerado pelo código do GFP-Growth. A *GFP-Tree* condicional gerada para o sufixo {*bactéria*} pode ser observada na figura 8.10.

```
Sufixo-> bactéria  Padrão-> <alérgico=sim, 1>
```

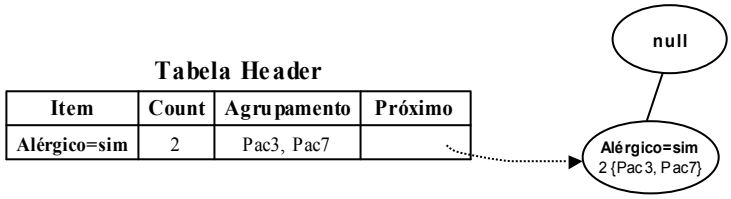

Figura 8.10 GFP-Tree condicional do sufixo bactéria

A função ***gfp_Growth*** volta recursivamente a processar os itens da *GFP-Tree* condicional de {*bactéria*}. Como a árvore tem apenas um ramo, não será necessário criar outra *GFP-Tree* condicional a partir dessa. O próximo passo é gerar combinações entre os itens do ramo e o sufixo.

```
MINERAÇÃO DA GFP-TREE CONDICIONAL DO ITEMSET -> {bactéria}
      Combinar-> {alérgico=sim, } com {bactéria }/[0.5/2] (1)
        Gerou-> {bactéria alérgico=sim }/[0.5/2] (2) agrup-> 2 sup-> 0.5
            Armazenou-> {bactéria alérgico=sim }/[0.5/2] (2)
```

A função ***gfp_Growth*** volta recursivamente a processar os itens da *GFP-Tree* inicial da figura 8.2. O próximo item da header é *alérgico=sim*, e como a árvore inicial tem vários ramos, será necessário criar uma *GFP-Tree* condicional para esse item. O item em questão é combinado com o sufixo nulo e essa combinação é armazenada, conforme se pode observar neste trecho gerado pelo código do GFP-Growth:

```
ITEM [2/8] -> alérgico=sim, agrup-> 2, agrupamentos->    Pac7   Pac3
Armazenou-> {alérgico=sim }/[0.5/2] (1)
```

O item alérgico=sim ocorre em um único nó da GFP-Tree condicional da figura 8.2, e percorrendo os nós ascendentes do mesmo, pode-se observar que existe apenas o nó raiz no mesmo ramo. Isso indica que a GFP-Tree do sufixo {alérgico=sim} é nula, o que faz com que a análise desse sufixo termine.

A função ***gfp_Growth*** volta recursivamente a processar os itens da *GFP-Tree* inicial da figura 8.2. O próximo item da header é *alérgico=não*, e como a árvore inicial tem vários ramos, será necessário criar uma *GFP-Tree* condicional para esse item. O item em questão é combinado com o sufixo nulo e essa combinação é armazenada, conforme se pode observar neste trecho gerado pelo código do GFP-Growth:

```
ITEM [1/8] -> alérgico=não, agrup-> 2, agrupamentos-> Pac4, Pac1
Armazenou-> {alérgico=não}/[0.5/2] (1)
```

O item alérgico=não ocorre em um único nó da GFP-Tree condicional da figura 8.2, e percorrendo os nós ascendentes do mesmo, pode-se observar que existe apenas o nó raiz no mesmo ramo. Isso indica que a GFP-Tree do sufixo {alérgico=não} é nula, o que faz com que a análise desse sufixo termine.

Nesse ponto, todas as combinações possíveis já foram feitas e todos os padrões já foram encontrados e estão armazenados. Os padrões encontrados são os seguintes:

```
Padrões encontrados:
Padrão -> {vírus }/[0.5/2] (1)
Padrão -> {vírus fuma=não }/[0.5/2] (2)
Padrão -> {vírus fuma=não coriza }/[0.5/2] (3)
Padrão -> {vírus fuma=não coriza alérgico=sim }/[0.5/2] (4)
Padrão -> {vírus fuma=não alérgico=sim }/[0.5/2] (3)
Padrão -> {vírus coriza }/[0.5/2] (2)
Padrão -> {vírus coriza alérgico=sim }/[0.5/2] (3)
Padrão -> {vírus alérgico=sim }/[0.5/2] (2)
Padrão -> {inflamação }/[0.5/2] (1)
Padrão -> {inflamação fuma=não }/[0.5/2] (2)
Padrão -> {inflamação fuma=não bactéria }/[0.5/2] (3)
Padrão -> {inflamação fuma=não bactéria alérgico=sim }/[0.5/2] (4)
Padrão -> {inflamação fuma=não alérgico=sim }/[0.5/2] (3)
Padrão -> {inflamação bactéria }/[0.5/2] (2)
Padrão -> {inflamação bactéria alérgico=sim }/[0.5/2] (3)
Padrão -> {inflamação alérgico=sim }/[0.5/2] (2)
Padrão -> {fuma=sim }/[0.5/2] (1)
Padrão -> {fuma=sim alérgico=não }/[0.5/2] (2)
Padrão -> {fuma=não }/[0.5/2] (1)
Padrão -> {fuma=não coriza }/[0.5/2] (2)
Padrão -> {fuma=não coriza alérgico=sim }/[0.5/2] (3)
Padrão -> {fuma=não bactéria }/[0.5/2] (2)
Padrão -> {fuma=não bactéria alérgico=sim }/[0.5/2] (3)
```

```
Padrão -> {fuma=não alérgico=sim }/[0.5/2] (2)
Padrão -> {coriza }/[0.5/2] (1)
Padrão -> {coriza alérgico=sim }/[0.5/2] (2)
Padrão -> {bactéria }/[0.5/2] (1)
Padrão -> {bactéria alérgico=sim }/[0.5/2] (2)
Padrão -> {alérgico=sim }/[0.5/2] (1)
Padrão -> {alérgico=não }/[0.5/2] (1)
```

8.6. Método *generateRules*

O próximo passo é gerar as regras, analisando a medida *conf_a* das mesmas. Essa tarefa é executada pelo método ***generateRules***. Nessa função, inicialmente os padrões são ordenados de acordo com seu tamanho, de forma que os menores serão tratados primeiro. Para cada padrão, é construída uma string com seus valores de peso, as quais serão escritas no arquivo de resultado.

Se o padrão tem mais de um item, então serão feitas combinações entre seus itens para gerar as regras. O processo ocorre da seguinte forma: para cada item de cada padrão, esse item será removido para ser usado do lado direito da regra gerada. O lado esquerdo será composto pelos itens que permaneceram no padrão. Dessa forma, a confiança da regra, será igual ao suporte do padrão inicial pelo suporte do padrão formado pelos itens remanescentes (lado esquerdo). O valor desse último pode ser obtido procurando por esse padrão previamente armazenado.

A regra gerada é formada pelos itens remanescentes do padrão no lado esquerdo e pelo item removido no lado direito. Caso a regra satisfaça a mínima *conf_a*, ela é escrita no arquivo de resultado, juntamente com seus valores de *sup_a*, *conf_a* e com a string gerada no início da função com os valores de peso.

Nesse exemplo foram geradas 56 regras, sendo que uma parcela dessas regras é mostrada abaixo:

```
alérgico=não -> fuma=sim
        sup=50.0%, c=100.0%, peso(alérgico=não)=50.0%, peso(fuma=sim)=66.66667%
fuma=sim -> alérgico=não
       sup=50.0%, c=100.0%, peso(alérgico=não)=50.0%, peso(fuma=sim)=66.66667%
coriza -> fuma=não
        sup=50.0%, c=100.0%, peso(coriza)=100.0%, peso(fuma=não)=50.0%
alérgico=sim fuma=não -> bactéria
        sup=50.0%, c=100.0%, peso(alérgico=sim)=66.66667%, peso(bactéria)=100.0%,
        peso(fuma=não)=50.0%
bactéria fuma=não -> alérgico=sim
        sup=50.0%, c=100.0%, peso(alérgico=sim)=66.66667%, peso(bactéria)=100.0%,
        peso(fuma=não)=50.0%
```

alérgico=sim coriza fuma=não -> vírus
sup=50.0%, c=100.0%, peso(alérgico=sim)=66.66667%, peso(coriza)=100.0%, peso(fuma=não)=50.0%, peso(vírus)=100.0%
coriza fuma=não vírus -> alérgico=sim
sup=50.0% c=100.0% peso(alérgico=sim)=66.66667% peso(coriza)=100.0% peso(fuma=não)=50.0% peso(vírus)=100.0%
alérgico=sim bactéria fuma=não -> inflamação
sup=50.0% c=100.0% peso(alérgico=sim)=66.66667% peso(bactéria)=100.0% peso(fuma=não)=50.0% peso(inflamação)=100.0%
alérgico=sim bactéria inflamação -> fuma=não
sup=50.0% c=100.0% peso(alérgico=sim)=66.66667% peso(bactéria)=100.0% peso(fuma=não)=50.0% peso(inflamação)=100.0%