UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Mineração de Regras de Associação no Problema da Cesta de Compras Aplicada ao Comércio Varejista de Confecção

por

SANDRO DA SILVA CAMARGO

Dissertação submetida à avaliação,
como requisito parcial para a obtenção do grau de Mestre
em Ciência da Computação

Prof. Dr. Paulo Martins Engel
Orientador

Porto Alegre, abril de 2002.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Camargo, Sandro da Silva

Mineração de regras de associação no problema da cesta de compras aplicada ao comércio varejista de confecção / por Sandro da Silva Camargo. - Porto Alegre: PPGC da UFRGS, 2002.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Sul. Programa de Pós - Graduação em Ciência da Computação, Porto Alegre, BR-RS, 2002. Orientador : Engel, Paulo Martins.

1. Descoberta de Conhecimento em Bancos de Dados. 2. Mineração de Dados. 3. Regras de Associação. 4. Comércio Varejista. I.Engel, Paulo Martins. II. Título.

Agradecimentos

A minha família.

Ao Prof. Dr. Paulo Martins Engel por ter, desde o princípio, acreditado na idéia proposta, e pelo indispensável auxílio fornecido durante o desenvolvimento to trabalho.

Ao Prof. Hércules Prado pela motivação e auxílio prestado.

Aos colegas do Projeto SUS.

A URCAMP e a UFRGS.

Sumário

Lista de Abreviaturas

KDD	*Knowledge Discovery in Databases* ou Descoberta de conhecimento em banco de dados
SGBD	Sistema de Gerenciamento de Banco de Dados
TID	*Transaction ID* ou Identificador de Transação

Lista de Símbolos

I	Conjunto de itens
I_k	Um item contido em I
T	Banco de dados de transações
t	Uma transação contida no conjunto T
X	Subconjunto de I
Y	Subconjunto de I
$t[k]=1$	I_k está contido em t
$t[k]=0$	I_k não está contido em t
c	Confiança da regra
L_k	Conjunto de itens freqüentes com k itens
C_k	Conjunto de itens candidatos com k itens
D^i	Conjunto de dados locais com paralelismo de dados
P^i	Processador i
C_1^i	Conjunto de itens candidatos com l itens no processador i

Lista de Figuras

Lista de Tabelas

Resumo

A maioria das empresas interage com seus clientes através de computadores. Com o passar do tempo está armazenado nos computadores um histórico da atividade da empresa que pode ser explorado para a melhoria do processo de tomada de decisões. Ferramentas de descoberta de conhecimento em bancos de dados exploram este histórico a fim de extrair vários tipos de informação.

Um dos tipos de informação que pode ser extraída destes tipos de bancos de dados são as regras de associação que consistem em relacionamentos ou dependências importantes entre itens tal que a presença de alguns itens em uma transação irá implicar a presença de outros itens na mesma transação.

Neste trabalho são aplicadas técnicas de descoberta de conhecimento na área do comércio varejista de confecção. Foram detectadas algumas peculiaridades dos bancos de dados desta área sendo proposto um novo algoritmo para melhorar o desempenho da tarefa de extração de regras de associação. Para a validação dos resultados apresentados pelo algoritmo foi desenvolvido o protótipo de uma ferramenta para extração de regras de associação. Foram realizados experimentos com bancos de dados reais de uma empresa da área de comércio varejista de confecção para análise de desempenho do algoritmo.

Palavras-chave: descoberta de conhecimento, mineração de dados, regras de associação, comércio varejista.

TITLE: "MINING ASSOCIATION RULES IN A BASKET SALES PROBLEM APPLIED TO A CLOTHING RETAIL"

Abstract

Most of the companies interacts with their customers through computers. In the course of time; computers will store a report of company´s activity which can be explored to improve the process of making decision. Tools of knowledge discovery in databases explore this historical in order to extract several kinds of information.

One of the kinds of information that can be extracted of these databases is the association rules that consist of relationships or important dependences among items such a that the presence of some items in a transaction that will implicate the presence of other items in the same transaction.

In this work are is applied knowledge discovery techniques in the clothing retail area. Some database peculiarities were detected in this area being proposed a new algorithm to improve the acting of extraction association rules task. For the validation of the results presented by the algorithm the prototype of a tool was developed for extraction of association rules. Experiments were accomplished with real databases of a company of the clothing retail area for algorithm performance analysis.

Keywords: knowledge discovery, data mining, association rules, retail trade.

1 Introdução

Atualmente a maioria das empresas possui armazenados em seu banco de dados o histórico de sua atividade de alguns anos, tais dados formam uma mina em potencial de valiosas informações sobre o negócio, que podem ser exploradas a fim de melhorar a qualidade de vários processos dentro da empresa [AGR93a]. A análise destes dados pelas técnicas tradicionais, onde grande parte do trabalho é realizado por especialistas humanos, torna-se inviável pela grande quantidade de dados, o que cria uma necessidade urgente por novas técnicas e ferramentas que possam inteligente e automaticamente transformar os dados processados em informação útil e conhecimento.

Estas novas técnicas e ferramentas são o objeto de estudo da área de descoberta de conhecimento em banco de dados. Descoberta de conhecimento consiste em um processo não trivial de extração de informação implícita, previamente desconhecida e potencialmente útil, dos bancos de dados. Informação de alto nível pode ser extraída de conjuntos relevantes de dados do banco de dados e ser investigada de diferentes ângulos, e grandes bancos de dados servem como uma fonte rica e confiável para geração e verificação de conhecimento. A descoberta de conhecimento pode ser aplicada ao gerenciamento de informação, processamento de consulta, tomada de decisão, controle de processos e muitas outras aplicações [CHE96].

Dentre as várias aplicações da descoberta de conhecimento em banco de dados, uma das que tem o maior potencial de interesse do comércio varejista é a que focaliza a descoberta de regras de associação em bancos de dados de transações de vendas.

A descoberta de regras de associação é freqüentemente referida na literatura como o problema da análise de cestas de compras, que foi formulado por Agrawal [AGR93]. Neste problema, tem-se uma grande população de transações que contém informações sobre instâncias de transações (cestas); tais cestas são compostas por itens. O objetivo é encontrar itens que freqüentemente ocorrem juntos em várias cestas [CHE96], onde a presença de um item implica a presença de outro na mesma transação. A tarefa de mineração de regras de associação é essencialmente descobrir associações fortes entre itens em grandes bancos de dados [AGR93, AGR94, BRI96, CHE96].

Desde que o problema da mineração de regras de associação foi introduzido, diversos autores têm se preocupado em pesquisar e apresentar novos algoritmos para efetuar esta tarefa de forma eficiente.

Análise da cesta de compras busca descobrir que produtos os clientes tendem comprar em conjunto, a fim de fornecer discernimento sobre o comportamento dos consumidores e fornecer um suporte de alto nível para o processo de tomada de decisão [BER 97].

De acordo com [CAM2000], em um banco de dados do comércio varejista de confecção a média de itens por transação é geralmente inferior a 2; por outro lado, dentre

os algoritmos estudados foi detectada a preocupação na aplicação dos mesmos em bancos de dados onde a média de itens por transação é bem superior a esta, tais como os bancos de dados de supermercados. Em [AGR94], [SRI96] e [SRI97] são descritos testes de desempenho com a média de itens por transação variando entre 5 e 20. Em [BAY99] e [BAY99a] os testes de desempenho são feitos em bancos de dados com médias de 43 e 49 itens por transação. Como a média de itens por transação é um fator determinante no desempenho do algoritmo, detectou-se a necessidade de novos algoritmos dirigidos a bancos de dados com média de itens por transação menores, tais como comércio de confecção, livrarias, lojas de CD, ferragens dentre outros.

Foi desenvolvido um protótipo para descoberta de regras de associação que implementa o algoritmo proposto neste trabalho que é otimizado para um banco de dados de comércio varejista de confecção.

1.1 Motivação

O incremento da competição entre as empresas, principalmente nos setores mais lucrativos da economia, faz com que as empresas recorram a suas bases de dados objetivando aumentar o conhecimento sobre o negócio a fim de melhorar a qualidade nos processos de tomada de decisão. Muitas empresas já reconhecem a descoberta de conhecimento em bancos de dados como uma área importante com uma enorme oportunidade para aumentar seus rendimentos [BRU98].

Atualmente, a maioria das empresas possui uma memória histórica de suas atividades de alguns anos. Este conjunto de dados forma uma mina em potencial de valiosas informações sobre o negócio, que podem ser exploradas a fim de melhorar a qualidade das decisões dentro da empresa [AGR93a]. As empresas podem se utilizar da ferramentas de descoberta de regras de associação como uma vantagem no extremamente competitivo mercado do comércio varejista de confecção, sendo visível a carência de ferramentas disponíveis no mercado que possam ser utilizadas para esta finalidade e acima de tudo, otimizadas para o ambiente em estudo.

Na área de comércio varejista a aplicação de ferramentas de descoberta de regras de associação pode ser utilizada para entender o comportamento do cliente, determinação do *layout* da loja, determinação de produtos sujeitos a promoção e fornecer suporte aos processos de tomada de decisão[BER97].

1.2 Objetivos

Objetivo geral

Desenvolver um algoritmo que otimize a tarefa de mineração de regras de associação em bancos de dados com baixo tamanho médio de transação a fim de aumentar a eficiência desta tarefa.

Objetivos específicos

a) analisar alguns algoritmos existentes para descoberta de regras de associação em bancos de dados. Dentre os algoritmos a serem estudados estão: algoritmos sequenciais [AGR93] [AGR94] [BAY99] e algoritmos paralelos [AGR96a] e [HAN97];
b) comparar os algoritmos estudados sob aspectos como tempo de execução, espaço necessário para execução do processo, confiabilidade dos resultados, e identificar aspectos positivos e negativos de cada um dos algoritmos;
c) desenvolver um algoritmo otimizado para a área do comércio varejista de confecção visando melhora de desempenho;
d) implementar o protótipo da ferramenta em uma linguagem de programação orientada a objetos;
e) validar os resultados apresentados pelo algoritmo.

1.3 Metodologia

Foi realizada uma revisão bibliográfica sobre o problema da descoberta de regras de associação e selecionados alguns dos algoritmos mais utilizados para descoberta de regras de associação em bancos de dados de transação. Alguns dos algoritmos selecionados foram implementados.

Para testes dos algoritmos foi utilizada uma população da pesquisa experimental com um conjunto de transações de uma empresa de comércio varejista com uma história de 1 mês da empresa, com aproximadamente 20 mil itens, 2266 transações, 4380 detalhes de transação.

Foram realizados vários experimentos sobre esta base de dados visando-se encontrar os pontos críticos dos algoritmos estudados em termos de desempenho e foram buscadas soluções para resolver a perda de desempenho nestes pontos críticos. Com base nos algoritmos estudados é proposto um algoritmo, chamado MiRABIT (Mineração de Regras de Associação em bancos de dados com Baixa média de Itens por Transação), otimizado para a área do comércio varejista de confecção.

Foi implementado o protótipo de uma ferramenta que utiliza o algoritmo MiRABIT na linguagem de programação orientada a objetos DataFlex que acessa o banco de dados definido como população através de um driver nativo.

1.4 Organização deste trabalho

Este trabalho divide-se em cinco capítulos. No capítulo 2 são apresentados fundamentos de descoberta de conhecimento, definições de mineração de dados e uma descrição geral de técnicas utilizadas no processo de mineração. No capítulo 3 são apresentadas as regras de associação, além de uma descrição formal destas regras e a descrição do processo básico com os passos necessários para solução do problema da análise de cesta de compras. No capítulo 4 é apresentado o algoritmo MiRABIT, assim como a ferramenta desenvolvida. No capítulo 5 é realizada a validação da solução proposta.

2 Fundamentos Teóricos

2.1 Descoberta de Conhecimento em Bancos de Dados

O termo Descoberta de Conhecimento em Bancos de Dados, ou KDD (*Knowledge Discovery in Databases*), foi introduzido no final da década de 80 para se referir ao amplo processo de encontrar conhecimento a partir de dados e enfatizar o mais alto nível de aplicações particulares de mineração de dados [FAY96].

Em [FAY96] o processo de KDD é definido como:

"*Um processo não trivial de identificação de padrões válidos, desconhecidos, potencialmente úteis e compreensíveis.*"

Atualmente o incremento da competição entre as empresas, principalmente nos setores mais lucrativos da economia, faz com que as empresas recorram a suas bases de dados objetivando conhecer melhor os seus clientes a fim de sempre estarem prontas para suprir as demandas dos mesmos e, através do conhecimento do perfil dos seus clientes, buscar novos clientes em potencial com perfis semelhantes. Muitas companhias já reconhecem a descoberta de conhecimento em bancos de dados como uma área importante com uma enorme oportunidade para aumentar seus rendimentos[BRU98].

O processo de KDD, executado sobre um conjunto, subconjunto ou amostra de dados, é composto por uma série de etapas [FAY96]:

- Pré - processamento: consiste em um conjunto de atividades com a finalidade de preparar o banco de dados para o processo de mineração. Esta etapa inclui ações como: eliminação de problemas como ruído e dados incompletos, a seleção manual ou automática dos atributos relevantes, amostragem, transformações de representação, entre outras.
- Mineração de dados: é o núcleo do processo de descoberta de conhecimento, é onde são aplicados os algoritmos de mineração para a extração do conhecimento dos dados pré - processados.
- Pós - processamento: avaliação e interpretação das descobertas, seleção e ordenação das descobertas interessantes, mapeamentos de representação de conhecimento, apresentação do conhecimento resultante de todo este processo geralmente na forma de gráficos ou relatórios.

O processo de descoberta de conhecimento em bancos de dados é iterativo e interativo, envolvendo vários passos com muitas decisões a serem tomadas pelo usuário. O processo é iterativo por haver um conjunto de passos a serem executados sendo possível, de qualquer passo, retornar a algum passo já previamente executado. O processo é interativo por ser semi-automático, necessitando do usuário para modificar e especificar parâmetros do processo, refinando os padrões encontrados de forma que coincidam com metas estabelecidas.

A figura 2.1 demonstra o processo básico do KDD.

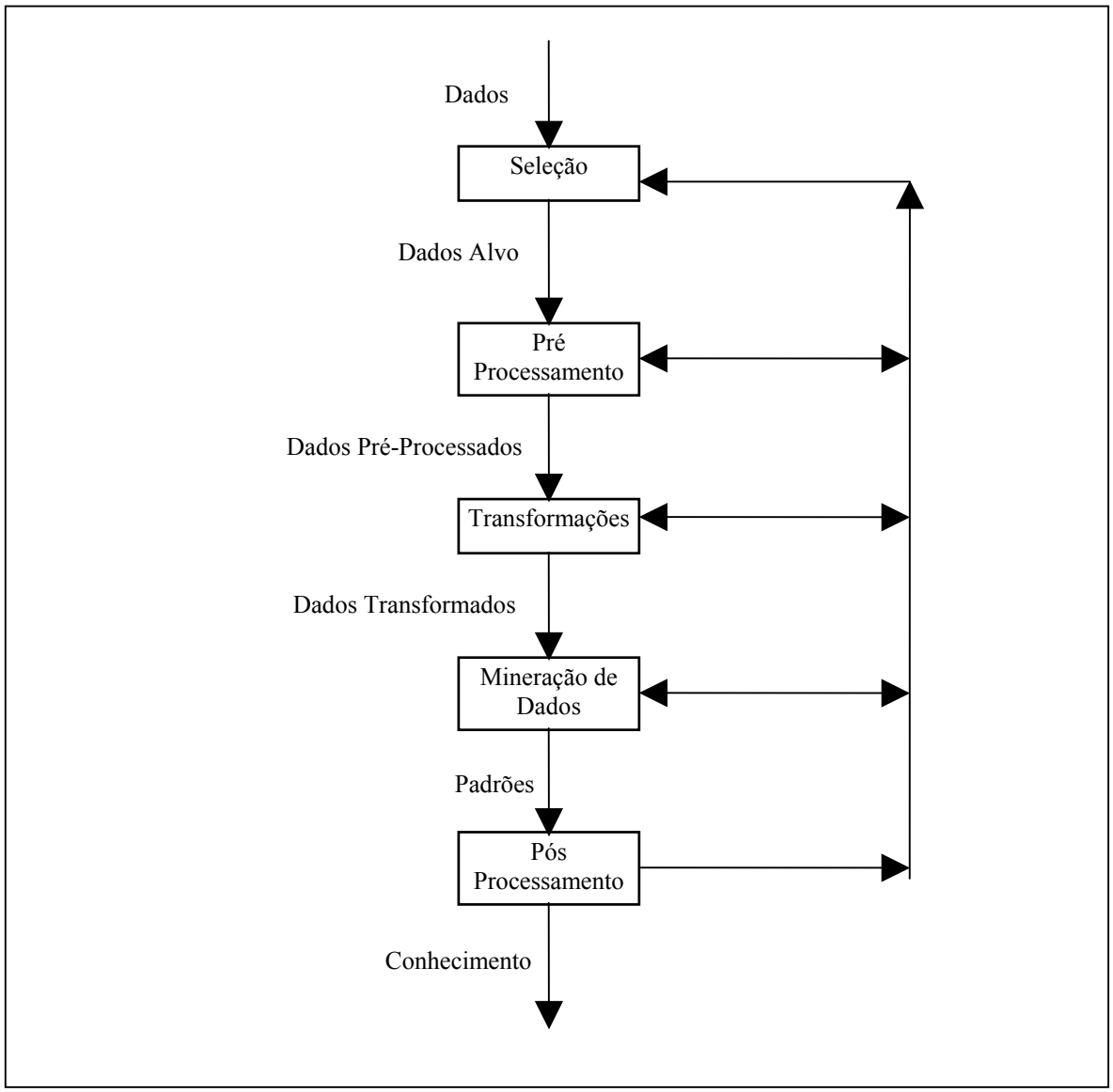


FIGURA 2.1 - O processo de KDD [FAY96]

2.1.1 Requisitos dos sistemas de descoberta de conhecimento

Há algumas características desejáveis em sistemas de descoberta de conhecimento, dentre elas podemos citar [CHE96]:

- Necessidade de manuseio de diferentes tipos de dados. Visto que atualmente os bancos de dados relacionais dominam o mercado de SGBD, constitui-se em uma característica fundamental a um sistema de descoberta de conhecimento a capacidade de mineração eficiente em bancos de dados relacionais, assim como a habilidade de tratamento de

dados complexos como dados dos tipos: multimídia, hipertexto, meta dados, dados espaciais, temporais, transacionais, entre outros. Sistemas de descoberta de conhecimento poderosos devem estar habilitados a minerarem diferentes tipos de dados, mas deve-se ter ciência do fato que dificilmente um sistema estará capacitado a trabalhar com todos os tipos de dados.

- Outras características altamente desejáveis em algoritmos de mineração de dados são a eficiência e a escalabilidade. Por eficiência entende-se a capacidade de extrair informações úteis do banco de dados, e a escalabilidade constitui-se no crescimento não exponencial do tempo de execução em função do incremento da quantidade de registros a serem pesquisados no banco de dados.

2.1.2 Desafios da descoberta de conhecimento

O conhecimento descoberto deve descrever com a maior exatidão possível o conteúdo do banco de dados, devendo ser indubitavelmente verdadeiro. A imperfeição deve ser expressa por medidas de incerteza, na forma de regras de aproximação ou regras quantitativas, assim como ruído e dados excepcionais devem ser manipulados adequadamente em sistemas de mineração de dados. Isto também motiva uma sistemática de estudo de medidas de qualidade da descoberta de conhecimento, incluindo interesse e confiabilidade, pela construção de modelos e ferramentas estatísticas, analíticas e simulativas.

As requisições do usuário, assim como o conhecimento descoberto devem ser apresentados em linguagem de alto nível, sob forma de relatórios, gráficos, ou qualquer forma passível de ser facilmente compreendida pelo usuário final, o que implica a utilização de técnicas de representação de conhecimento expressivas por parte do sistema de descoberta de conhecimento.

O sistema deve ser uma ferramenta interativa, que permita a intervenção do usuário para redefinir, dinamicamente, a qualquer momento, o foco da pesquisa, possibilitando que os resultados obtidos estejam o mais próximo possível do objetivo determinado inicialmente. Também deve fornecer os resultados da mineração em diversos níveis de abstração e sob diversos pontos de vista a fim de fornecer subsídios para as intervenções do usuário no sistema.

A utilização de fontes de dados em bancos de dados distribuídos e heterogêneos também constitui-se em um requisito desejável para um sistema de descoberta de conhecimento, o qual deve ser hábil para lidar com todas as dificuldades que geralmente decorrem da utilização de tais arquiteturas de bancos de dados tais como: incompletudes, inconsistências, diferentes formatos de dados, etc .

A disponibilização dos dados sob diversos ângulos em diferentes níveis de abstração, consiste em uma ameaça à segurança dos dados, possibilitando eventualmente que pessoas não autorizadas a tenham acesso a determinado nível de abstração.

2.2 Mineração de Dados

A mineração de dados, ou *data mining*, é uma etapa na descoberta de conhecimento em bancos de dados que consiste na análise automática ou semi-automática de grandes volumes de dados sob diferentes perspectivas, a fim de descobrir informações úteis e conhecimento previamente desconhecidos, através da utilização de técnicas que envolvem métodos matemáticos, algoritmos e heurísticas para descobrir padrões e regularidades entre os dados pesquisados [BRU98]. Estes grandes volumes de dados servem como fontes ricas e confiáveis para geração e verificação de conhecimento [CHE96].

Em [FAY96], mineração de dados é definido como:

"Um passo do processo de KDD que consiste em algoritmos particulares de data mining que, sobre algumas limitações aceitáveis de eficiência computacional, produzem uma série particular de padrões sobre os dados."

2.2.1 Tarefas da mineração de dados

A mineração de dados pode executar um conjunto limitado de tarefas que podem ser divididas em seis tipos distintos [BER97]: classificação, regressão, predição, regras de associação, agrupamento por similaridade e descrição.

Em [AGR96a] é apresentada uma outra divisão das tarefas de mineração; são apresentadas três classes de problemas: classificação (que englobaria as tarefas de classificação, regressão e agrupamento por similaridade), associações (assemelha-se as regras de associação ou análise da cesta de compras) e seqüências (assemelha-se a tarefa de predição).

Diferentes técnicas podem ser utilizadas para execução das tarefas de mineração, tais como: árvores de decisão, redes neurais, raciocínio baseado em memória e algoritmos genéticos.

Classificação

O processo de classificação consiste em examinar as características de um novo objeto e alocá-lo a uma classe de um conjunto de classes previamente definido. Os objetos a serem classificados são representados por tuplas em um banco de dados, suas características são os campos da tupla e a ação de classificação consiste na atualização de cada registro pelo preenchimento em um campo com a classe. A tarefa de classificação é caracterizada por um conjunto de classes bem definidos e por um conjunto de treinamento de exemplos pré - classificados. Árvores de decisão, redes neurais e raciocínio baseado em

memória são técnicas que se ajustam muito bem à classificação, assim como análise de ligação também pode ser utilizada para classificação em certos casos.

Regressão

O processo de regressão é semelhante ao processo de classificação, a principal diferença entre ambos é que o último lida com valores discretos enquanto o outro, com valores contínuos. Como conseqüência disso temos que através do processo de regressão é possível ordenar registros individualmente. Por exemplo, se pelo processo de classificação classificamos registros como 0 ou 1, pelo processo de regressão é possível classificarmos registros com qualquer valor real entre 0 e 1. Redes neurais se ajustam muito bem a tarefas de regressão.

Predição

O processo de predição também é semelhante aos processos anteriores exceto pelo fato de que os registros possuem dados temporais são classificados de acordo com alguma predição de comportamento futuro ou predição de valor futuro. Tanto classificação como regressão podem ser adaptadas para uso em predição através dos exemplos de treinamento onde os valores passados das variáveis a serem preditas são conhecidos, de acordo com os dados históricos para estes exemplos. Os dados históricos são usados para construir um modelo que explica o comportamento corrente observado. A técnica de análise da cesta de compras, usada para descobrir que itens provavelmente serão comprados juntos, pode ser adaptada ao modelo de que compras futuras ou ações tendem a ser tomadas de acordo com os dados correntes. As técnicas de análise da cesta de compras, raciocínio baseado em memória, árvores de decisão e redes neurais podem ser utilizadas no processo de predição.

Regras de associação ou análise da cesta de compras

Considere-se um ambiente de comércio onde os itens comprados por um cliente em um único momento formam uma transação [AGR96a]. O processo de agrupamento por afinidade consiste em determinar que itens co - ocorrem em um conjunto de transações. Agrupamento por afinidade é uma abordagem simples para geração de regras de associação dos dados que consiste no foco deste trabalho e será discutida na seção 2.3.

Agrupamento por similaridade

O processo de agrupamento por similaridade consiste em dividir uma população heterogênea em um número de subgrupos mais homogêneos. Estes subgrupos não são pré - definidos e também não há exemplos assim como ocorre no processo de classificação.

Agrupamento por similaridade é freqüentemente utilizado como um prelúdio para alguma outra forma de mineração de dados.

Descrição

O processo de descrição tem como propósito simplesmente descrever os relacionamentos em um complicado banco de dados a fim de aumentar a nossa compreensão sobre pessoas, produtos ou processos. Um bom processo de descrição de um comportamento freqüentemente irá sugerir uma explicação para o comportamento.

2.2.2 Abordagens da mineração de dados

A mineração de dados pode ser abordada de duas maneiras diferentes[BER97]:

- Mineração de dados *top-down* ou teste de hipóteses, onde é analisado o comportamento passado em um banco de dados para verificar ou refutar noções, idéias ou relacionamentos pré-concebidos entre os dados.
- Mineração de dados *bottom-up* ou descoberta de conhecimento, onde não há pré - concepções, os dados são analisados e deles é retirado o conhecimento. A descoberta de conhecimento pode ser feita de duas maneiras distintas: dirigida e não dirigida.
  - Descoberta de conhecimento dirigida tenta explicar ou categorizar algum campo de dados em particular.
  - Descoberta de conhecimento não dirigida tenta encontrar padrões ou similaridades entre grupos de registros sem uso de um campo particular como alvo ou de conjuntos pré - definidos de classes.

2.3 Mineração de Regras de Associação na Cesta de Compras

Dado um banco de dados de transações de vendas, análise da cesta de compras é uma forma de encontrar grupos de itens que tendem a ocorrer juntos em uma transação, ou seja, as associações importantes entre itens tal que a presença de alguns itens em uma transação implicará a presença de outros itens na mesma transação. Um modelo matemático foi proposto em [AGR93] para resolver o problema de mineração de regras de associação.

2.3.1 Modelo formal da mineração de regras de associação na cesta de compras proposto por [AGR94]

Temos que $I = \{i_1, i_2, i_3, ..., i_m\}$é um conjunto de atributos binários, chamados itens. Temos que $D = \{T_1, T_2, T_3, ..., T_m\}$ é um banco de dados de transações, onde cada transação T é um conjunto de itens tal que $T \subseteq I$. Cada transação T é representada como um vetor

binário, com t[k]=1 se T contém o item i_k, e t[k]=0 caso contrário. Há uma tupla no banco de dados para cada transação. Considerando que X é um conjunto de alguns itens em I, uma transação T contém X se para todos itens i_k em X, t[k]=1.

Uma regra de associação consiste em uma implicação da forma $X => Y$, onde $X \subset I$, $Y \subset I$ e $X \cap Y = \varnothing$, ou seja, X é um conjunto de alguns itens de I, e Y é um conjunto de itens em I que não está presente em X. A regra $X => Y$ é satisfeita no conjunto de transações T com o fator de confiança $0 \leq c \leq 1$, se no mínimo c% das transações em T que satisfizerem X também satisfaçam Y. Será utilizada a notação $X => Y \mid c$ para especificar que a regra $X => Y$ tem um fator de confiança de c.

Dado um conjunto de transações T, é necessário gerar todas as regras que satisfaçam certas restrições, sob dois aspectos diferentes:

1. Limites sintáticos: envolvem restrições em itens que podem aparecer em uma regra. Por exemplo, pode-se estar interessado apenas em regras que tenham um item específico i_x aparecendo no conseqüente, ou regras que têm um item específico i_y aparecendo no antecedente. Combinações sobre limites também são possíveis – podem ser necessárias todas as regras que têm itens de algum conjunto predefinido X aparecendo no conseqüente, e itens de algum outro conjunto Y aparecendo no antecedente.

2. Limites de suporte: referem-se ao número de transações em T que obedecem a regra. O suporte para uma regra é definido como sendo uma fração de transações em T que satisfaçam a união dos itens em conseqüente e antecedente de uma regra. A notação $s(i_1)=7$ significa que o item 1 possui suporte igual a 7.

Conceitualmente, enfatiza-se a diferença entre suporte e confiança: enquanto confiança é uma medida da força da regra, suporte corresponde a uma significância estatística. Além da significância estatística, outra motivação para o limite de suporte vem do fato de geralmente haver interesse apenas nas regras com suporte maior que algum limiar mínimo por razões do negócio. Se o suporte não é grande o suficiente, isto significa que a regra não vale a pena ser considerada ou que ela é simplesmente menos importante. Vários autores eliminam estas regras por razões de otimização de tempo de execução do processo.

2.3.2 Decomposição do modelo

Nesta formulação, o problema de mineração de regras pode ser decomposto em dois subproblemas [AGR93]:

1. Geração de todas combinações de itens que têm uma fração de suporte transacional maior que um certo limiar, chamado suporte mínimo. Estas combinações são chamadas de itens freqüentes, e todas outras combinações com suporte menor que o limiar são chamadas de itens não freqüentes. Limites sintáticos adicionalmente limitam as combinações admissíveis contribuindo para a diminuição do tempo de execução. Por

exemplo, se apenas regras envolvendo um item i_x em um antecedente são interessantes, então é suficiente gerar apenas aquelas combinações que contenham i_x no antecedente.

2. Para um dado conjunto de itens freqüentes $Y = \{i_1, i_2, ..., i_k\} | k >= 2$, gerar todas as regras que contenham itens do conjunto $i_1, i_2, ..., i_k$. O antecedente de cada uma destas regras será um subconjunto X de Y tal que X tem k-1 itens, e o conseqüente será o item $Y - X$. Para gerar uma regra $X => Y | c$, onde $X = i_1, i_2, ..., i_{j-1}, i_{j+1}, i_k$, é necessário dividir o suporte de Y pelo suporte de X. Se a taxa é maior que c e que o fator de confiança mínimo definido então a regra é satisfeita com o fator de confiança c, caso contrário não.

Se o conjunto de itens Y é freqüente, então todo subconjunto de Y também será freqüente, e, também, todas regras derivadas de Y satisfazem o limite de suporte se Y satisfaz o limite de suporte.

O primeiro passo faz sucessivas passagens sobre o banco de dados e é responsável pela maior parte do tempo de processamento, deste resulta o conjunto de itens freqüentes e seus respectivos contadores de suporte. Já o segundo passo é pouco oneroso em tempo de processamento, de onde resulta o conjunto de regras com seus suporte e confiança.

2.3.3 Número de passagens x ganho de desempenho

Conforme [AGR93] na abordagem mais direta de descoberta de conjuntos de itens, cada conjunto de itens presente em qualquer das tuplas será medido em uma passagem, o que evitaria a necessidade de múltiplas passagens sobre o banco de dados para apresentação do conjunto de itens candidatos. No pior caso, esta abordagem irá requerer a existência de $2^m - 1$ contadores correspondendo a todos os subconjuntos de itens I, onde m é o número de itens em I. Suponhamos que uma transação tenha 5 itens, somente para esta transação devem ser gerados $2^5 - 1 = 31$ contadores, conforme demonstrado na tabela 2.1.

TABELA 2.1 - Contadores gerados

Tamanho do conjunto de itens	Possibilidades
1	5
2	10
3	10
4	5
5	1
Total	31

Aplicando-se este algoritmo sobre um banco de dados com 20000 itens tem-se então $2^{20000} - 1$ possibilidades, o que cria um grande problema de gerenciamento de memória e faz com que o tempo de processamento se eleve exponencialmente, tornando esta técnica inviável para aplicação em bancos de dados com uma grande quantidade de itens.

A melhor abordagem é medir na *k*-ésima passagem apenas conjuntos de itens com exatamente *k* itens [AGR93]. Após a *k*-ésima passagem, será feita a (*k*+1)-ésima passagem onde serão contados conjuntos de itens com (*k*+1) itens que serão obtidos através da extensão dos conjuntos de itens encontrados na passagem anterior. Nesta abordagem serão executadas diversas passagens sobre o banco de dados mas em contrapartida o conjunto de itens freqüentes encontrados será menor. Também é executada uma poda dentre as possibilidades visto que se um conjunto de itens não é freqüente, qualquer uma de suas extensões também não será freqüente.

A tabela 2.2 mostra o banco de dados de exemplo com 9 transações e 5 itens.

TABELA 2.2 - Banco de dados de exemplo

TID	Itens
1	1,3
2	3,5
3	1,2,4
4	2,4
5	1,2,3,4,5
6	1
7	1,2,3,4
8	1,2,4
9	1,2

Tomando-se como base o banco de dados descrito na tabela 2.2 e supondo-se que o suporte mínimo seja 1, as tabelas 2.3 a 2.7 demonstram todas as combinações possíveis de conjuntos de itens freqüentes. L_k é o conjunto de itens freqüentes na passagem *k*.

TABELA 2.3 - Itens Freqüentes
1ª Passagem

L_1	Suporte
{1}	7
{2}	6
{3}	4
{4}	5
{5}	2

TABELA 2.4 - Itens Freqüentes
2ª Passagem

L_2	Suporte
{1,2}	5
{1,3}	3
{1,4}	4
{1,5}	1
{2,3}	2
{2,4}	5
{2,5}	1

{3,4}	2
{3,5}	2
{4,5}	1

TABELA 2.5 - Itens Freqüentes
3ª Passagem

L_3	Suporte
{1,2,3}	2
{1,2,4}	4
{1,2,5}	1
{1,3,4}	2
{1,3,5}	1
{1,4,5}	1
{2,3,4}	2
{2,3,5}	1
{2,4,5}	1
{3,4,5}	1

TABELA 2.6 - Itens Freqüentes
4ª Passagem

L_4	Suporte
{1,2,3,4}	2
{1,2,3,5}	1
{1,2,4,5}	1
{1,3,4,5}	1
{2,3,4,5}	1

TABELA 2.7 - Itens Freqüentes
5ª Passagem

L_5	Suporte
{1,2,3,4,5}	1

Com suporte mínimo 1 serão gerados 31 contadores. Se o valor do suporte mínimo for 2 serão gerados 17 contadores. Com valor de suporte mínimo 3 serão gerados 9 contadores. Ou seja, à medida que aumentamos o suporte mínimo diminui o conjunto de contadores gerados conforme demonstrado na figura 2.2.

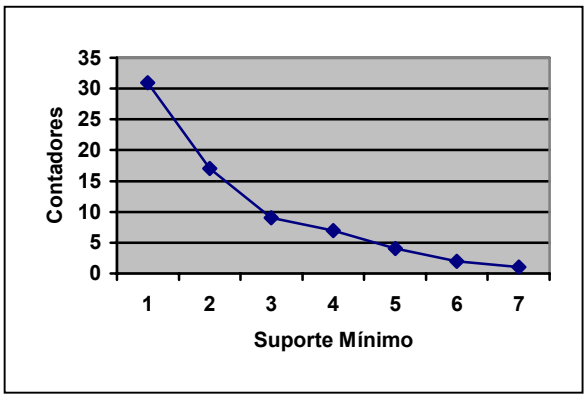


FIGURA 2.2 - Gráfico da relação entre contadores e suporte mínimo

A partir dos conjuntos de itens freqüentes encontrados, devem ser geradas as regras de associação. A tabela 2.8 mostra todas as regras de associação encontradas no banco de dados da tabela 2.2 levando-se em consideração a não aplicação da confiança mínima para a geração da regra.

TABELA 2.8 - Regras de Associação Encontradas

Regra	%Confiança	Regra	%Confiança
1=>2 | 5	71	2=>1 | 5	83
1=>3 | 3	42	3=>1 | 3	75
1=>4 | 4	57	4=>1 | 4	80
1=>5 | 1	14	5=>1 | 1	50
2=>3 | 2	33	3=>2 | 2	50
2=>4 | 5	83	4=>2 | 5	100
2=>5 | 1	16	5=>2 | 1	50
3=>4 | 2	50	4=>3 | 2	40
3=>5 | 2	50	5=>3 | 2	100
4=>5 | 1	20	5=>4 | 1	50
1,2=>3 | 2	40	1,3=>2 | 2	66
2,3=>1 | 2	100	1=>2,3 | 2	28
2=>1,3 | 2	33	3=>1,2 | 2	50
1,2=>4 | 4	80	1,4=>2 | 4	100
2,4=>1 | 4	80	1=>2,4 | 4	57
2=>1,4 | 4	66	4=>1,2 | 4	80
1,2=>5 | 1	20	1,5=>2 | 1	100
2,5=>1 | 1	100	1=>2,5 | 1	14
2=>1,5 | 1	16	5=>1,2 | 1	50
1,3=>4 | 2	66	1,4=>3 | 2	50
3,4=>1 | 2	100	1=>3,4 | 2	28
3=>1,4 | 2	50	4=>1,3 | 2	40
1,3=>5 | 1	33	1,5=>3 | 1	100
3,5=>1 | 1	50	1=>3,5 | 1	14
3=>1,5 | 1	25	5=>1,3 | 1	50
1,4=>5 | 1	25	1,5=>4 | 1	100
4,5=>1 | 1	100	1=>4,5 | 1	14
4=>1,5 | 1	20	5=>1,4 | 1	50
2,3=>4 | 2	100	2,4=>3 | 2	40
3,4=>2 | 2	100	2=>3,4 | 2	33
3=>2,4 | 2	50	4=>2,3 | 2	40
2,3=>5 | 1	50	2,5=>3 | 1	100
3,5=>2 | 1	50	2=>3,5 | 1	16
3=>2,5 | 1	25	5=>2,3 | 1	50
2,4=>5 | 1	20	2,5=>4 | 1	100
4,5=>2 | 1	100	2=>4,5 | 1	16
4=>2,5 | 1	20	5=>2,4 | 1	50

3,4=>5 | 1	50	3,5=>4 | 1	50
4,5=>3 | 1	100	3=>4,5 | 1	25
4=>3,5 | 1	20	5=>3,4 | 1	50
1,2,3=>4 | 2	100	1,2,4=>3 | 2	50
1,3,4=>2 | 2	100	2,3,4=>1 | 2	100
1,2=>3,4 | 2	40	1,3=>2,4 | 2	66
1,4=>2,3 | 2	50	2,3=>1,4 | 2	100
2,4=>1,3 | 2	40	3,4=>1,2 | 2	100
1=>2,3,4 | 2	28	2=>1,3,4 | 2	33
3=>1,2,4 | 2	50	4=>1,2,3 | 2	40
1,2,3=>5 | 1	50	1,2,5=>3 | 1	100
1,3,5=>2 | 1	100	2,3,5=>1 | 1	100
1,2=>3,5 | 1	20	1,3=>2,5 | 1	33
1,5=>2,3 | 1	100	2,3=>1,5 | 1	50
2,5=>1,3 | 1	100	3,5=>1,2 | 1	50
1=>2,3,5 | 1	14	2=>1,3,5 | 1	16
3=>1,2,5 | 1	25	5=>1,2,3 | 1	50
1,2,4=>5 | 1	25	1,2,5=>4 | 1	100
1,4,5=>2 | 1	100	2,4,5=>1 | 1	100
1,2=>4,5 | 1	20	1,4=>2,5 | 1	25
1,5=>2,4 | 1	100	2,4=>1,5 | 1	20
2,5=>1,4 | 1	100	4,5=>1,2 | 1	100
1=>2,4,5 | 1	14	2=>1,4,5 | 1	16
4=>1,2,5 | 1	20	5=>1,2,4 | 1	50
1,3,4=>5 | 1	50	1,3,5=>4 | 1	100
1,4,5=>3 | 1	100	3,4,5=>1 |1	100
1,3=>4,5 | 1	33	1,4=>3,5 | 1	25
1,5=>3,4 | 1	100	3,4=>1,5 | 1	50
3,5=>1,4 | 1	50	4,5=>1,3 | 1	100
1=>3,4,5 | 1	14	3=>1,4,5 | 1	25
4=>1,3,5 | 1	20	5=>1,3,4 | 1	50
2,3,4=>5 | 1	50	2,3,5=>4 | 1	100
2,4,5=>3 | 1	100	3,4,5=>2 | 1	100
2,3=>4,5 | 1	50	2,4=>3,5 | 1	20
2,5=>3,4 | 1	100	3,4=>2,5 | 1	50
3,5=>2,4 | 1	50	4,5=>2,3 | 1	100
2=>3,4,5 | 1	16	3=>2,4,5 | 1	25
4=>2,3,5 | 1	20	5=>2,3,4 | 1	50
1,2,3,4=>5 | 1	50	1,2,3,5=>4 | 1	100
1,2,4,5=>3 | 1	100	1,3,4,5=>2 | 1	100
2,3,4,5=>1 | 1	100	1,2,3=>4,5 | 1	50
1,2,4=>3,5 | 1	25	1,2,5=>3,4 | 1	100
1,3,4=>2,5 | 1	50	1,3,5=>2,4 | 1	100
1,4,5=>2,3 | 1	100	2,3,4=>1,5 | 1	50
2,3,5=>1,4 | 1	100	2,4,5=>1,3 | 1	100
3,4,5=>1,2 | 1	100	1,2=>3,4,5 | 1	20

1,3=>2,4,5 | 1	33	1,4=>2,3,5 | 1	25
1,5=>2,3,4 | 1	100	2,3=>1,4,5 | 1	50
2,4=>1,3,5 | 1	20	2,5=>1,3,4 | 1	100
3,4=>1,2,5 | 1	50	3,5=>1,2,4 | 1	50
4,5=>1,2,3 | 1	100	1=>2,3,4,5 | 1	14
2=>1,3,4,5 | 1	16	3=>1,2,4,5 | 1	25
4=>1,2,3,5 | 1	20	5=>1,2,3,4 | 1	50

Supondo-se que não foi definido o parâmetro de confiança mínima serão geradas 70 regras. Com confiança mínima de 50% serão geradas 115 regras. Com confiança mínima de 75% serão geradas 63 regras. A medida que aumenta a confiança mínima diminui o número de regras geradas conforme a figura 2.3.

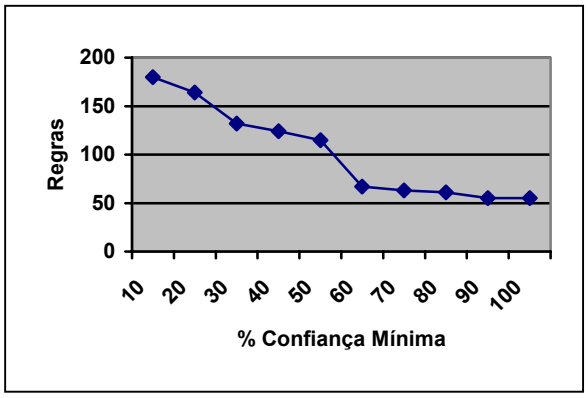


FIGURA 2.3 - Gráfico da relação entre regras e confiança mínima.

3 Algoritmos de Descoberta de Regras de Associação

[AGR93] introduziu o problema de mineração de regras de associação, desde então foram apresentados diversos estudos principalmente com o objetivo de otimizar a idéia inicial. Neste capítulo será apresentada uma revisão bibliográfica dos principais algoritmos propostos na literatura para o problema de descoberta de regras de associação.

3.1 Algoritmo *Apriori*

O algoritmo *Apriori* foi proposto por [AGR93] e é o algoritmo mais utilizado para descobrir regras de associação. Para isto, o algoritmo *Apriori* executa múltiplas passagens sobre os banco de dados de transações. Na primeira passagem é contado o suporte de cada item. Aqueles itens que tem o suporte individual maior que o suporte mínimo são considerados itens freqüentes. Em cada uma das passagens subsequentes, tendo que k é o número da passagem, os itens freqüentes da passagem imediatamente anterior (k - 1) são agrupados em conjuntos de k itens e sendo estes conjuntos considerados como itens candidatos, sendo então executada a contagem do suporte dos itens candidatos. Se este suporte for maior que o suporte mínimo estes conjuntos de k itens candidatos são considerados freqüentes. O processo continuará iteragindo até que o conjunto de itens freqüentes seja um conjunto vazio. Este algoritmo gera um conjunto muito menor de itens candidatos do que o algoritmo original *AIS* proposto por [AGR93].

Como método de otimização de tempo de execução, este algoritmo parte do princípio de que se $X \subset Y$, e X não é freqüente, logo Y também não é freqüente. Deste modo, a cada nova passagem sobre o banco de dados, o algoritmo não necessita ler novamente todo o banco de dados, mas somente o conjunto de itens candidatos selecionados na passagem anterior, o que implica a geração e contagem de poucos itens candidatos, o que resulta em uma diminuição do tempo de execução e na geração de um conjunto pequeno de regras.

Na primeira passagem do algoritmo *Apriori* são contadas as ocorrências de cada item para determinar o conjunto de itens freqüentes. A passagem seguinte, denominada passagem k, consiste de duas fases: na primeira fase o conjunto de itens freqüentes L_{k-1} encontrado na passagem (k-1) é usado para gerar o conjunto de itens candidatos usando a função *Apriori* - *Gen*. Na segunda fase o banco de dados é pesquisado e o suporte dos conjuntos de itens candidatos em C_k é contado.

A figura 3.1 mostra o pseudocódigo do algoritmo *Apriori*.

```
L1 = {large 1-itemsets};
for (k=2; Lk-1 ≠ ∅; k++) do begin
   Ck = apriori_gen(Lk-1);  // Novos candidatos
   forall transactions t ∈ D do begin
      Ct = subset(Ck, t);  // Candidatos contidos em t
      forall candidates c ∈ Ct do
         c.count++;
   end
   Lk = {c ∈ Ck | c.count ≥ minsup};
end
Answer = ∪k Lk;
```

FIGURA 3.1 - Algoritmo Apriori [AGR93]

3.1.1 Função *Apriori - gen*

A função *Apriori - Gen* requer como argumento o L_{k-1} que consiste no conjunto de todos os conjuntos de itens freqüentes com (*k-1*) itens e retorna o conjunto de todos os conjuntos de itens candidatos com *k* itens. Esta função é constituída de dois passos, o primeiro de união e o segundo de poda. A figura 3.2 mostra o pseudocódigo da função *Apriori - Gen*.

```
insert into Ck
   select p.item1, p.item2, ..., p.itemk-1, q.itemk-1
   from Lk-1 p, Lk-1 q
   where p.item1 = q.item1 and p.item2 = q.item2 and ...
      and p.itemk-2 = q.itemk-2 and p.itemk-1 < q.itemk-1;

forall itemsets c ∈ Ck do
   forall (k-1)- subsets s of c do
      if (s ∉ Lk-1) then
         delete c from Ck;
```

FIGURA 3.2 - Função Apriori - Gen [AGR93]

3.1.2 A função *Subset*

A função *subset* é executada com a finalidade de identificar que conjuntos de itens em uma transação são candidatos para posterior contagem destes conjuntos de itens. Esta função recebe como argumentos o conjunto de itens candidatos para a passagem atual C_k e a transação atual t. Cada conjunto de itens c contido na transação que também está contido no conjunto de itens candidatos C_k é incluído no conjunto de itens candidatos da transação C_t.

Quando a função *subset* é executada, o conjunto de itens candidatos está armazenado em uma *hash-tree*, tal estrutura de dados é explicada no item 3.1.3. Um nó em uma *hash-tree* ou contém uma lista de conjuntos de itens (nó folha), caso esteja em no nível k+1; ou contém uma tabela *hash* (nó interno), caso esteja em um nível menor ou igual a k, onde cada divisão desta tabela aponta para outro nó em um nível imediatamente inferior da árvore. Define-se a raiz da árvore como tendo a profundidade 1. Quando está sendo acrescentado um novo conjunto de itens, parte-se do nodo raiz para baixo até encontrar-se uma folha. Quando um conjunto de itens c é adicionado, inicia-se da raiz da árvore até alcançar-se uma folha; a definição do caminho a ser seguido é dada pela função *hash*. Cada nó é inicialmente criado como uma folha, e quando o número de itens ultrapassar um certo limite o nó é convertido para um nó interno.

A função *subset*, a partir do nó raiz, procura todos os candidatos contidos em uma transação t da seguinte maneira: se o nó atual é uma folha, verifica-se que conjuntos de itens na folha estão contidos em t e acrescenta-se referências a estes itens no conjunto resposta. Se o nó atual é um nó interno e foi alcançado através da aplicação da função *hash* em um item i, divide-se em cada item procedente i em t e recursivamente aplica-se este procedimento no nó da divisão correspondente. Os conjuntos de itens contidos em C_t que também estão contidos em C_k têm seu contados de suporte incrementado em uma unidade.

Supondo-se que se tem a transação 7 do banco de dados da tabela 2.2, t_7={1,2,3,4}, e está sendo executada a terceira passagem sobre o banco de dados, o conjunto de itens candidatos seria o descrito na figura 3.5. Já que está sendo executada a terceira passagem os itens da transação 7 seriam agrupados em conjuntos de três sendo gerados os conjuntos C_t={{1,2,3}, {1,2,4}, {1,3,4}, {2,3,4}}. Como todos os conjuntos de itens estão contidos no conjunto de itens candidatos da passagem, todos eles têm seu suporte incrementado em uma unidade. Este processo é feito para cada transação do banco de dados.

As figuras 3.3 a 3.4 mostram as árvores hash com os conjuntos de itens candidatos em cada uma das passagens. As tabelas 3.1 a 3.3 mostram os conjuntos de itens freqüentes selecionados em cada uma das cinco passagens do algoritmo *Apriori* sobre o banco de dados definido na tabela 2.2 , sendo que o suporte mínimo é 3.

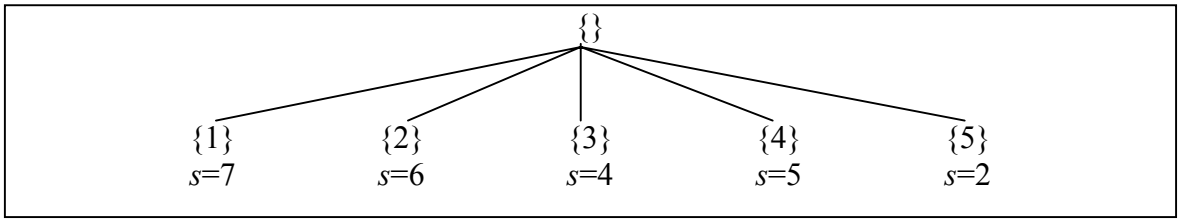


FIGURA 3.3 – Hash Tree com Conjuntos de Itens Candidatos 1ª Passagem

TABELA 3.1 - Conjuntos de Itens Freqüentes 1ª Passagem

L_1	Suporte
{1}	7
{2}	6
{3}	4
{4}	5

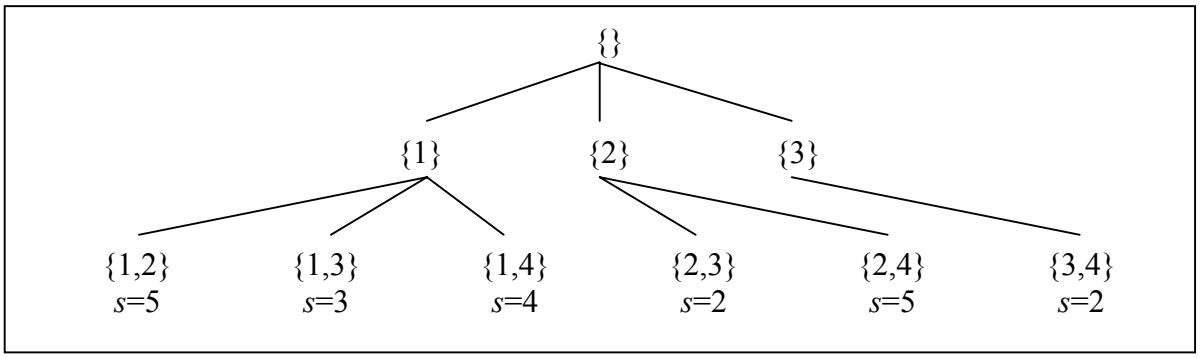


FIGURA 3.4 – Hash Tree com Conjuntos de Itens Candidatos 2ª Passagem

TABELA 3.2 - Conjuntos de Itens Freqüentes 2ª Passagem

L_2	Suporte
{1,2}	5
{1,3}	3
{1,4}	4
{2,4}	5

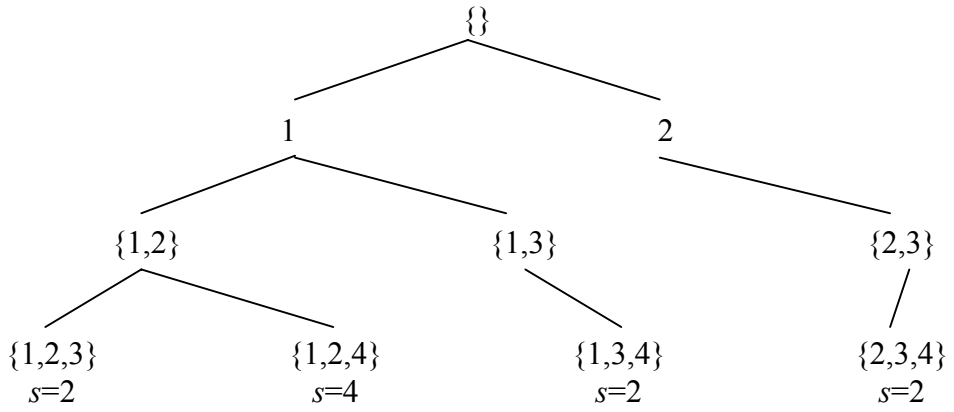


FIGURA 3.5 – Hash Tree com Conjuntos de Itens Candidatos 3ª Passagem

TABELA 3.3 - Conjuntos de Itens Freqüentes 3ª Passagem

L_3	Suporte
{1,2,4}	4

A tabela 3.4 apresenta as regras de associação encontradas no banco de dados de teste sem considerar a restrição de confiança mínima.

TABELA 3.4 - Regras de Associação Encontradas

Regra	%Confiança	Regra	%Confiança
1=>2 | 5	71	2=>1 | 5	83
1=>3 | 3	42	3=>1 | 3	75
1=>4 | 4	57	4=>1 | 4	80
2=>4 | 5	83	4=>2 | 5	100
1,2=>4 | 4	80	1,4=>2 | 4	100
2,4=>1 | 4	80	1=>2,4 | 4	57
2=>1,4 | 4	66	4=>1,2 | 4	80

3.1.3 *Hash Tree*

Muitas tarefas de mineração de dados, tais como: regras de associação e padrões seqüenciais, usam estruturas de dados complexas baseadas em ponteiros. Dentre estas estruturas a *hash tree* é a mais utilizada [PAR98]. Uma *hash tree* é uma estrutura híbrida com características de estruturas de *tree* e *hash table*.

Um nodo interno de uma *hash tree* em uma profundidade *d* contém uma *hash table* na qual células apontam para nodos em uma profundidade *d*+1. Todos os conjuntos de itens (*itemsets*) são armazenados nas folhas e ordenados em uma lista. A profundidade máxima de uma árvore em uma iteração *k* é *k*. Os diferentes componentes de uma *hash tree* são os seguintes: (HTN) *hash tree node*, (HTNP) *hash table*, (ILN) *Itemset list header*, (LN) *list node* e os *itemsets* (ISET). Um nodo interno em uma *hash table* aponta para

nodos no próximo nível, e uma lista de conjuntos vazios, enquanto um nodo tem uma lista de conjuntos de itens. Para contar o suporte dos conjuntos de itens candidatos em uma passagem *k*, para cada transação *T* no banco de dados, são formados todos os conjuntos de itens com *k* elementos de *T* em uma ordem lexicográfica. Para cada subconjunto a hash tree é vasculhada em busca do conjunto de itens candidatos e seu contador é atualizado.

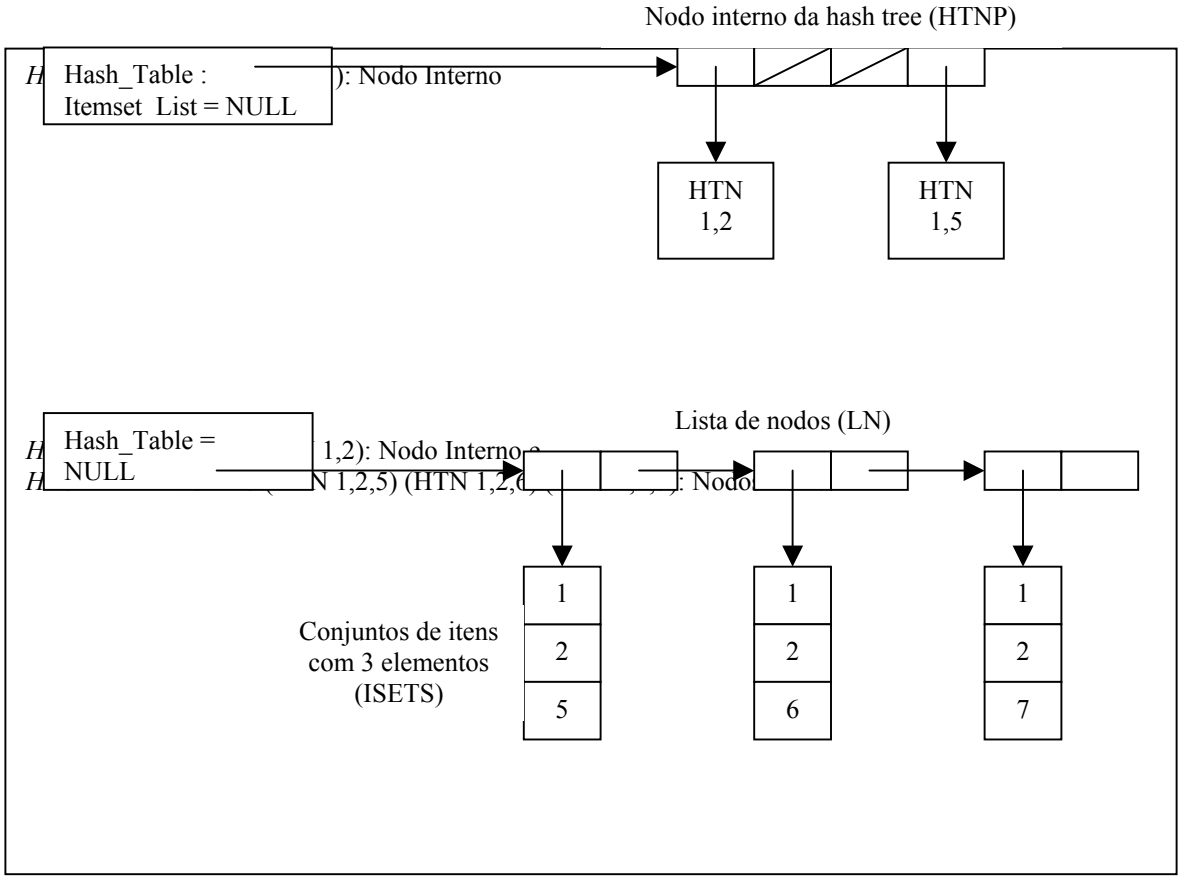


FIGURA 3.6 – Exemplo de uma hash tree

A figura apresenta uma *hash tree* criada na terceira passagem do algoritmo *Apriori* sobre um banco de dados qualquer. Supondo-se que os conjuntos de itens candidatos seriam {1,2,5}, {1,2,6} e {1,2,7} a *hash tree* teria a forma apresentada na figura 3.6. A primeira parte da figura 3.6 mostra o HTN 1, ou seja, o nodo interno da árvore com o item 1 no primeiro nível de profundidade. A segunda parte da figura mostra os HTN 1,2 e HTN 1,5 no segundo nível da árvore. Também são mostrados os nodos folha: ISET 1,2,5; ISET 1,2,6 e ISET 1,2,7.

3.2 Algoritmo *AprioriTid*

Este algoritmo, proposto por [AGR94] também utiliza a função *Apriori - Gen*, descrita na figura 3.2, para determinar os conjuntos de itens candidatos antes da passagem começar. Seu diferencial em relação ao algoritmo *Apriori* consiste no fato de que o banco de dados não é utilizado para contagem do suporte após a primeira passagem, pois um conjunto $\underline{C_k}$ é usado para este propósito.

Um conjunto $\underline{C_k}$ consiste em um conjunto dos itens candidatos na passagem k associados com o seu respectivo identificador de transação (*TID*). Cada membro do conjunto $\underline{C_k}$ está na forma $< TID, \{ X_k \} >$, onde TID é o identificador da transação e cada X_k é um conjunto de dados candidato presente na transação com o identificador *TID*. O membro de $\underline{C_k}$ correspondente a transação t é <t.TID, $\{c \in C_k \mid c$ está contido em $t\}$>.

3.2.1 Estrutura de dados

Cada conjunto de itens candidato tem um número único chamado de seu *ID*. Cada conjunto de itens candidato em C_k é mantido em um *array* indexado pelo *ID* do conjunto de itens candidato em C_k. Um membro de $\underline{C_k}$ está agora na forma <*TID*, {*ID*}>. Cada conjunto $\underline{C_k}$ está armazenado em uma estrutura seqüencial.

A função *Apriori - gen* gera um conjunto de itens candidatos c_k com k itens pela união de conjunto de dados freqüentes com (k-1) itens. São mantidos dois campos adicionais para cada conjunto de itens candidatos: geradores e extensões. O primeiro campo de conjunto de itens candidato c_k armazena os *ID*s dos dois conjuntos de itens freqüentes com (*k-1*) itens que originaram o conjunto de itens com k item gerado. O campo extensões de um conjunto de itens C_k armazena os *ID*s de todos os conjuntos de itens candidatos com (*k+1*) itens que são extensão do conjunto de itens c_k.

Quando houver a necessidade de contar o suporte de um conjunto de itens candidato, é então lido o conjunto $\underline{C_k}$, e não todo o conjunto de transações como ocorria com o algoritmo *Apriori*, o que significa a leitura de um conjunto menor de elementos, otimizando o tempo de execução do algoritmo *AprioriTid*.

A figura 3.7 mostra o pseudocódigo do algoritmo *AprioriTid*.

```
L1 = {large 1-itemsets};
```

```
C̲1 = database D;
for (k=2; Lk-1 ≠ ∅; k++) do begin
   Ck = apriori_gen(Lk-1);  // Novos candidatos
   C̲k = 0;
   forall entries t ∈ C̲k-1 do begin
   // determinar conjuntos de itens candidatos em Ck
   // contidos na transação com identificador t.TID
      Ct = {c ∈ Ck | (c - c[k] ) ∈ t.set-of-itemsets ∨
         (c - c[k-1]) ∈ t.set-of-itemsets};
      forall candidates c ∈ Ct do
         c.count++;
      if (Ct ≠ 0) then C̲k += <t.TID,Ct>;
   end
   Lk = {c ∈ Ck | c.count ≥ minsup};
end
Answer = ∪k Lk;
```

FIGURA 3.7 - Algoritmo AprioriTid [AGR93]

Considerando-se o banco de dados da tabela 2.2 e assumindo-se o suporte mínimo 3, o resultado obtido pela aplicação do algoritmo *AprioriTid* são apresentados nas tabelas 3.5 a 3.13:

TABELA 3.5 - Conjuntos de Itens $\underline{C_1}$

TID	$\underline{C_1}$
1	{ {1} {3} }
2	{ {3} {5} }
3	{ {1} {2} {4} }
4	{ {2} {4} }
5	{ {1} {2} {3} {4}{5} }
6	{ {1} }
7	{ {1} {2} {3} {4}}
8	{ {1} {2} {4} }
9	{ {1} {2} }

TABELA 3.6 - Conjuntos de Itens Freqüentes 1ª Passagem

L_1	Suporte
{1}	7
{2}	6
{3}	4
{4}	5
{5}	2

TABELA 3.7 - Conjuntos de Itens Candidatos 2ª Passagem

C_2	Suporte

{1,2}	5
{1,3}	3
{1,4}	4
{2,3}	2
{2,4}	5
{3,4}	2

TABELA 3.8 - Conjuntos de Itens $\underline{C}_2$

TID	$\underline{C}_2$
1	{ {1,3} }
2	
3	{ {1,2} {1,4} {2,4} }
4	{ {2,4} }
5	{ {1,2} {1,3} {1,4} {2,3} {2,4} {3,4} }
6	
7	{ {1,2} {1,3} {1,4} {2,3} {2,4} {3,4} }
8	{ {1,2} {1,4} {2,4} }
9	{ {1,2} }

TABELA 3.9 - Conjuntos de Itens Freqüentes 2ª Passagem

L_2	Suporte
{1,2}	5
{1,3}	3
{1,4}	4
{2,4}	5

TABELA 3.10 - Conjunto de Itens Candidatos 3ª Passagem

C_3	Suporte
{1,2,3}	2
{1,2,4}	4
{1,3,4}	2
{2,3,4}	2

TABELA 3.11 - Conjunto de Itens $\underline{C}_3$

TID	$\underline{C}_3$
1	

2	
3	{ {1,2,4} }
4	
5	{ {1,2,4} }
6	
7	{ {1,2,4} }
8	{ {1,2,4} }
9	

TABELA 3.12 - Conjuntos de Itens Candidatos 4ª Passagem

C_4	Suporte

TABELA 3.13 - Regras de Associação Encontradas

Regra	%Confiança	Regra	%Confiança
1=>2 | 5	71	2=>1 | 5	83
1=>3 | 3	42	3=>1 | 3	75
1=>4 | 4	57	4=>1 | 4	80
2=>4 | 5	83	4=>2 | 5	100
1,2=>4 | 4	80	1,4=>2 | 4	100
2,4=>1 | 4	80	1=>2,4 | 4	57
2=>1,4 | 4	66	4=>1,2 | 4	80

3.3 Algoritmo *AprioriHybrid* [AGR94]

Durante a execução de um processo de mineração de dados não é necessária a execução de um mesmo algoritmo durante todas as passagens sobre o banco de dados. Conforme [AGR94], é constatado que o algoritmo *Apriori* executa mais rapidamente o processo de mineração nas passagens iniciais que o algoritmo *AprioriTid*, enquanto, após um certo número de passagens, o *AprioriTid* passa a ser mais rápido que *Apriori*. A razão para este comportamento consiste no fato de que o número de conjuntos de itens candidatos reduz gradativamente após cada passagem do algoritmo sobre o banco de dados, entretanto o algoritmo *Apriori* ainda examina todas as transações no banco de dados, enquanto o algoritmo *AprioriTid* pesquisa $\underline{C_k}$ para obter a contagem de suporte, e o tamanho de $\underline{C_k}$ tornou-se menor que o tamanho do banco de dados. Quando o conjunto $\underline{C_k}$ pode ser alocado em memória, não necessita-se mais do custo de escrita deste no disco.

Baseado neste fato, é proposto o algoritmo *AprioriHybrid* que combina a utilização de *Apriori* nas passagens iniciais e *AprioriTid* nas passagens subsequentes quando o conjunto $\underline{C_k}$ puder ser alocado em memória. A mudança de *Apriori* para *AprioriTid* envolve um custo: assumindo-se que seja decidida a mudança no fim da passagem k, na

passagem (*k+1*), após a pesquisa dos conjuntos de itens candidatos contidos em uma transação, deve-se acrescentar o *ID* da transação ao conjunto C_{k+1}, e então, a partir da passagem (*k+2*) é possível mudar para *AprioriTid*.

A vantagem de *AprioriHybrid* sobre *Apriori* depende de como o tamanho do conjunto $\underline{C_k}$ diminui a cada passagem sobre o banco de dados [AGR96]. Se o declínio de tamanho do conjunto $\underline{C_k}$ for gradual, é obtida uma diminuição significativa no tempo de execução do algoritmo, mas se o declínio de tamanho do conjunto $\underline{C_k}$ for abrupto, então *AprioriHybrid* pode fornecer uma diminuição do tempo de execução muito pequena ou até aumento do tempo de execução, caso a mudança ocorrer muito perto da última passagem do algoritmo.

3.4 Algoritmo *Dense - Miner*

Em [BAY99] é apresentado o algoritmo *Dense - Miner*, o qual aplica todos os limites definidos pelo usuário durante a execução do algoritmo de mineração a fim de melhorar a eficiência em dados densos, tais como dados relacionais. Visto que as regras satisfazendo os limites de suporte e confiança em bancos de dados densos, obtidas através da aplicação de algoritmos convencionais, são freqüentemente muito numerosas para serem mineradas eficientemente ou compreendidas pelo usuário final, o algoritmo *dense – miner* explora outros limites que eliminam regras que não são interessantes porque elas contém condições que não contribuem decisivamente para a habilidade preditiva da regra, estes limites considerados seriam a melhoria e o limite de conseqüente.

Tem-se como exemplo a regra:

Bermuda & tênis => meia (confiança 60%)

Esta regra determina que 60% das pessoas que compram bermuda e tênis também compram meia. Esta regra, apesar de possuir uma confiança elevada, pode ser interessante ou não. Se 61% das pessoas em estudo compram meia, a regra não é interessante, visto que ela caracteriza uma população que é menos comum de compradores de meia. Este fato motiva a utilização de medidas adicionais que representem uma vantagem preditiva de uma regra em relação a outra.

Ainda referente a regra acima, suponhamos que a venda de tênis implique na venda de meia em 95% dos casos, então haveria um decremento da confiança da regra sobre uma regra mais concisa que é mais largamente aplicável.

No caso do algoritmo *dense - miner*, há dois modos propostos para se resolver o problema de dados densos, são eles: a limitação de conseqüente e a limitação de melhora mínima.

3.4.1 Limitação de conseqüente

A limitação de conseqüente consiste no fato de que uma regra minerada seja considerada somente se ela tiver um dado conjunto de itens conseqüente *C* definido pelo usuário. Este conjunto de itens *C* deve ser composto por no mínimo 1 elemento. Dentre as finalidades desta limitação está reduzir o conjunto de itens freqüentes considerados.

3.4.2 Limitação de melhora mínima

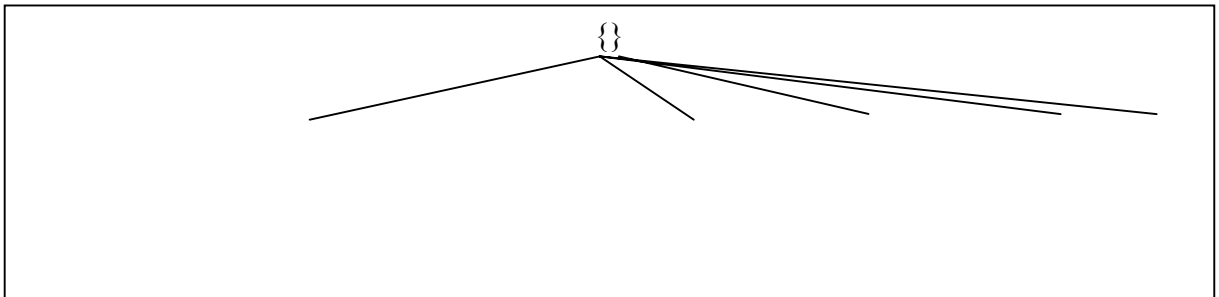
É permitido ao usuário especificar um limite de melhora mínima, denominado de *minimp*, tal que uma regra tenha uma confiança com ao menos *minimp* maior que a confiança de qualquer uma das suas sub - regras, onde uma sub - regra é uma simplificação da regra formada pela remoção de um ou mais condições de seu antecedente. À medida que o usuário aumenta o valor de *minimp* torna-se mais conciso o conjunto de regras geradas, sendo tal conjunto formado somente por itens que são decisivamente importantes para a habilidade preditiva da regra.

Um algoritmo que roda eficientemente em bancos de dados densos, sem restrições adicionais, pode fornecer um resultado final com um conjunto extremamente grande de regras, sem indicação de quais destas regras são boas. Tem-se a melhora de uma regra como sendo a diferença mínima entre sua confiança e a confiança de qualquer de suas sub–regras com o mesmo conseqüente.

Uma regra com melhoria negativa é geralmente indesejável porque a regra pode ser simplificada para produzir uma sub–regra que é mais preditiva, e se aplica a uma população igual ou maior. Uma regra com melhoria igual a 0 é, na maioria dos casos, desejável. Uma regra com melhoria positiva é completamente desejável, porque a maioria das regras em bancos de dados densos não são úteis apresentando um decremento de confiança.

3.4.3 Pesquisa por enumeração de conjunto em conjuntos de dados freqüentes

Suponhamos que uma regra será representada usando apenas seu conjunto de itens antecedente, pois assume-se que o conseqüente é um conjunto de itens *C*. Temos que *U* é o conjunto de todos os itens presentes no banco de dados exceto aqueles presentes no conseqüente da regra. O problema de mineração é a força do conjunto de *U* regras que satisfaçam o limite de suporte , confiança e melhoramento mínimos. Este algoritmo baseia-se na estrutura de árvore de enumeração de conjunto de *Rymon*, que é definida em Raymon apud [BAY99], para reduzir o espaço de regras consideradas. A idéia é primeiro impor uma ordem ao conjunto de itens, e então enumerar conjuntos de itens de acordo com a ordenação. A figura 3.8 mostra uma árvore completamente expandida sobre U = {1,2,3,4,5}, com itens ordenados lexicamente.



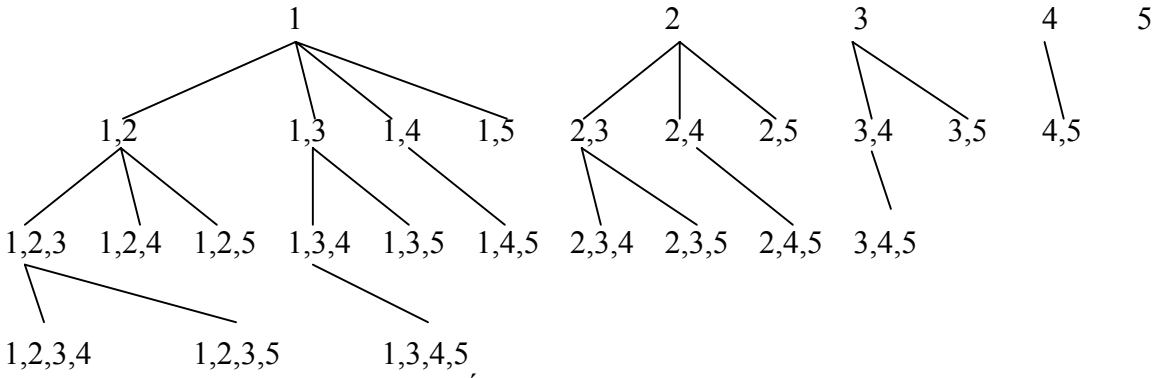


FIGURA 3.8 - Árvore de enumeração de conjunto.

Cada nó na árvore é chamado de grupo, sendo representado por dois conjuntos de itens. O primeiro conjunto de itens, chamado de cabeça, é simplesmente uma enumeração do conjunto de itens de um dado nó. O segundo conjunto de itens, chamados de corpo, é um conjunto ordenado e consiste daqueles itens que podem ser potencialmente acrescentados a cabeça para formar qualquer regra viável enumerada por um sub–nó.

A cabeça e o corpo de um grupo g são denotadas como $h(g)$ e $t(g)$. Cada filho g_c de um grupo g é formado por um item $i \in t(g)$ e acrescido de $h(g)$ para forma $h(g_c)$. Então, $t(g_c)$ é feito para conter todos os itens em $t(g)$ que seguem i em ordem.

Uma regra é dita derivável de um grupo g se $h(g) \subset r$, e $r \subseteq h(g) \cup t(g)$. Define-se um conjunto de itens candidatos de um grupo g sendo formado pelos seguintes conjuntos de itens:

- $h(g)$e $t(g) \cup C$;
- $h(g) \cup \{i\}$ e $h(g) \cup \{i\} \cup C$ para todo $i \in t(g)$;
- $h(g) \cup t(g)$ e $h(g) \cup t(g) \cup C$.

3.4.4 Descrição do algoritmo

O corpo do algoritmo (Figura 3.10) implementa uma ampla pesquisa transversal da árvore de enumeração, a fim de limitar o número de passagens sobre o banco de dados, e tendo o procedimento *Generate - Initial - Groups* conduzindo a pesquisa. Os grupos representando um nível inteiro da árvore são processados juntos em uma passagem sobre os dados.

A implementação considera a pesquisa no segundo nível da árvore, após a fase de otimização, onde é calculado o suporte de todas as regras de 1 e 2 itens e seus antecedentes, usando estruturas de dados em *array*, ao invés de estruturas em árvores *hash*. Na primeira passagem *Generate - Initial - Groups* pode simplesmente produzir o nó raiz que consiste de uma cabeça vazia e tendo *U* como seu corpo. Os parâmetros de entrada *minconf*, *minsup*, *minimp* e *C* são assumidos como globais.

A figura 3.9 mostra o pseudocódigo do nível superior do algoritmo *Dense - Miner*.

```
Procedure Dense - Miner
DENSE – MINER (Set of Transactions T)
  // Retorna todas regras freqüentes, confidentes,
  // freqüentes e melhoradas presentes em T
  Set of Rules R ← ∅
  Set of Groups G ← GENERATE-INITIAL-GROUPS(T,R)
  While G is non-empty do
     Scan T to process all groups in G
     PRUNE-GROUPS (G,R)
     G ← GENERATE-NEXT-LEVEL (G)
     R ← R ∪ EXTRACT-RULES (G)
     PRUNE-GROUPS (G,R)
  Return POST-PROCESS (R,T)
end
```

FIGURA 3.9 - Nível superior do algoritmo *Dense – Miner*.

Generate - Next - Level gera os grupos que compreendem o próximo nível da árvore de enumeração de conjunto. Após a expansão dos filhos, qualquer regra representada pela cabeça do grupo é colocada em *R* pela *Extract - Rules* se ela tem suporte e confiança mínimas. A informação de suporte requerida para determinação de suporte e confiança é fornecida pelos pais de g na árvore de enumeração de conjunto porque *h(g)* e *h(g)* ∪ *C* são membros de seu conjunto candidato. A figura 3.10 mostra o pseudocódigo do procedimento *Generate - Next - Level*.

```
Procedure Generate - next - level
GENERATE-NEXT-LEVEL (Set of groups G)
  // Retorna um conjunto de grupos representando o próximo
  // nível da árvore de enumeração de conjunto
  Set of Groups Gc ← ∅
  For each group g in G do
```

```
      Reorder the items in t(g)
      For each item i in t(g) do
         Let g_c be a new group
            With h(g_c)= h(g) ∪ {i} and
            t(g_c) = {j|j follows i in the ordering}
         G_c ← G_c ∪ {g_c}
   Return G_c
end
```

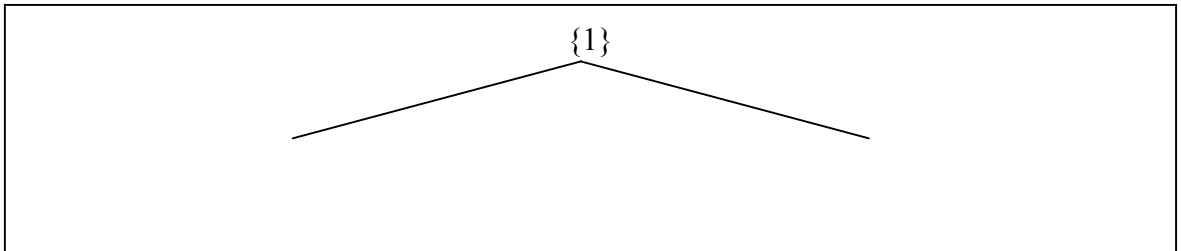
FIGURA 3.10 – Procedimento para expansão do próximo nível da árvore de enumeração de conjunto.

Os grupos são podados pelo procedimento *Prune - Groups* (Figura 3.11) seguindo a expansão da árvore no momento em que esta expansão ocorre. A decisão sobre poda ou não do grupo é feita baseada em informação de suporte reunida durante as passagens anteriores sobre o banco de dados.

```
Procedure Prune – Groups
PRUNE-GROUPS(Set of groups G, Set of rules R)
  // Poda grupos e marca itens dos grupos de G
  // G e R são passados por referência
  For each group g in G do
     do
        try_again ← false
        if IS-PRUNABLE(g)
           then remove g from G
           else for each i ∈ t(g) do
              let g' be a group
                 with h(g') = h(g) ∪ {i}
                 and t(g') = t(g) – {i}
           if IS-PRUNABLE(g')
              then remove i from t(g)
                 put h(g) ∪ {i} in R if
                  it is a frequent and confident rule
                 try_again
     while try_again = true
end
```

FIGURA 3.11 - Nível superior da função de poda.

A seguir são aplicados os conceitos utilizados pelo algoritmo *Dense - Miner* sobre o conjunto de dados apresentado na tabela 2.1, supondo que os limites fornecidos pelo usuário são suporte mínimo é 2, a confiança mínima é 50%, limite de item conseqüente é 1, e a melhora mínima 5%. A figura 3.12 apresenta a árvore de enumeração de conjunto gerada.



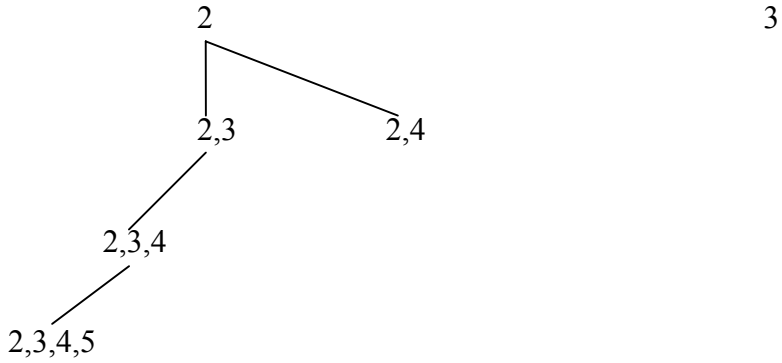


FIGURA 3.12 - Árvore de enumeração de conjunto.

A Figura 3.12 mostra uma árvore de enumeração de conjunto completamente expandida com itens ordenados lexicamente sobre o conjunto definido na tabela 2.1. As tabelas 3.14 a 3.19 mostram o resultado das sucessivas passagens do algoritmo sobre o banco de dados. A tabela 3.18 demonstra somente as regras com cardinalidade 2 (com 2 items), estas regras são apenas um subconjunto de todas as regras geradas antes do processo de poda. O conjunto completo das regras pode ser encontrado na tabela 1 do Anexo I.

TABELA 3.14 - Resultado de Generate - Initial - Groups

Grupo	Suporte
{2}	5
{3}	3
{4}	4
{5}	1

TABELA 3.15 - Generate - Next - Level 1ª Passagem

Grupo	Suporte
{2,3}	2
{2,4}	4
{2,5}	1
{3,4}	2
{3,5}	1
{4,5}	1

TABELA 3.16 - Generate - Next - Level 2ª Passagem

Grupo	Suporte
{2,3,4}	2
{2,3,5}	1
{3,4,5}	1

TABELA 3.17 - Generate - Next - Level
3ª Passagem

Grupo	Suporte
{2,3,4,5}	1

TABELA 3.18 - Regras de associação encontradas antes do processo de poda

Regra	% Confiança	Melhora
1=>2 | 5	71	-6
2=>1 | 5	83	17
1=>3 | 3	42	-35
3=>1 | 3	75	31
1=>4 | 4	57	-20
4=>1 | 4	80	25
1=>5 | 1	14	-63
5=>1 | 1	50	28
2=>3 | 2	33	-33
3=>2 | 2	50	6
2=>4 | 5	83	17
4=>2 | 5	100	45
2=>5 | 1	16	-50
5=>2 | 1	50	28
3=>4 | 2	50	6
4=>3 | 2	40	-15
3=>5 | 2	50	6
5=>3 | 2	100	78
4=>5 | 1	20	-35
5=>4 | 1	50	28

TABELA 3.19 - Regras de associação encontradas após o processo de poda

Regra	% Confiança	Melhora
2=>1 | 5	83	17
3=>1 | 3	75	31
4=>1 | 4	80	25

5=>1 | 1	50	28
3=>2 | 2	50	6
2=>4 | 5	83	17
4=>2 | 5	100	45
5=>2 | 1	50	28
3=>4 | 2	50	6
3=>5 | 2	50	6
5=>3 | 2	100	78
5=>4 | 1	50	28
2,3=>1 | 2	100	17
1=>2,3 | 2	28	6
1,2=>4 | 4	80	38
1=>2,4 | 4	57	2
2=>1,4 | 4	66	22
4=>1,2 | 4	80	25
1,5=>2 | 1	100	29
2,5=>1 | 1	100	27
1=>2,5 | 1	14	3
2=>1,5 | 1	16	5
1,3=>4 | 2	66	9
1,4=>3 | 2	50	8
3,4=>1 | 2	100	20
1=>3,4 | 2	28	6
3=>1,4 | 2	50	6
4=>1,3 | 2	40	7
3=>1,5 | 1	25	14
5=>1,3 | 1	50	17
1,4=>5 | 1	25	5
1,5=>4 | 1	100	43
4,5=>1 | 1	100	20
1=>4,5 | 1	14	3
4=>1,5 | 1	20	9
5=>1,4 | 1	50	6
2,3=>4 | 2	100	17
2=>3,4 | 2	33	11
4=>2,3 | 2	40	18
3=>2,5 | 1	25	14
5=>2,3 | 1	50	28
2,5=>4 | 1	100	17
2=>4,5 | 1	16	5
4=>2,5 | 1	20	9
3=>4,5 | 1	25	14
5=>3,4 | 1	50	28
1,2=>3,4 | 2	40	7
1,3=>2,4 | 2	66	9
1,4=>2,3 | 2	50	10

2,3=>1,4 | 2	100	34
3,4=>1,2 | 2	100	20
3=>1,2,4 | 2	50	6
1,3=>2,5 | 1	33	8
1,5=>2,3 | 1	100	50
2,3=>1,5 | 1	50	25
2,5=>1,3 | 1	100	50
1=>2,3,5 | 1	14	3
2=>1,3,5 | 1	16	5
3=>1,2,5 | 1	25	14
5=>1,2,3 | 1	50	28
1,2=>4,5 | 1	20	4
1,4=>2,5 | 1	25	5
1,5=>2,4 | 1	100	43
2,5=>1,4 | 1	100	34
4,5=>1,2 | 1	100	20
1=>2,4,5 | 1	14	3
2=>1,4,5 | 1	16	5
4=>1,2,5 | 1	20	9
5=>1,2,4 | 1	50	6
1,3=>4,5 | 1	33	8
1,4=>3,5 | 1	25	3
1,5=>3,4 | 1	100	50
3,4=>1,5 | 1	50	25
4,5=>1,3 | 1	100	50
1=>3,4,5 | 1	14	3
3=>1,4,5 | 1	25	14
4=>1,3,5 | 1	20	9
5=>1,3,4 | 1	50	28
2,3=>4,5 | 1	50	25
2,5=>3,4 | 1	100	50
3,4=>2,5 | 1	50	30
4,5=>2,3 | 1	100	50
2=>3,4,5 | 1	16	5
3=>2,4,5 | 1	25	14
4=>2,3,5 | 1	20	9
5=>2,3,4 | 1	50	28
1,2=>3,4,5 | 1	20	4
1,3=>2,4,5 | 1	33	8
1,4=>2,3,5 | 1	25	5
1,5=>2,3,4 | 1	100	50
2,3=>1,4,5 | 1	50	25
2,5=>1,3,4 | 1	100	50
3,4=>1,2,5 | 1	50	25
4,5=>1,2,3 | 1	100	50
1=>2,3,4,5 | 1	14	3

2=>1,3,4,5 | 1	16	5
3=>1,2,4,5 | 1	25	14
4=>1,2,3,5 | 1	20	9
5=>1,2,3,4 | 1	50	28

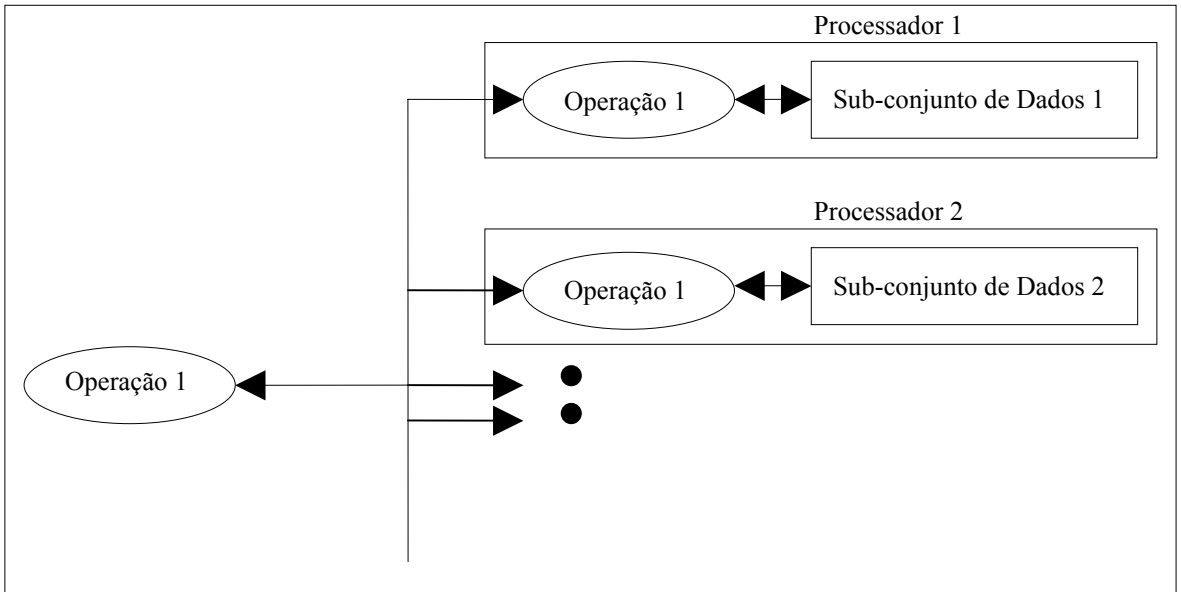
Como resultado da aplicação do algoritmo *Dense - Miner* é possível verificar a geração de um conjunto extremamente conciso de regras, sendo que em todas as regras, o conjunto de itens presente no antecedente contribui decisivamente para a habilidade preditiva da regra.

3.5 Algoritmos Paralelos

O tempo de execução dos algoritmos é um ponto crítico na utilização de ferramentas de mineração de dados, visto que em geral os bancos de dados onde estas ferramentas são aplicadas são densos, o que implica em um tempo de execução extremamente grande. Visto que a tarefa de encontrar conjuntos de itens freqüentes é a parte do problema mais dispendiosa em termos de processamento, conforme discutido na seção 2.2, tem-se procurado métodos que mais eficientemente executem esta tarefa com a diminuição do tempo de processamento. Os próximos algoritmos apresentados utilizam como técnica para redução de seu tempo de execução a paralelização de algumas atividades.

O paralelismo pode ser explorado de duas maneiras distintas: uma é o paralelismo de dados e outra é o paralelismo de controle [FRE98].

No paralelismo de dados o conjunto de transações a serem analisadas é dividido em múltiplos sub - conjuntos, sendo que a mesma operação é executada em diferentes sub - conjuntos ao mesmo tempo, enquanto no paralelismo de controle são executadas operações distintas no mesmo conjunto de dados. Também é importante salientar que as duas técnicas podem ser utilizadas em conjunto. O paralelismo de dados é mais indicado para problemas com bancos de dados muito grandes, enquanto o paralelismo de controle é mais indicado para tratar o problema de um espaço de pesquisa muito grande.



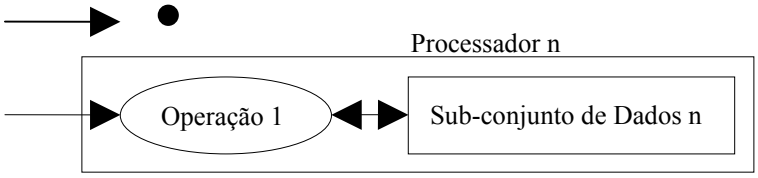


FIGURA 3.13 - Paralelismo de dados

A figura 3.13 ilustra o paralelismo de dados e a figura 3.14 ilustra o paralelismo de controle.

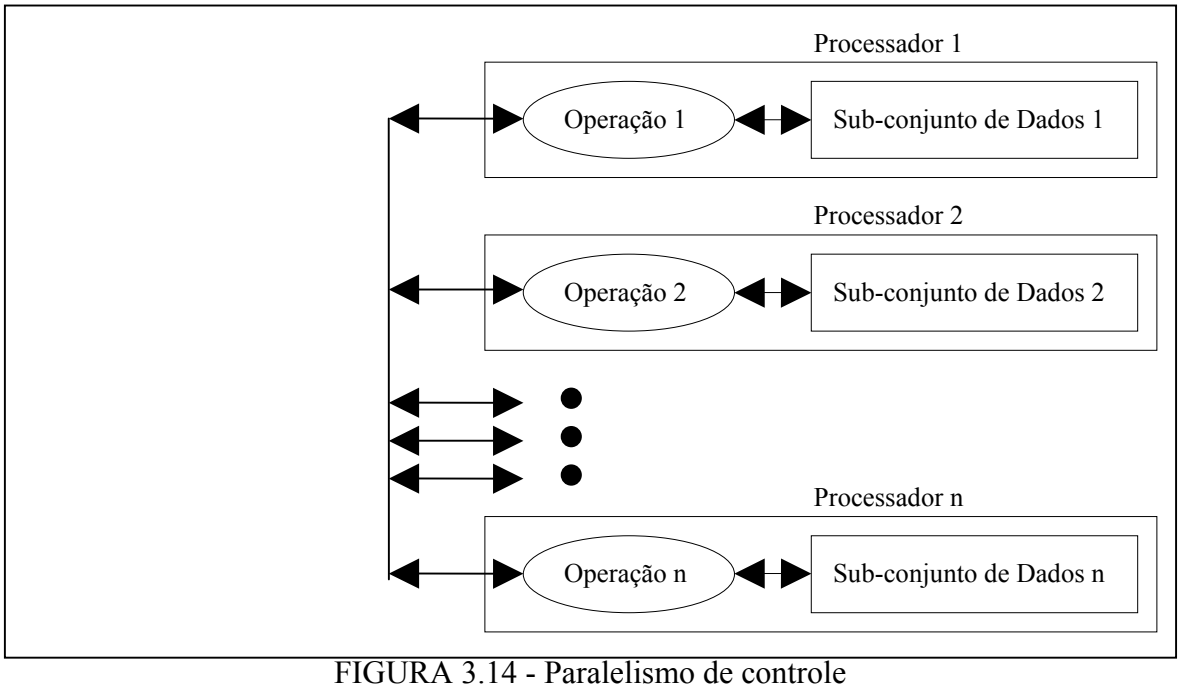


FIGURA 3.14 - Paralelismo de controle

De acordo com [BRU98], o paralelismo de dados têm três vantagens principais sobre o paralelismo de controle:

1. O algoritmo utilizado para processamento seqüencial pode ser reaproveitado com poucas alterações para o processamento com paralelismo de dados, visto que o fluxo de controle é o mesmo para ambos os casos, o que facilita a atividade do programador.
2. O paralelismo de dados é altamente independente da arquitetura do ambiente de hardware, enquanto o mesmo não acontece com o paralelismo de controle. O problema de dependência da arquitetura do ambiente deve ser tratado pelo programador.
3. No paralelismo de controle, a medida que o conjunto de processadores cresce, cresce também a necessidade de comunicação entre eles, enquanto no paralelismo de dados a comunicação entre processadores permanece praticamente a mesma.

3.6 Algoritmo *Count Distribution*

Em [AGR96a] é apresentado o algoritmo *Count Distribution* (*CD*) que utiliza o paralelismo de dados, ou seja, o conjunto de transações é dividido em tantos sub - conjuntos quantos forem os processadores disponíveis para execução da tarefa. Cada processador armazena localmente seu sub - conjunto de transações e também descobre os conjuntos de dados freqüentes em seu sub - conjunto local de dados através da aplicação do algoritmo. A figura 3.15 exibe o funcionamento do algoritmo *Count Distribution*.

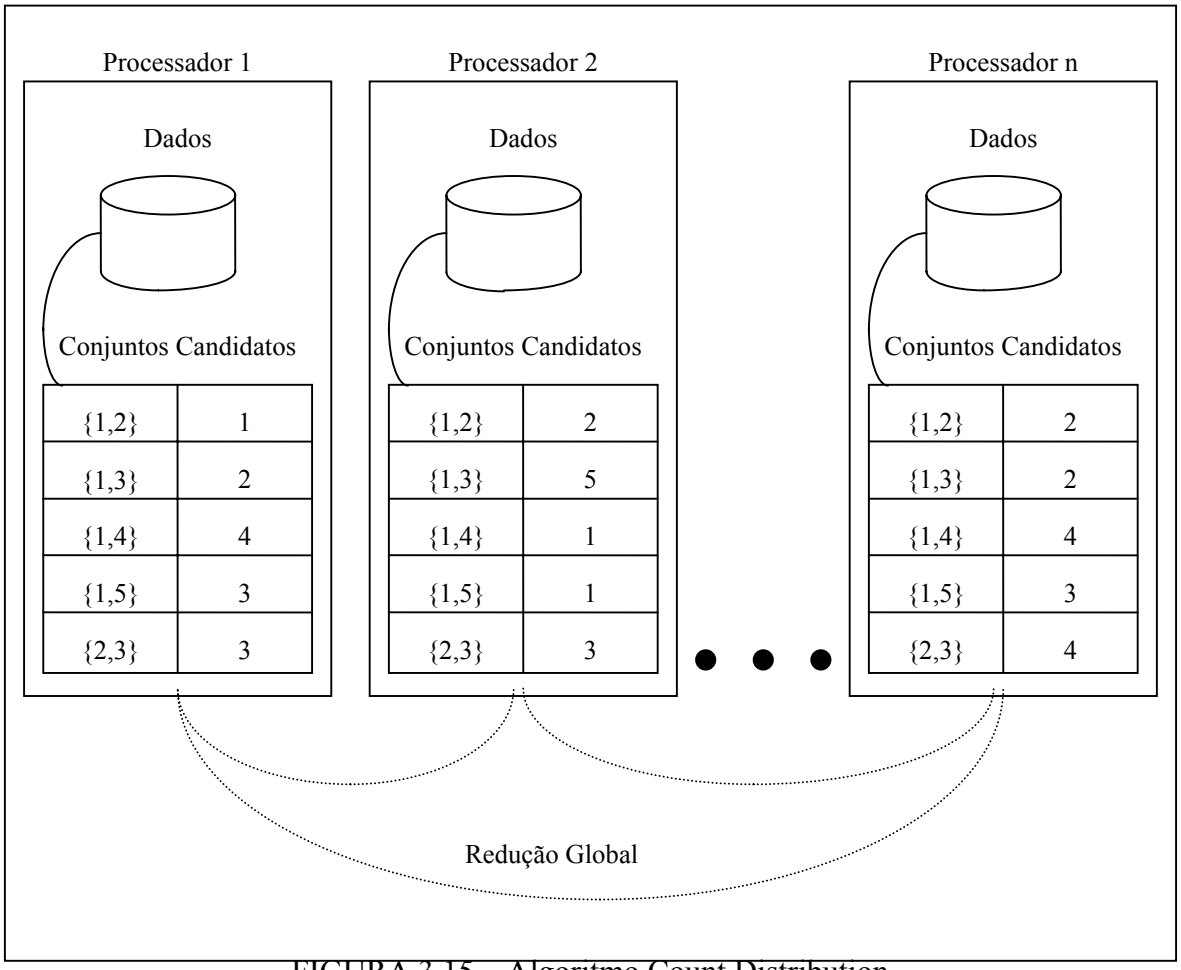


FIGURA 3.15 – Algoritmo Count Distribution

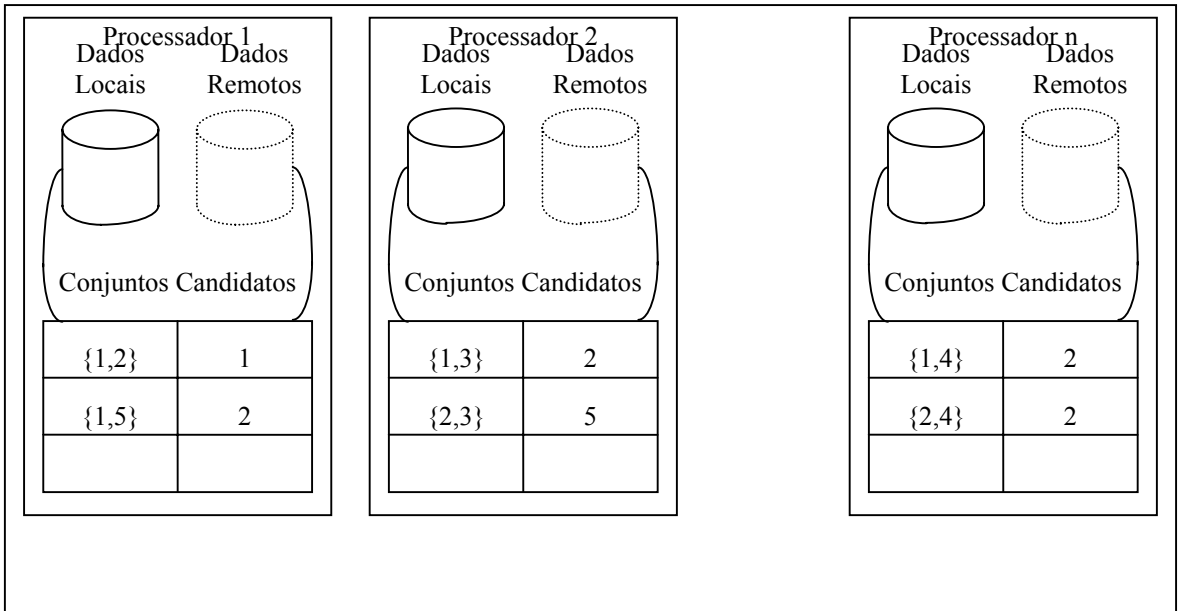
Em cada passagem do algoritmo *Count Distribution*, cada processador P^i calcula o suporte de seus conjuntos de itens candidatos locais C_1^i através da análise de seu conjunto de transações local D^i, através da aplicação do algoritmo *Apriori* localmente.

A contagem global do suporte dos conjuntos de itens candidatos é calculada pela soma das contagens locais dos conjuntos de itens em cada processador, este processo é denominado de redução global, tal processo é executado após todos os processadores terem acabado a passagem sobre o banco de dados e contado o suporte dos conjuntos de itens, e consiste no envio do suporte calculado pelo processador local para os demais processadores.

3.7 Algoritmo *Data Distribution*

Em [AGR96a] é apresentado o algoritmo *Data Distribution* (*DD*) que utiliza paralelismo de controle. Cada processador é responsável pela contagem de um subconjunto dos conjuntos de itens candidatos utilizando todas as transações no banco de dados, sendo que parte das transações estão armazenadas na memória local do processador atual e outra parte está armazenada remotamente na memória de outros processadores. A comunicação entre os processadores aumenta, visto que a necessidade de troca de informações é muito maior que no algoritmo Count Distribution, pois parte dos dados é armazenada localmente e outra parte remotamente. A necessidade de comunicação poderia ser minimizada caso os dados , na sua totalidade, estivessem armazenados localmente, mas quando se trata de bancos de dados com tamanho muito grande não poderia ser armazenado exclusivamente na memória local. Nesta solução há trabalho redundante visto que cada transação é processada por todos os processadores [HAN97], o que indica que a quantidade de processamento não foi reduzida significantemente.

Para encontrar o suporte de seus conjuntos de itens candidatos, cada processador necessita ler além de suas transações locais, transações armazenadas na memória de processadores remotos. Cada um dos P processadores aloca B *buffers* para receber os dados. Em qualquer processador o *buffer* B_i é utilizado para receber dados armazenados na memória do processador P_i. Os processadores processam todas as páginas até que não haja mais dados nas mesmas, atualizando o suporte de seus conjuntos de itens candidatos, após este processamento o conteúdo *buffer* B_i do processador P_i é enviado para os demais processadores, e o conteúdo do *buffer* B_i do processador P_i é limpo para receber os dados do processador não P .



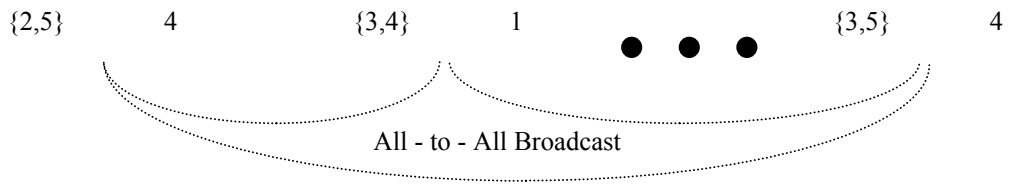


FIGURA 3.16 – Algoritmo Data Distribution

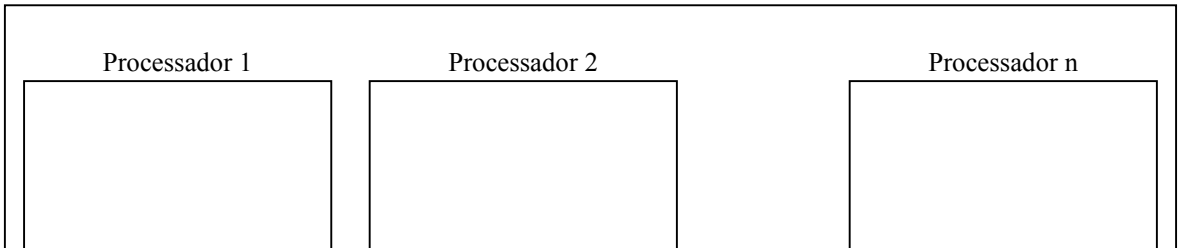
A figura 3.16 exibe o funcionamento do algoritmo *Data Distribution*.

Após a execução da contagem de suporte dos conjuntos de itens candidatos em todos os processadores, cada processador descobre seus conjuntos de itens freqüentes locais e os envia para os demais processadores simultaneamente usando uma operação de transmissão de todos para todos (*all-to-all broadcast*). Em alguns casos é necessário que um único processador envie os dados a todos os outros processadores, neste caso a operação de transmissão é de um para todos (*one-to-all broadcast*).

3.8 Algoritmo *Intelligent Data Distribution*

Em [HAN97] e [HAN2000] é apresentado o algoritmo *Intelligent Data Distribution* (*IDD*) que resolve o grande problema do algoritmo *Data Distribution*, que é a grande comunicação entre os processadores, resolvido através do uso de um anel lógico. Cada processador determina seus vizinhos esquerdo e direito, e possui um *buffer* de recepção e um *buffer* de envio, que inicialmente, é preenchido com um bloco de dados locais. A partir daí, cada processador inicia operações assíncronas de envio para o vizinho direito e de recepção do vizinho esquerdo. Enquanto estas operações são executadas cada processador analisa as transações do *buffer* de envio e conta o suporte de seus conjuntos de itens candidatos. Ao concluir esta operação, cada processador aguarda que estas operações assíncronas sejam concluídas. Supondo que P seja o número de processadores, então o conteúdo dos *buffers* da esquerda e da direita são trocados por P - 1 vezes. Ao final de cada passagem todos os conjuntos de itens freqüentes são transmitidos para os processadores vizinhos.

A partir do momento que o número de itens candidatos estiver abaixo de um limiar, passa-se a utilizar o algoritmo *Count Distribution*, a fim de diminuir a necessidade de comunicação entre os processadores. A fim de eliminar trabalho redundante passou-se a utilizar um particionamento inteligente dos conjuntos de itens candidatos entre os processadores, de modo que cada processador receba conjuntos de itens que comecem com um mesmo sub - conjunto de itens. Cada processador possui um mapa de *bits* que registra que itens iniciais de seus conjuntos de itens candidatos pertencem a um dado subconjunto de itens da transação, podendo determinar a partir do primeiro nível da árvore *hash* se estes conjuntos de itens estarão presentes na árvore sem precisar pesquisá-la. A figura 3.17 ilustra o funcionamento do algoritmo *Intelligent Data Distribution*.



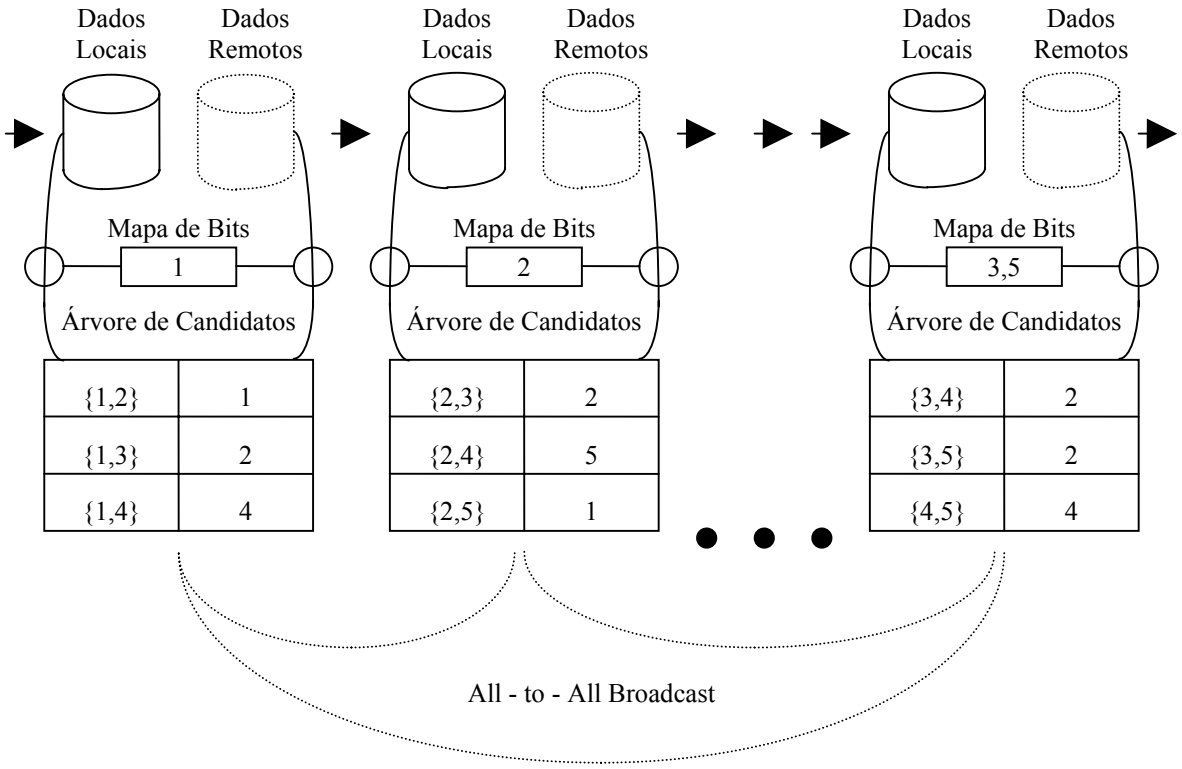


FIGURA 3.17 - Algoritmo IDD - Intelligent Data Distribution

3.9 Algoritmo *Hybrid Distribution*

Em [HAN97] e [HAN2000] é apresentado o algoritmo *Hybrid Distribution* (*HD*) que combina as características do algoritmo *Count Distribution* (*CD*) e do algoritmo *Intelligent Data Distribution* (*IDD*). O algoritmo *Hybrid Distribution* explora a memória total do sistema enquanto minimiza a comunicação envolvida entre processadores. A quantidade média de conjuntos de itens candidatos alocada para cada processador é definida pela quantidade total de candidatos dividida pelo número de processadores. A medida que o número de processadores aumenta, diminui a quantidade média de conjuntos de itens candidatos para cada processador, até o ponto em que o tempo de computação começa a se tornar menor que o de comunicação, o que reduz a eficiência do algoritmo. Tem-se um sistema com *P* processadores em que os processadores são divididos em *G* grupos de igual tamanho, cada grupo um contendo *P*/*G* processadores. O algoritmo CD é executado como se houvesse apenas P/G processadores. As transações e os conjuntos de itens candidatos são particionadas entre os processadores de cada grupo, então deve ser executada a tarefa de cálculo de suporte para cada um dos sub conjuntos de transações, de cada um dos conjuntos de itens candidatos para cada um dos grupos de processadores. Logo após, cada grupo de processadores calcula o suporte utilizando o algoritmo IDD, também é executada a redução global entre os P/G grupos de processadores.

No exemplo abaixo é mostrado que dentro de cada processador é executado o algoritmo IDD, e para distribuir o suporte calculado por cada algoritmo é executado o

algoritmo CD. A figura 3.18 ilustra o funcionamento do algoritmo *Hybrid Data Distribution*.

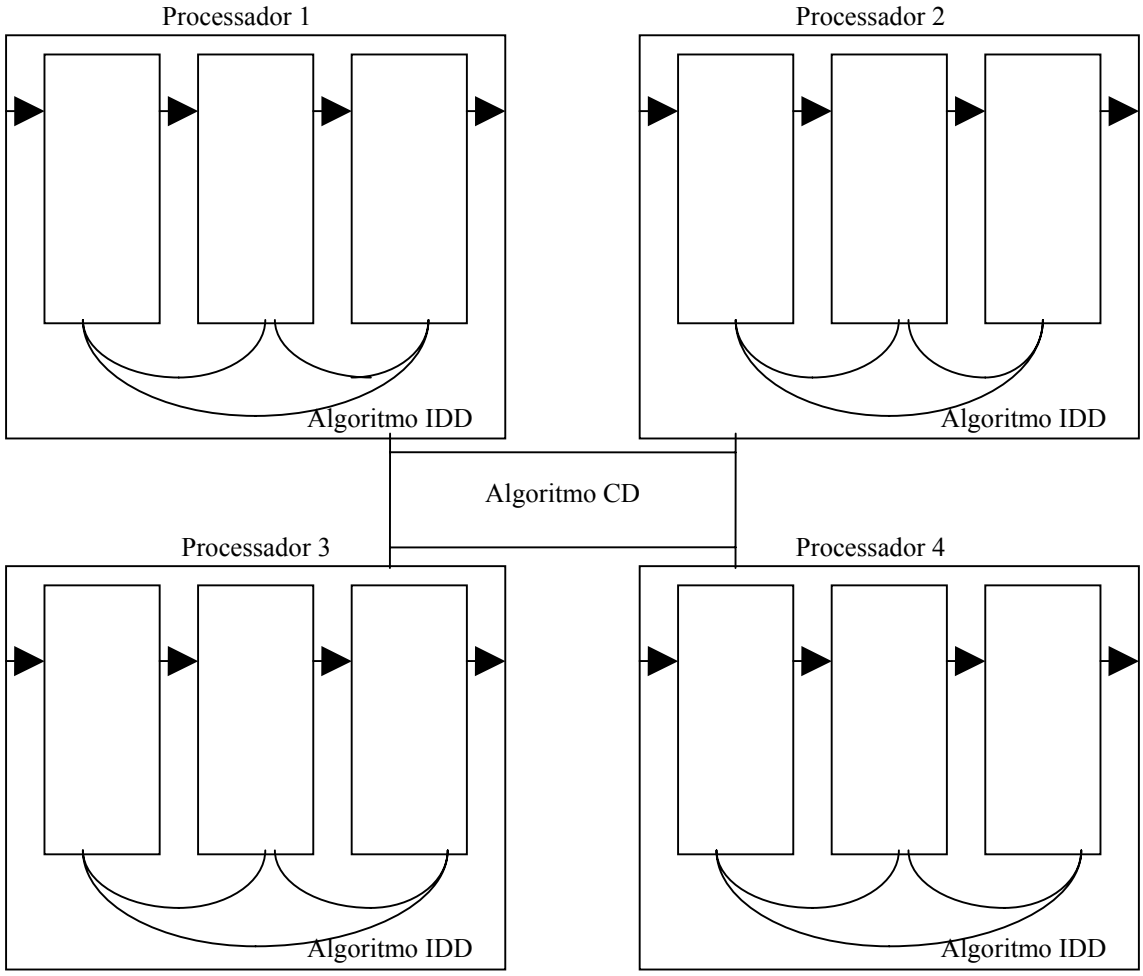


FIGURA 3.18 - Algoritmo Hybrid Distribution

3.10 Resumo

Após a análise destes algoritmos verificou-se a necessidade de um processo mais eficiente de geração de conjuntos de itens candidatos. Em qualquer dos algoritmos estudados é gerada uma grande quantidade de conjuntos de itens candidatos para cada passagem, e uma pequena proporção irá se tornar frequente. Um outro fator verificado é que os testes de desempenho sempre utilizam bancos de dados com um tamanho médio de transação variando com valores entre 14 e 50. Bancos de dados com um tamanho médio de transação inferior a 14 não são tratados em testes de desempenho nos algoritmos analisados.

Também após a análise dos algoritmos foi levantada a seguinte hipótese: a geração prévia (a priori) dos conjuntos de itens candidatos é sempre eficiente?

4 Proposta

Neste capítulo é apresentado um novo algoritmo para mineração de regras de associação, chamado MiRABIT (**Mi**neração de **R**egras de **A**ssociação em Bancos de dados com **B**aixa média de **I**tens por **T**ransação), e a ferramenta, chamada MIRA (**Mi**nerador de **R**egras de **A**ssociação), que implementa o algoritmo MiRABIT.

4.1 Motivação

De acordo com [ZAK97] e [ZAK97a] o algoritmo *Apriori* em cada passagem sobre o banco de dados, além da contagem de suporte dos conjuntos de itens candidatos desta passagem, gera o conjunto de itens candidatos para a próxima passagem. Do conjunto de itens candidatos, apenas um pequeno número é confirmado como freqüente na próxima passagem. Esta afirmação motivou a investigação da real necessidade da geração do conjunto de itens candidatos para a próxima passagem do algoritmo sobre o banco de dados, em função das características do banco de dados. Para esta investigação foram utilizados três bancos de dados com características distintas em relação à distribuição de itens por transação: um banco de dados de um supermercado, um banco de dados de uma empresa de comércio varejista de confecção e um banco de dados de um comércio varejista de ferragens. Os bancos de dados utilizados apresentam características peculiares, como o tamanho médio das transações, que refletem as características do negócio. A seguir são apresentadas os gráficos de tamanho das transações para cada um dos bancos de dados utilizados.

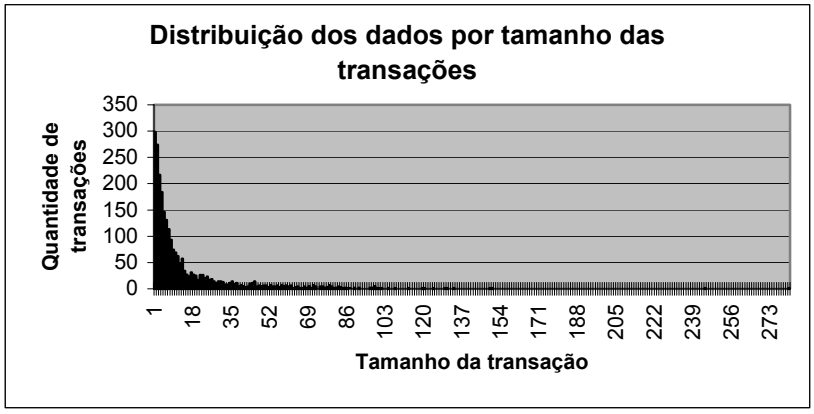


FIGURA 4.1 – Distribuição do tamanho das transações em um banco de dados de supermercado

A figura 4.1 mostra o gráfico de distribuição dos dados por tamanho das transações em um banco de dados real de um ambiente de supermercado. Neste banco de dados foram analisadas, no período de um mês de vendas, 2449 transações com 34375 itens o que resulta em uma média de tamanho das transações de aproximadamente 14,03 itens. O tamanho da maior transação encontrada era de 281 itens.

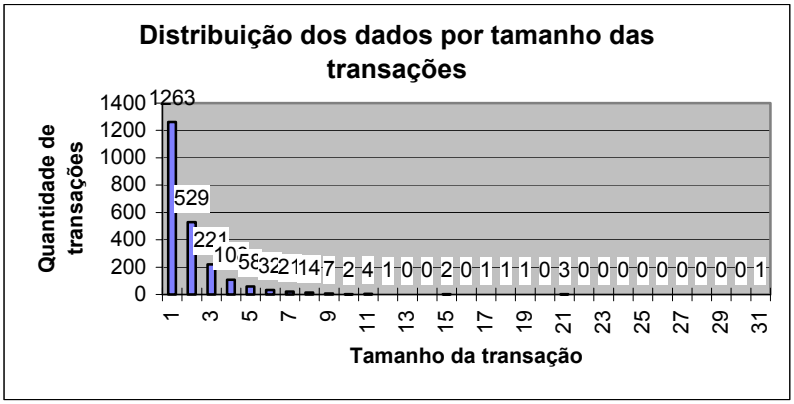


FIGURA 4.2 – Distribuição do tamanho das transações em um banco de dados de uma empresa de comércio varejista de confecção.

A figura 4.2 apresenta o gráfico de distribuição dos dados por tamanho das transações em um banco de dados real de uma empresa de comércio varejista de confecção. Neste banco de dados foram analisadas, no período de um mês de vendas, 2266 transações e 4380 itens o que resulta em uma média de aproximadamente 1,93 itens por transação. A maior transação encontrada possuía 19 itens.

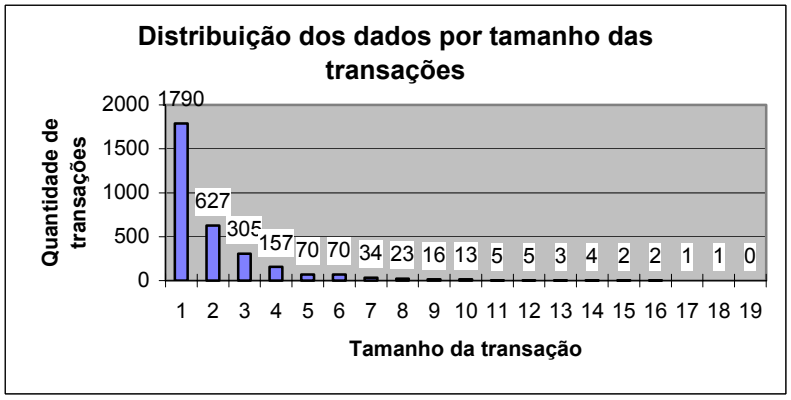


FIGURA 4.3 - Gráfico do tamanho das transações em um banco de dados de uma empresa de comércio varejista de ferragens.

A figura 4.3 apresenta o gráfico de distribuição dos dados por tamanho das transações em um banco de dados real de uma empresa de comércio varejista de ferragens. Neste banco de dados foram analisadas, no período de um mês de vendas, 3132 transações e 6454 itens o que resulta em uma média de aproximadamente 2,06 itens por transação. A maior transação encontrada possuía 31 itens.

Nos trabalhos encontrados na literatura que aplicaram o algoritmo *Apriori*, verificou-se que os bancos de dados utilizados correspondiam ao setor de supermercados, com um tamanho médio de transação entre 5 e 20. Em [BAY99] e [BAY99a], por exemplo, os testes de desempenho são executados em bancos de dados com tamanho médio de 43 e 49 itens por transação. O banco de dados de supermercado analisado neste trabalho, tem um tamanho médio de transação que também se situa neste intervalo. Entretanto, tanto os setores de comércio varejista de confecção, como o setor de ferragens, apresentam um tamanho médio de transação bem abaixo do intervalo correspondente aos supermercados. A partir desta constatação, levanta-se a hipótese de que o desempenho da tarefa de descoberta de regras de associação pode ser influenciado decisivamente pelo tamanho médio das transações do banco de dados analisado.

Conforme demonstrado através desta análise de diferentes bancos de dados, percebe-se que há bancos de dados com outras características diferentes das de supermercados. Estas outras características podem ser exploradas a fim de aumentar o desempenho da tarefa de descoberta de regras de associação em bancos de dados com características diferentes.

Neste sentido, foram executados vários experimentos com os algoritmos estudados, analisando-se seus pontos críticos de desempenho. Após a análise do algoritmo *Apriori* proposto por [AGR93], identificou-se que um dos pontos críticos de desempenho do algoritmo é a geração de conjuntos de itens candidatos para a próxima passagem sobre o banco de dados, que é executada no final de cada passagem. Conforme demonstrado posteriormente através da análise do banco de dados típico de um supermercado, verificou-se que, na segunda passagem do algoritmo, 34% dos conjuntos de itens candidatos tornaram-se freqüentes. Por outro lado verificou-se que, em um banco de dados típico de uma empresa do comércio varejista de confecção, apenas 0,03% dos conjuntos de itens candidatos tornaram-se freqüentes. Constatou-se então que a geração dos conjuntos de itens candidatos para a próxima passagem melhora o desempenho do algoritmo apenas quando grande parte dos conjuntos de itens candidatos torna-se freqüente. Para que isto ocorra é necessário que a média de itens por transação seja alta. No banco de dados do supermercado esta média era em torno de 14 itens por transação, já no comércio varejista era em torno de 1,93 itens por transação. A geração dos conjuntos de itens candidatos torna-se mais crítica em um ambiente de comércio varejista de confecção, e outros bancos de dados com tamanho médio de transação similar, onde a razão entre itens freqüentes e itens candidatos é bem menor que em um ambiente de supermercado. Constatou-se então que, nestes bancos de dados, seria mais eficiente gerar o conjunto de itens candidatos a cada transação do que gerar o conjunto de itens candidatos no final da passagem.

A partir desta análise é proposto que, em bancos de dados com uma pequena quantidade de itens por transação, a geração de conjuntos de itens candidatos seja executada a cada transação, e não no início da passagem, como é feito pelo algoritmo *Apriori*, resultando em uma melhora de desempenho da tarefa de descoberta de regras de associação.

Para provar esta hipótese foram executados experimentos para verificar o desempenho de um novo algoritmo, baseado no algoritmo *Apriori* [AGR93] com um processo de geração de conjuntos de itens candidatos mais eficiente. Os resultados apresentados no capítulo 5 demonstram claramente uma melhora de desempenho do algoritmo proposto em relação ao algoritmo *Apriori* nas áreas de comércio varejista de confecção e comércio varejista de ferragens.

4.2 Algoritmo MiRABIT

A partir da análise desenvolvida no item anterior, é proposto um novo algoritmo, chamado MiRABIT, apresentado em [CAM2001a] e [CAM2001b], cuja principal característica é que a geração dos conjuntos de itens candidatos para a passagem k é feita durante a passagem k e não durante a passagem (k-1) tal qual o algoritmo *Apriori* [AGR93] e similares. O principal objetivo desta característica é melhorar o desempenho da tarefa de mineração de regras de associação em bancos de dados com um baixo tamanho médio de transações. Como conseqüência desta alteração, o algoritmo MiRABIT gera um conjunto de itens candidatos geralmente entre 0,5 e 2 % do conjunto de itens gerados pelo *Apriori* [AGR93] no caso em estudo, o que resulta em um relevante ganho de desempenho.

A figura 4.4 apresenta o pseudocódigo do algoritmo MiRABIT.

```
F1 = {conjuntos de itens freqüentes com 1- elemento};
  for (k = 2; (( |Fk-1| ≥ 2) or (k ≤ tamanho_maximo_regra)); k++) do begin
    forall transações t ∈ D do begin
      Ct = subset (Fk-1,t,k); // Gera possíveis candidatos na transação
      forall candidatos c ∈ Ct do
        c.count ++;
      Ck=Ck ∪ Ct;
    end
    Fk = {c ∈ Ck | c.count ≥ suporte_minimo}
    Answer=Answer ∪ Fk;
  end
```

FIGURA 4.4 - Pseudocódigo do algoritmo MiRABIT.

A primeira linha do pseudocódigo apresentado na figura 4.4 demonstra que a execução do algoritmo parte do conjunto de itens freqüentes com 1 elemento. Para gerar este conjunto, o algoritmo necessita passar uma vez sobre o banco de dados contando em quantas transações cada item está presente. De acordo com os conceitos apresentados no item 2.3.1, para que um item seja considerado freqüente é necessário que seu suporte seja maior que um limiar previamente definido pelo usuário chamado suporte mínimo.

A segunda linha do pseudocódigo define uma estrutura de controle de iteração. Esta iteração será controlada pela variável *k* que identifica o número da passagem do algoritmo sobre o banco de dados. A variável *k* tem seu valor inicial definido como 2 e a iteração será executada enquanto o conjunto de itens freqüentes gerado na passagem anterior tiver 2 ou mais elementos. A iteração deixará de ocorrer se o conjunto tiver menos de dois elementos pois será feita uma combinação de *n* itens em conjuntos com *p* itens, logo *p* não poderá ser menor que *n*. Outra condição para a iteração ser executada é que *k* seja menor ou igual ao valor definido pelo usuário como tamanho máximo da regra. Esta restrição limita a quantidade de passagens do algoritmo sobre o banco de dados, diminuindo também o tempo para execução do algoritmo. A cada nova iteração, o valor da variável *k* é incrementado de uma unidade.

A cada nova passagem sobre o banco de dados diminui a probabilidade de ser encontrado um conjunto de itens freqüente; por este fato, a utilização de uma restrição de tamanho máximo da regra pode ser muito importante. Tendo-se um banco de dados, a probabilidade de um item qualquer ser freqüente é determinada pela razão entre o número de itens freqüentes na primeira passagem e o número total de itens; este resultado é elevado à ordem da passagem. Por exemplo, em um banco de dados com a distribuição de itens por transação mostrada na figura 4.1, temos 1263 / 34375 = 0,0367 ≅ 3,67% de probabilidade para a primeira passagem. A probabilidade de um conjunto de 2 itens ser freqüente é $0{,}0367^2 = 0{,}0013 \cong 0{,}13\%$. A probabilidade de um conjunto de 3 itens ser freqüente é $0{,}0367^3 = 0{,}0000494 \cong 0{,}00494\%$.

A terceira linha do pseudocódigo define uma nova estrutura de controle de iteração que obriga a execução das linhas seguintes para todas as transações no banco de dados. Neste ponto é importante salientar-se que a seleção, que é um dos passos do pré-processamento, já foi executada, sendo portanto consideradas somente as transações que atendem as restrições pré-estabelecidas.

A quarta linha do pseudocódigo executa a função *subset* que terá seu funcionamento detalhado posteriormente. A função *subset* gera todos os conjuntos de itens candidatos para cada transação.

Na quinta linha do pseudocódigo é definida uma estrutura de controle de iteração que obriga a execução das linhas posteriores para todos os conjuntos de itens gerados anteriormente pela função *subset*.

Na sexta linha é definida a instrução para que o contador de cada conjunto de itens encontrado na transação seja incrementado em uma unidade.

Na sétima linha o conjunto de itens candidatos encontrado na transação (C_t) é inserido no conjunto de itens candidatos da passagem.

Na oitava linha é finalizada a estrutura de controle iniciada na terceira linha.

Na nona linha do pseudocódigo é definida a instrução que irá incluir os conjuntos de itens candidatos com suporte maior ou igual ao suporte mínimo no conjunto de itens

freqüentes da passagem. Esta linha será executada após a análise de todas as transações do banco de dados selecionadas para serem mineradas.

Na décima linha do pseudocódigo é definida a instrução que irá inserir os conjuntos de itens freqüentes na passagem *k* ao conjunto resposta. Ao final da execução do algoritmo o conjunto resposta conterá todos os conjuntos de itens freqüentes encontrados em cada uma das passagens do algoritmo sobre o banco de dados juntamente com seus respectivos contadores de suporte.

Na décima primeira linha do algoritmo é definido o fim da estrutura de controle iniciada na segunda linha e finalizada a execução do algoritmo.

A partir dos conjuntos de itens freqüentes são geradas as regras de associação que satisfaçam a restrição de confiança mínima definida pelo usuário.

4.2.1 Função *Subset*

No algoritmo MiRABIT a função *subset* tem como finalidade principal identificar que conjuntos de itens em uma transação são candidatos para posterior contagem destes conjuntos de itens. Esta função recebe como argumentos o conjunto de itens freqüentes na passagem anterior L_{k-1} e a transação atual *t*. Cada conjunto de itens *c* contido na transação que seja derivado de um conjunto de itens freqüentes na passagem anterior é incluído no conjunto de itens candidatos da transação C_t para sua posterior contagem.

if (**all** (*k*-1) – subsets **of** *c*) $\in F_{k-1}$ **then**
    $c \in C_t$;

FIGURA 4.5 - Pseudocódigo da função *subset* do algoritmo MiRABIT.

A figura 4.5 apresenta a função *subset*.

A função *subset* gera as possíveis combinações de itens em conjuntos de *k* itens e também implementa uma função de poda, onde para cada conjunto de itens gerado em uma transação, é verificado se algum de seus subconjuntos não é freqüente; em caso positivo, este conjunto de itens não é considerado candidato e por conseqüência, seu suporte não é contado.

Visto que o conjunto de itens conjuntos gerados é muito menor que no algoritmo *Apriori*, conforme demonstrado em experimentos no próximo capítulo, a quantidade de memória utilizada durante o processo também é menor.

4.3 Ferramenta desenvolvida

Para a realização dos experimentos descritos no capítulo 5, foi desenvolvida uma ferramenta que implementa o algoritmo MiRABIT apresentada em [CAM2001]. A ferramenta tem facilidades de acesso a bancos de dados e de consulta das regras de associação obtidas. Esta ferramenta também possibilita o controle de índices de desempenho utilizados para a comparação dos algoritmos implementados. Foram gravados em um arquivo de log, o tempo utilizado para cada uma das passagens sobre o banco de dados assim como a quantidade de conjuntos de itens gerados nas passagens. No capítulo 5 serão dados mais detalhes sobre o sistema de avaliação dos algoritmos. A seguir são apresentadas as características do protótipo da ferramenta desenvolvida.

4.3.1 Requisitos de Hardware

Os requisitos mínimos de hardware para utilização do protótipo são um microcomputador IBM-PC compatível 486, com 16Mb de memória RAM, e espaço disponível em disco de 50% do tamanho do banco de dados sendo analisado.

4.3.2 Requisitos de Software

O protótipo necessita de um equipamento com ambiente Windows. Já foram executados testes de compatibilidade com os sistemas operacionais Windows 95, Windows 98 e Windows ME. Ainda não foram executados testes do protótipo em microcomputadores com sistema operacional Windows NT. A resolução de vídeo recomendada é de 800 x 600 pixels.

4.3.3 Banco de dados analisado

Os dados utilizados para os testes do protótipo estão armazenados em um banco de dados relacional que utiliza três tabelas básicas, são elas:

- Tabela de transações;
- Tabela de itens;
- Tabela de produtos.

Como já foi definido anteriormente, uma transação é um conjunto de itens comprados por um cliente em uma mesma visita à empresa e produtos são os itens disponíveis na empresa para serem comprados por um cliente.

O protótipo também utiliza somente um subconjunto de campos de cada tabela, não sendo levados em consideração os demais campos.

Na tabela de transações são necessários os seguintes campos: número da transação e data da transação. Nesta tabela há dois índices: o índice 1 é composto pelo campo

número da transação e o índice 2 é composto pela concatenação dos campos data da transação e número da transação.

TABELA 4.1 - Descrição da tabela transações

Nome do campo	Descrição do conteúdo	Tipo de dado	Tamanho
Nota	Número da transação	Inteiro	6
Data	Data da transação	Data	8

Na tabela de itens são necessários os seguintes campos: número da transação e código do produto. Nesta tabela há um índice que é composto pela concatenação dos campos número da transação e código do produto.

TABELA 4.2 - Descrição da tabela itens

Nome do campo	Descrição do conteúdo	Tipo de dado	Tamanho
Nota	Número da transação	Inteiro	6
Codg	Código do item	Inteiro	6

Na tabela de produtos são necessários os seguintes campos: código do produto e descrição do produto. Nesta tabela há um índice composto pelo código do produto.

TABELA 4.3 - Descrição da tabela produtos

Nome do campo	Descrição do conteúdo	Tipo de dado	Tamanho
Codg	Código do produto	Inteiro	6
Descricao	Descrição do produto	String	36

A ferramenta Mira é baseada em um banco de dados relacional. Para se fazer a modelagem da arquitetura básica do banco de dados sobre o qual a ferramenta trabalha é utilizado o modelo ER por ser um modelo formal, preciso e não ambíguo [HEU98]. A figura 4.6 apresenta o modelo ER produzido na ferramenta ER Win.

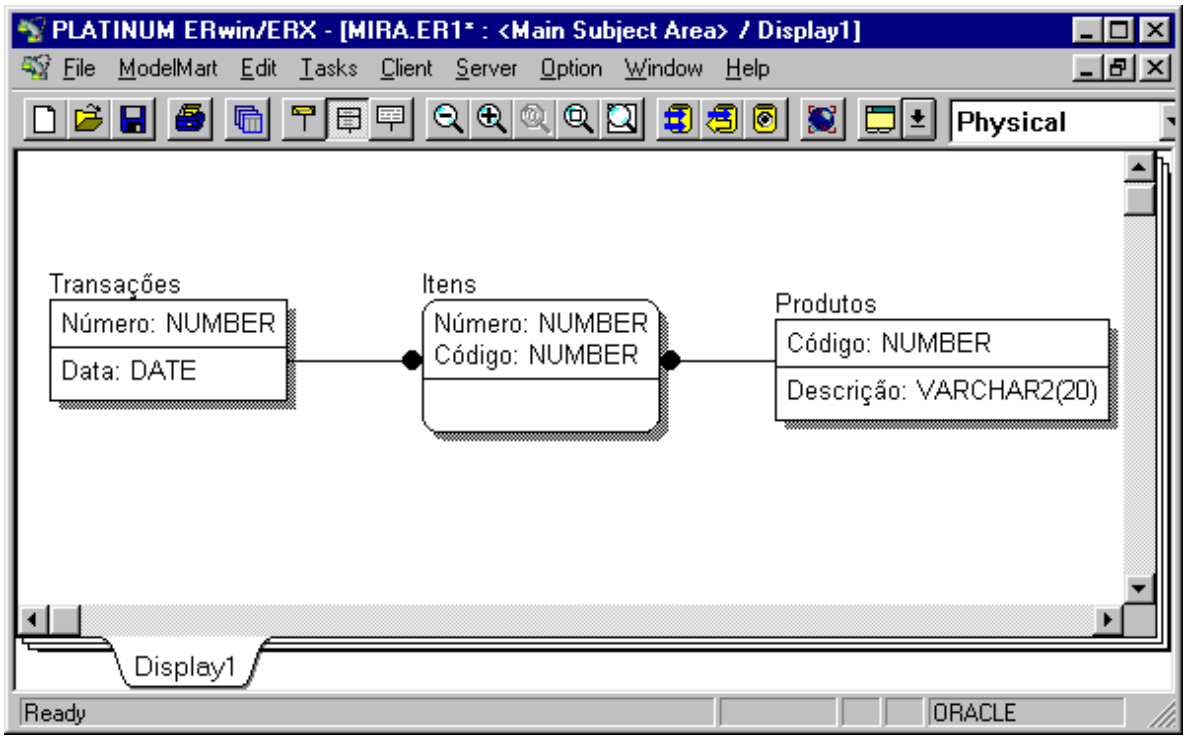


FIGURA 4.6 - Diagrama ER das tabelas utilizadas.

4.3.4 Arquitetura da ferramenta

A ferramenta é baseada em um banco de dados relacional. As tabelas utilizadas para o processo de mineração são as seguintes:

- Tabela de parâmetros;
- Tabela de conjuntos de itens freqüentes;
- Tabela de regras.

A tabela de parâmetros possui os seguintes campos:

TABELA 4.4 - Descrição da tabela parâmetros de processamento

Nome do campo	Descrição do conteúdo	Tipo de dado	Tamanho
Datainicial	Data inicial do processamento	Data	8
Datafinal	Data final do processamento	Data	8
Minsup	Suporte mínimo	Numérico	4
Minconf	Confiança mínima	Numérico	4,4
Maxlex	Tamanho máximo da regra	Numérico	4
Transações	Número de transações analisadas	Numérico	6
Itensvendidos	Número de itens vendidos	Numérico	6
Restrições	Indica quando aplicar restrições	String	1
Procseq	Indica se o processamento é seqüencial	String	1

A tabela de conjuntos de itens freqüentes possui os seguintes campos:

TABELA 4.5 - Descrição da tabela itens freqüentes

Nome do campo	Descrição do conteúdo	Tipo de dado	Tamanho
Item	Conjunto de itens	String	120
Suporte	Suporte do conjunto de itens	Numérico	4

A tabela de regras de associação possui os seguintes campos:

TABELA 4.6 - Descrição da tabela regras de associação

Nome do campo	Descrição do conteúdo	Tipo de dado	Tamanho
Regra	Descrição da regra	String	120
Suporte	Suporte da regra	Numérico	4
Confiança	Confiança da regra	Numérico	4,4

4.3.5 Operação do protótipo

O protótipo do sistema possui uma interface extremamente simples, sendo necessário ao usuário apenas um pequeno conhecimento sobre os conceitos de regras de associação para que o mesmo consiga utilizar o sistema. O usuário deve fornecer ao sistema o valor de suporte mínimo, confiança mínima e tamanho máximo da regra. A tela principal do protótipo é apresentada na figura 4.7.

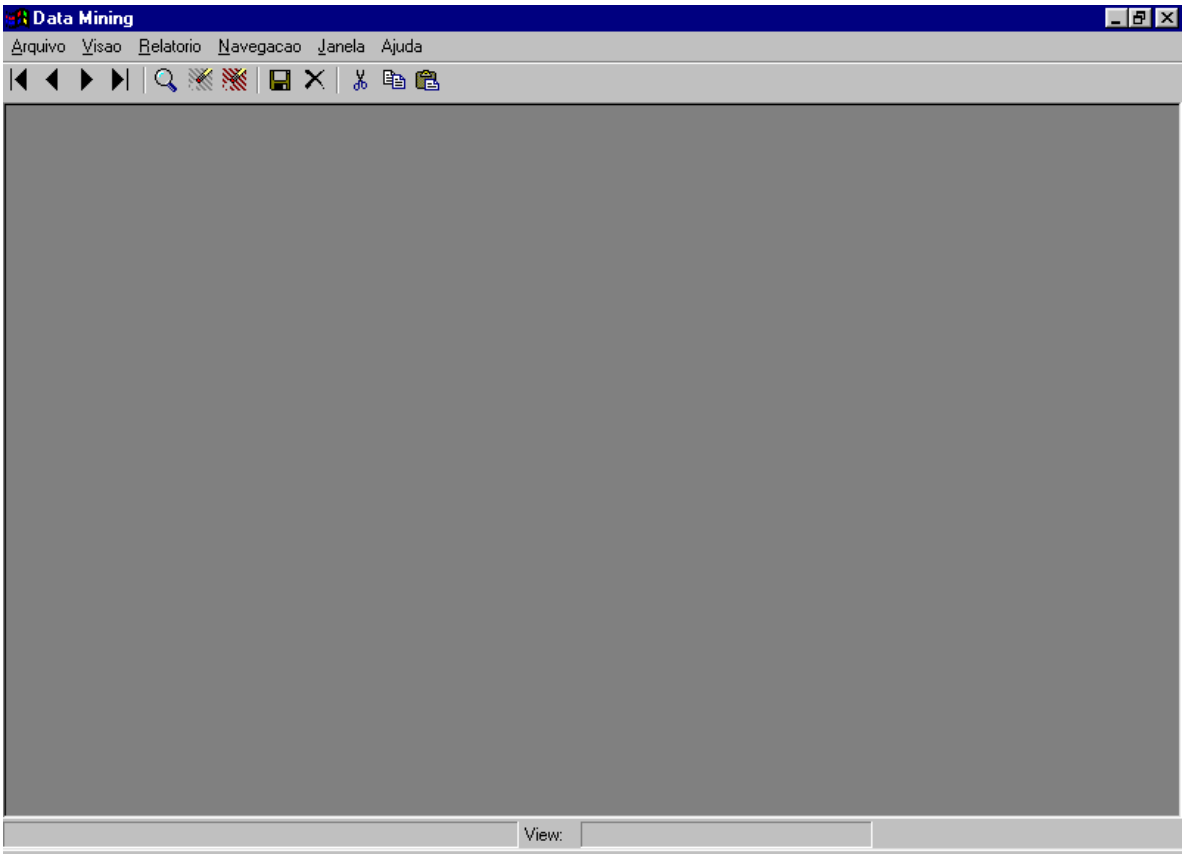


FIGURA 4.7 - Tela principal do protótipo

A tela de parâmetros é onde o usuário delimita os dados que devem ser selecionados e os parâmetros para o processo de mineração. O usuário pode selecionar as transações a serem analisadas entre um período qualquer de datas. Também, nesta visão, é informado o `suporte mínimo` para que um *itemset* seja considerado freqüente. `Confiança mínima` é o valor mínimo de confiança para que uma regra seja gerada. `Tamanho máximo da regra` é o número máximo de itens que devem existir em uma regra, este parâmetro também delimita o número máximo de passagens sobre o banco de dados. O parâmetro de `aplicar restrições` indica como os parâmetros devem influir sobre o processamento. Os valores que este parâmetro podem assumir são: nunca, processamento e relatórios. O valor dos campos transações processadas e itens processados são atualizados automaticamente pelo sistema quando o usuário inicia o processo de descoberta de regras.

A tela de descoberta de regras permite ao usuário iniciar o processo. Os parâmetros data inicial e data final não podem ser modificados pelo usuário nesta tela, conforme apresentado na figura 4.8.

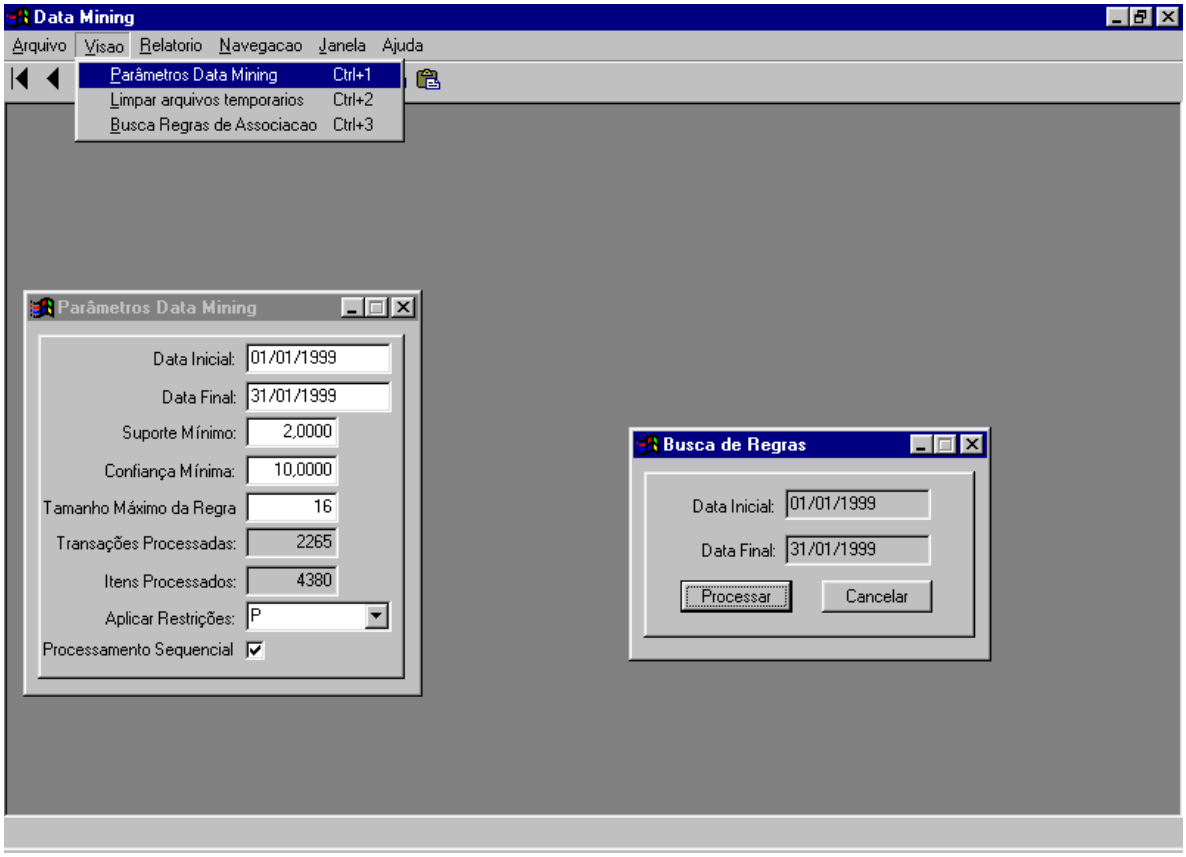


FIGURA 4.8 - Telas do menu visões

O protótipo possui um conjunto de relatórios que além de fornecer como saída o conjunto de regras, também pode fornecer suporte para a descoberta de regras de associação de maneira dirigida. A figura 4.9 apresenta os relatórios disponíveis no protótipo.

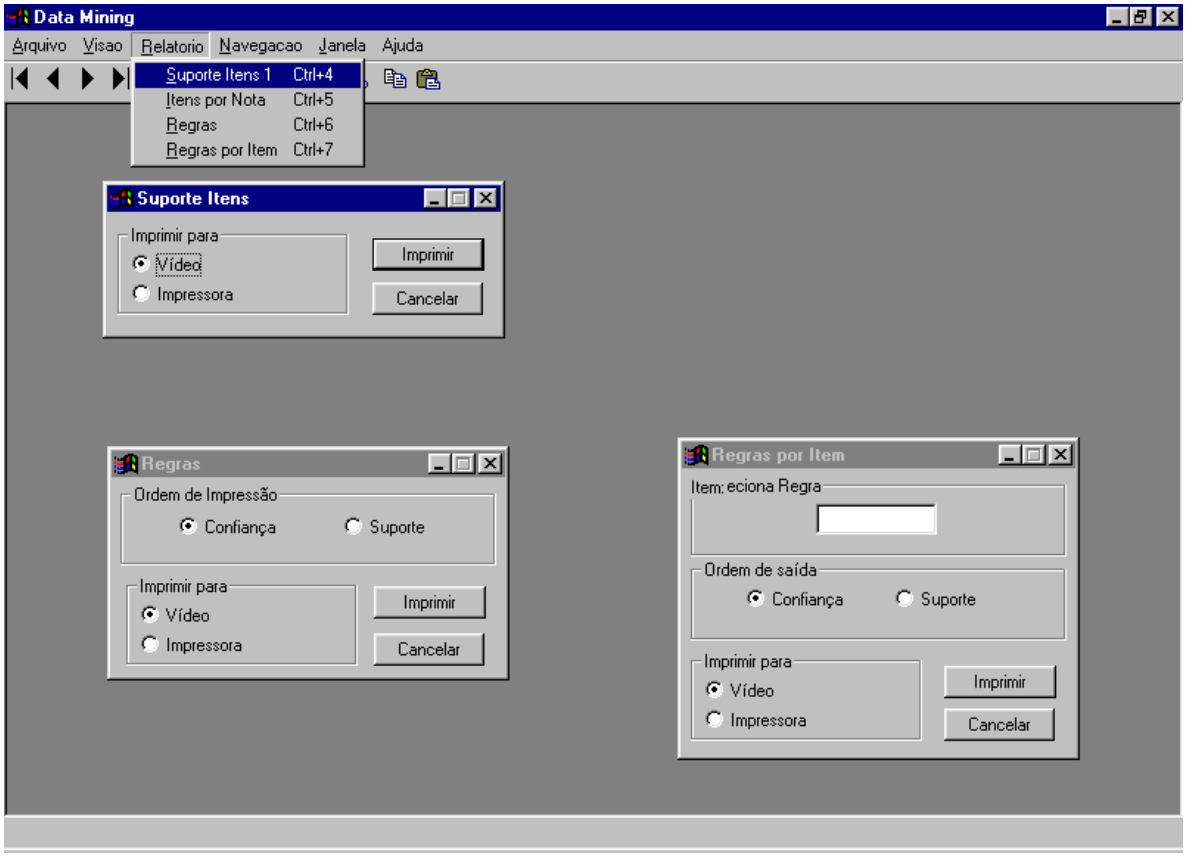


FIGURA 4.9 - Telas dos relatórios

A figura 4.10 demonstra o relatório de conjuntos de itens com 1 elemento. Este relatório mostra os seguintes campos: descrição do item, código do item e suporte do item. Este relatório é ordenado pelo suporte de cada item, itens com suporte maior são listados primeiro.

A figura 4.11 mostra um exemplo de relatório de regras ordenadas por suporte.

Descrição	Item	Suporte	Suporte %
RACAO CAMAL 18% VACA LEIT 40KG	7.394	520	16,60
07323QUIRERA DE ARROZ 60KG	7.323	109	3,48
07404R.PROFRANGO T 10KG	7.404	105	3,35
2967S.MILHETO (PASTO ITALIANO)	2.967	99	3,16
08881A	8.881	75	2,39
RACAO CAMAL 9% VACA SECA 40KG	7.395	73	2,33
98FORMICIDA PO-50 1KG NITROSIN	698	65	2,08
07402MILHO EM GRAO 50KG	7.402	63	2,01
07400SORGO EM GRAO 50KG	7.400	51	1,63
03515SAL MOIDO 25KG	3.515	45	1,44
07405R.TERNEIRA LAM 25KG	7.405	44	1,40
00474SEM.FISC.SORGO FOR.F-855	474	42	1,34
9551FITA ISOLANTE IMPERIAL 10M	9.551	42	1,34
02690ADESIVO SUPER BONDER 3G	2.690	41	1,31
03463PREGO GERDAU 17X27	3.463	40	1,28
07374CIMENTO VOTORAN 50KG	7.374	37	1,18
08888A	8.888	37	1,18
03224ELETRODO WELD 3,25MM	3.224	36	1,15
02683ADESIVO DUREPOXI 100G	2.683	31	0,99
LAMINA SERRA NICHOLSON BIMETAL	3.332	31	0,99
07401MILHO EM GRAO 1KG	7.401	31	0,99
04901PORCA SEXT. 5/16	4.901	30	0,96
01913CAL CREM C/FIXADOR 8KG	1.913	29	0,93
03279FORMICIDA LANDREX 500G	3.279	26	0,83
734PLUG MACHO BRANCO 51000	3.734	26	0,83
02738ARAME N.12 ATILHO	2.738	25	0,80
03683FIO ELETRICO DUPLO 2X2,50	3.683	25	0,80
04804ARRUELAS LISAS 3/8	4.804	25	0,80
04907PORCA SEXT. 3/4	4.907	25	0,80
02747ARAME N.06 RABICHO	2.747	23	0,73
04902PORCA SEXT. 3/8	4.902	21	0,67
04904PORCA SEXT. 1/2	4.904	21	0,67
07413ADUBO 45-00-00 50KG UREIA	7.413	21	0,67
09440FORMICIDA BLITZ F 500G	9.440	21	0,67
12709LINHA DRIMA 100 126	12.709	21	0,67
13629BOTINA PRIMAVERA	13.629	21	0,67
02909FLUIDO FREIO VARGA 500ML	2.909	20	0,64
02911ADUBO 10-20-10 50KG	2.911	20	0,64
03873REMENDO A FRIO R02	3.873	20	0,64
04900PORCA SEXT. 1/4	4.900	20	0,64
07399SORGO EM GRAO 1KG	7.399	20	0,64
4802ARRUELAS LISAS 1/4	4.802	19	0,61
803ARRUELAS LISAS 5/16	4.803	19	0,61
07304OLEO MOBIL SUPER 4L	7.304	19	0,61
CAMISETA INWIKO C/BOTAO ADULTO	13.531	19	0,61
0CAMISETA INWIKO C/DETALHE 2-8	14.400	19	0,61
14402CAMISETA INWIKO 2-8	14.402	19	0,61
02944CABO ENXADA OVAL	2.944	18	0,57
00372ADESIVO SUPER BONDER 5G	372	17	0,54
03756TOMADA LUZ EXTERNA	3.756	17	0,54
04194PINO ARGOLA 810285	4.194	17	0,54
CAL HIDRATADA PRIMOR EXTRA 20K	4.629	17	0,54
12160CAMISETA INWIKO P.M.G.	12.160	17	0,54
ARAME GERDAU 1250M. GALVANIZAD	2.355	16	0,51
03743PORTA LAMPADA SIMPLES	3.743	16	0,51
04906PORCA SEXT. 5/8	4.906	16	0,51
13401TOALHA TEKA BANHO FELPUDO	13.401	16	0,51
15440LINHA CARRETEL BRANCA N30	15.440	16	0,51
17674PANO DE COPA TEKA	17.674	16	0,51
04806ARRUELAS LISAS 1/2	4.806	15	0,48
11203CAMISA MASC. XADRES	11.203	15	0,48
13400TOALHA TEKA ROSTO FELPUDO	13.400	15	0,48
0393PORCA SEXT. 3/16	393	14	0,45
02625ELETRODO WELD 2,5MM	2.625	14	0,45

FIGURA 4.10 - Relatório de suporte de itens

```
Confiança    Suporte    Regra

   100,00       7,00     4822:04822ARRUELAS PRESSAO 5/8 => 4906:04906PORCA SEXT. 5/8
   100,00       7,00     4927:27PARAF.SEXT. 5/16X1 1/2 => 4901:04901PORCA SEXT. 5/16
   100,00       7,00     4967:04967PARAF.SEXT. 1/2X2 => 4904:04904PORCA SEXT. 1/2
   100,00       6,00     4928:04928PARAF.SEXT. 5/16X2 => 4901:04901PORCA SEXT. 5/16
   100,00       5,00     4817:7ARRUELAS PRESSAO 5/16 => 4901:04901PORCA SEXT. 5/16
   100,00       5,00     4827:04827PARAF.FENDA 3/16X1 => 393:0393PORCA SEXT. 3/16
   100,00       5,00     5009:05009PARAF.SEXT. 3/4X3 => 4907:04907PORCA SEXT. 3/4
   100,00       4,00     2355:ARAME GERDAU 1250M. GALVANIZAD + 2738:02738ARAME N.12
ATILHO => 2747:02747ARAME N.06 RABICHO

   100,00       4,00     2747:02747ARAME N.06 RABICHO + 4771:04771RETRANCAS EUCALIPTO =>
4770:4770TRAMA 5X5X1,40M. EUCALIPTO

   100,00       4,00     3494:94GRAMPO ISOLADO N.5 DEL SARTO + 3743:03743PORTA LAMPADA
SIMPLES => 3756:03756TOMADA LUZ EXTERNA

   100,00       4,00     3961:LUVA MISTA 25X3/4" SD-08 + 4007:TEE SOLDAVEL 25MM SD-25 =>
3997:SOLDA PVC 17ML SD-33

   100,00       4,00     3997:SOLDA PVC 17ML SD-33 + 4007:TEE SOLDAVEL 25MM SD-25 =>
3961:LUVA MISTA 25X3/4" SD-08

   100,00       4,00     4770:4770TRAMA 5X5X1,40M. EUCALIPTO + 4771:04771RETRANCAS
EUCALIPTO => 2747:02747ARAME N.06 RABICHO

   100,00       4,00     4802:4802ARRUELAS LISAS 1/4 + 4803:803ARRUELAS LISAS 5/16 +
4900:04900PORCA SEXT. 1/4 => 4901:04901PORCA SEXT. 5/16

   100,00       4,00     4802:4802ARRUELAS LISAS 1/4 + 4803:803ARRUELAS LISAS 5/16 +
4901:04901PORCA SEXT. 5/16 => 4900:04900PORCA SEXT. 1/4

   100,00       4,00     4802:4802ARRUELAS LISAS 1/4 + 4803:803ARRUELAS LISAS 5/16 =>
4900:04900PORCA SEXT. 1/4

   100,00       4,00     4802:4802ARRUELAS LISAS 1/4 + 4803:803ARRUELAS LISAS 5/16 =>
4900:04900PORCA SEXT. 1/4 + 4901:04901PORCA SEXT. 5/16

   100,00       4,00     4802:4802ARRUELAS LISAS 1/4 + 4803:803ARRUELAS LISAS 5/16 =>
4901:04901PORCA SEXT. 5/16

   100,00       4,00     4802:4802ARRUELAS LISAS 1/4 + 4900:04900PORCA SEXT. 1/4 +
4901:04901PORCA SEXT. 5/16 => 4803:803ARRUELAS LISAS 5/16

   100,00       4,00     4802:4802ARRUELAS LISAS 1/4 + 4901:04901PORCA SEXT. 5/16 =>
4803:803ARRUELAS LISAS 5/16

   100,00       4,00     4802:4802ARRUELAS LISAS 1/4 + 4901:04901PORCA SEXT. 5/16 =>
4803:803ARRUELAS LISAS 5/16 + 4900:04900PORCA SEXT. 1/4

   100,00       4,00     4802:4802ARRUELAS LISAS 1/4 + 4901:04901PORCA SEXT. 5/16 =>
4900:04900PORCA SEXT. 1/4

   100,00       4,00     4802:4802ARRUELAS LISAS 1/4 + 4917:04917PARAF.SEXT. 1/4X1 =>
4900:04900PORCA SEXT. 1/4

   100,00       4,00     4803:803ARRUELAS LISAS 5/16 + 4900:04900PORCA SEXT. 1/4 +
4901:04901PORCA SEXT. 5/16 => 4802:4802ARRUELAS LISAS 1/4

   100,00       4,00     4803:803ARRUELAS LISAS 5/16 + 4928:04928PARAF.SEXT. 5/16X2 =>
4901:04901PORCA SEXT. 5/16

   100,00       4,00     4826:04826PARAF.FENDA 3/16X3/4 => 393:0393PORCA SEXT. 3/16
   100,00       4,00     4900:04900PORCA SEXT. 1/4 + 4901:04901PORCA SEXT. 5/16 =>
4802:4802ARRUELAS LISAS 1/4
```

FIGURA 4.11 - Relatório de regras

4.2.6 Importação de dados

Os dados para a mineração podem ser importados de arquivos texto em formato ASCII. Como já foi mencionado anteriormente, os arquivos necessários para a mineração são três: arquivo de transações, arquivo de itens e arquivo de produtos. Os arquivos podem estar no formato de campos delimitados por linha ou registros delimitados por linha.

No caso do arquivo estar no formato de campo delimitado por linha o layout do arquivo deve ser como o demonstrado nas tabelas 4.7 a 4.9:

TABELA 4.7 - Layout para importação do arquivo de transações

Modelo	**Arquivo real**
Número da transação	37875
Data da transação	02/01/1999
Número da transação	37876
Data da transação	02/01/1999
Número da transação	37877
Data da transação	02/01/1999
...	...
...	...
Número da transação	197629
Data da transação	31/01/1999

TABELA 4.8 - Layout para importação do arquivo de itens

Modelo	**Arquivo real**
Número da transação	37875
Código do item	20717
Número da transação	37876
Código do item	4282
Número da transação	37877
Código do item	2079
Número da transação	37877
Código do item	7488
...	...
...	...
Número da transação	197629
Código do item	21087
Número da transação	197629
Código do item	21703
Número da transação	197629
Código do item	21906
Número da transação	197629
Código do item	22259

TABELA 4.9 - Layout para importação do arquivo de produtos

Modelo	Arquivo real
Código do item	2
Descrição do item	COLAR DE PEROLA C/PING HF
Código do item	6
Descrição do item	TEN BASKET OLYPIKUS 38/43+
Código do item	7
Descrição do item	LINHO RAMI SAO PAULO+
...	...
...	...
Código do item	27204
Descrição do item	FRIGIDEIRA FLASH C/T 22 PANEX +

No caso do arquivo estar no formato de registro delimitado por linha o layout do arquivo deve ser como o demonstrado nas tabelas 4.10 a 4.12:

Arquivo de transação
Número da transação = 6 inteiros
Data da transação = 8 data

TABELA 4.10 - Layout para importação do arquivo de transações

Modelo	Arquivo real
Número da transação + data da transação	00070202102000
Número da transação + data da transação	00070707102000
Número da transação + data da transação	00070909102000
...	...
...	...
Número da transação + data da transação	98482525102000
Número da transação + data da transação	98483131102000

Arquivo de detalhes de transação
Número da transação = 6 inteiros
Código do item = 5 inteiros

TABELA 4.11 - Layout para importação do arquivo de itens

Modelo	Arquivo real
Número da transação + código do item	00070200273
Número da transação + código do item	00070201617
Número da transação + código do item	00070206214
Número da transação + código do item	00070700582

...	...
...	...
Número da transação + código do item	98483101797
Número da transação + código do item	98483107214
Número da transação + código do item	98483109835
Número da transação + código do item	98483109842

Arquivo de itens
Código do item = 5 inteiros
Descrição do item = 30 string

TABELA 4.12 - Layout para importação do arquivo de produtos

Modelo	**Arquivo real**
Código do item + descrição do item	00002AGUA SANITARIA SUPER GLOBO 1LT
Código do item + descrição do item	00003MEMBRANA DO LAVADOR 230131
Código do item + descrição do item	00004REGULADOR DE VACUO 230157
...	...
...	...
Código do item + descrição do item	17692BOTA VULCABRAS PVC BRANCA 44/5
Código do item + descrição do item	17693BOTA CALFOR BRANCA C/C N. 43
Código do item + descrição do item	17694BOTA CALFOR BRANCA C/C N. 44

4.3.7 Arquivo de log

A cada processamento executado pelo sistema com a finalidade de descobrir regras de associação é gerado automaticamente um conjunto de informações no arquivo de log do sistema com dados referentes a este processamento.

Os dados gravados no arquivo de log estão divididos em duas partes. Na primeira parte estão os parâmetros de processamento fornecidos pelo usuário que são: intervalo de datas de processamento, valor do suporte mínimo, confiança mínima, local de aplicação das restrições e tamanho máximo da regra. Na segunda parte estão os valores gerados pelo algoritmo em cada passagem sobre o banco de dados para contagem dos conjuntos de itens freqüentes, estes valores são: o número da passagem, seu tempo e a quantidade de itens freqüentes encontrados. Durante o processo de geração de regras também são incluídas no arquivo de log a quantidade de regras geradas e o tempo de geração destas regras. Após a geração das regras também são incluídas pelo sistema as informações referentes a quantidade de transações processadas e itens vendidos no período.

Na figura 4.12 é demonstrado um exemplo de uma entrada no arquivo de log gerada por um processamento.

```
---------------------------------------------------------------------- Início do
Processamento
Data inicial: 01/01/1999      Data Final: 31/01/1999
Suporte Mínimo: 2      Confiança Mínima: 10
Restrições: P    Tamanho Máximo da Regra: 16    Processamento Seqüencial: S
-------------------------------------------------------------------------------------------------
------
```

FIGURA 4.12 - Exemplo de uma entrada no arquivo de log.

5 Descrição dos Experimentos

Neste capítulo são apresentados resultados de testes realizados com o protótipo em um banco de dados real de uma empresa do ramo de comércio varejista de confecção. Este processo tem como finalidade descobrir as regras de associação existentes entre itens em um conjunto de transações.

Segundo [TOS2001], a complexidade de um algoritmo pode ser medida pela complexidade de tempo e pela complexidade de espaço. A complexidade de tempo refere-se à velocidade do algoritmo. A complexidade de espaço refere-se à quantidade de memória necessária para a execução deste algoritmo.

Na avaliação de desempenho do algoritmo proposto serão utilizadas duas métricas, o tempo de execução e a quantidade de conjuntos de itens candidatos gerados, a primeira refere-se a complexidade de tempo, a segunda refere-se a complexidade de espaço.

Foram realizados nove experimentos com dois bancos de dados reais: quatro experimentos com um banco de dados de comércio varejista de confecção e cinco com um banco de dados de comércio varejista de ferragens. Os experimentos sobre os mesmos bancos de dados foram realizados com diferentes valores de suporte mínimo. A confiança mínima não foi levada em consideração por ser somente relevante para a geração das regras, e o escopo deste trabalho trata da geração de conjuntos de itens freqüentes.

Para cada um dos experimentos são apresentados os resultados obtidos com os algoritmos MiRABIT e *Apriori* [AGR93].

5.1 Experimento 1

No experimento 1, foi analisado um banco de dados de uma empresa do comércio varejista de confecção de Bagé. O banco de dados analisado possui 39403 transações, 84733 itens de transação, e 26330 produtos com um ano de movimento de vendas da empresa. Devido à sazonalidade da grande maioria dos produtos do ramo do comércio varejista de confecção, a análise de um período muito extenso, tal como um ano de vendas, deturparia o resultado, ocultando regras importantes. Por isso, foi então analisado somente um mês de vendas da empresa. Por exemplo, analisando-se um ano inteiro de vendas da empresa, foi descoberto que o produto *blusa básica 1x1* foi vendido em 381 de 39403 transações, o que significa um suporte de 0,99%. Através de uma consulta SQL sobre o banco de dados, verifica-se que o produto foi vendido pela primeira vez durante o ano de 1999 em 26/04 e pela última vez em 30/08. Agora, analisando-se o mês de junho de 1999, foi descoberto que o mesmo produto *blusa básica 1x1* foi vendido em 175 de 3152 transações, o que significa um suporte de 5,55%. Durante 7 meses do ano não há demanda por este produto, consequentemente o mesmo não está disponível para venda, não estando exposto tanto em vitrines quanto em expositores dentro da empresa.

Este experimento demonstra que em bancos de dados que obedecem a alguma sazonalidade, os resultados podem ser deturpados quando a análise é executada em um período muito extenso, sendo muito importante a mineração em períodos menores o que irá demonstrar o quão importante um produto pode ser dentro daquele período analisado.

5.1.1 Pré processamento

O primeiro passo do pré-processamento do banco de dados analisado foi a seleção de um determinado período de vendas para ser minerado pelo protótipo. O subconjunto selecionado consistia das transações efetuadas no período de 01 de janeiro de 1999 a 31 de janeiro de 1999, totalizando um conjunto de 2265 transações e 4380 itens processados. Não foram encontrados problemas de integridade e completude nos dados.

5.1.2 Mineração

Antes de começar a mineração, o usuário deve informar um conjunto de parâmetros para o processo, que são: suporte mínimo, confiança mínima, aplicar restrições, tamanho máximo da regra e processamento seqüencial.

Neste experimento os valores para os parâmetros foram os seguintes:

Suporte mínimo: 2
Confiança mínima: 10%
Aplicar restrições: Processamento
Tamanho máximo da regra: Sem limite
Processamento seqüencial: Sim

5.1.3 Visualização das regras

Durante o processo de visualização das regras encontradas o usuário poderá selecionar entre ordenar regras por suporte ou por confiança.

Também é possível ao usuário fazer uma descoberta de conhecimento dirigida, ou seja, o usuário poderá selecionar um determinado produto e só serão apresentadas regras que contiverem este produto no seu corpo.

5.1.4 Apresentação e discussão dos resultados

TABELA 5.1 - Comparação tempo de execução dos algoritmos

	Algoritmo *Apriori* [AGR93]		Algoritmo *Mirabit*		
Número da Passagem	**Tempo (s)**	**Conjuntos de itens candidatos gerados**	**Tempo (s)**	**Conjuntos de itens candidatos encontrados**	**Conjuntos de itens freqüentes encontrados**
1	21		8		823
2	93	338253	24	2493	124
3	7	310124	6	15	11
4	9	330	12	1	1
Total	**130**	**648707**	**50**	**2509**	**959**

TABELA 5.2 - Conjunto de regras de associação encontradas

Número da Passagem	**Tempo (s)**	**Regras encontradas**
1	3	222
2	1	33
3	0	0
Total	**4**	**255**

As tabelas 5.1 e 5.2 apresentam os resultados obtidos no experimento 1. Foram encontrados durante a primeira passagem 823 itens freqüentes. O algoritmo *Apriori* [AGR93], no final da primeira passagem, fez a combinação destes 823 itens freqüentes encontrados, gerando os conjuntos de itens candidatos para a segunda passagem. Através de análise combinatória tem-se que $C_{n,p} = (n!)/(p!* (n-p)!)$ ou seja, (823!) / (2! * (823-2)!) = 823! / (2! * 821!) = 338253. Logo, foram gerados 338253 conjuntos de itens candidatos; destes apenas 124 conjuntos tornam-se freqüentes, o que resultou em uma taxa de eficiência de 0,03%. Já no algoritmo MiRABIT foram gerados 2493 conjuntos de itens dos quais 124 tornam-se freqüentes, o que resulta em 4,97% dos itens gerados tornaram-se freqüentes. Na terceira passagem percebe-se uma eficiência de 73% nos conjuntos de itens gerados, e na quarta passagem a eficiência é de 100%. No entanto o tempo gasto para obter esta grande eficiência aproxima-se, ou até supera o tempo do *Apriori* nestas passagens.

Conforme apresentado na tabela 5.1 verifica-se que o conjunto de itens candidatos gerados no algoritmo *Apriori*, que é gerada pela função *Apriori-gen*, é bem maior que no algoritmo Mirabit. Esta geração de um conjuntos grande de itens candidatos fez com que o tempo de execução do algoritmo *Apriori* também fosse elevado. O algoritmo Mirabit conseguiu fazer uma geração bem mais eficiente de conjuntos de itens candidatos.

Neste experimento, o algoritmo Mirabit chegou ao mesmo resultado que o algoritmo *Apriori*, ou seja, gerou o mesmo conjunto de regras de associação, no entanto, em 38% do tempo e gerando um conjunto de itens candidatos com 0,38% do tamanho do conjunto gerado pelo *Apriori*. Em virtude de os conjuntos de itens freqüentes gerados pelos dois algoritmos serem iguais, as regras geradas pelos dois algoritmos também foram iguais.

5.2 Experimento 2

No experimento 2 foi analisado o mesmo banco de dados do experimento 1 alterando-se somente os parâmetros de processamento.

5.2.1 Mineração

Neste experimento os valores para os parâmetros foram os seguintes:

Suporte mínimo: 3
Confiança mínima: 10%
Aplicar restrições: Processamento
Tamanho máximo da regra: Sem limite
Processamento seqüencial: Sim

5.2.2 Apresentação e discussão dos resultados

TABELA 5.3 - Comparação tempo de execução dos algoritmos

	Algoritmo *Apriori* [AGR93]		Algoritmo *Mirabit*		
Número da Passagem	**Tempo (s)**	**Conjuntos de itens candidatos gerados**	**Tempo (s)**	**Conjuntos de itens candidatos encontrados**	**Conjuntos de itens freqüentes encontrados**
1	6		6		460
2	37	105570	21	1418	24
3	6	2024	7	1	1
Total	**49**	**107594**	**34**	**1419**	**485**

TABELA 5.4 - Conjunto de regras de associação encontradas

Número da Passagem	**Tempo (s)**	**Regras encontradas**
1	2	45
2	0	3
Total	**2**	**48**

As tabelas 5.3 e 5.4 apresentam os resultados obtidos no experimento 2. Foram encontrados durante a primeira passagem 460 itens freqüentes. O algoritmo *Apriori* [AGR93], no final da primeira passagem, fez a combinação destes 460 itens freqüentes encontrados, gerando os conjuntos de itens candidatos para a segunda passagem. Através

de análise combinatória tem-se que $C_{n,p} = (n!)/(p!* (n-p)!)$ ou seja, (460!) / (2! * (460-2)!) = 460! / (2! * 458!) = 105570. Logo, foram gerados 105570 conjuntos de itens candidatos, destes apenas 24 conjuntos tornaram-se freqüentes, o que resultou em uma taxa de eficiência de 0,02%. Já no algoritmo MiRABIT foram gerados 1418 conjuntos de itens dos quais 24 tornaram-se freqüentes, o que resultou em 1,69% dos itens gerados tornaram-se freqüentes. Na terceira passagem a eficiência é de 100%, no entanto o tempo do algoritmo Mirabit é superior ao tempo do *Apriori*. A existência de somente um conjunto de itens candidato na terceira passagem faz com que o algoritmo finalize as passagens sobre o banco de dados. Este fato influencia o tempo dos dois algoritmos pois fazem uma passagem a menos no banco de dados em relação ao experimento anterior.

Conforme apresentado na tabela 5.3 verifica-se que o conjunto de itens candidatos gerados no algoritmo *Apriori*, que é gerada pela função *Apriori-gen*, é bem maior que no algoritmo Mirabit. Esta geração de um conjuntos grande de itens candidatos faz com que o tempo de execução do algoritmo *Apriori* também seja elevado. O algoritmo Mirabit consegue fazer uma geração bem mais eficiente de conjuntos de itens candidatos.

Como ocorreu no experimento 1, ambos algoritmos chegaram ao mesmo resultado final, mas o Mirabit chegou ao resultado em 69% do tempo do *Apriori* e gerou um conjunto de itens candidatos com 1,31% do tamanho do conjunto gerado pelo *Apriori*. Em virtude de os conjuntos de itens freqüentes gerados pelos dois algoritmos serem iguais, as regras geradas pelos dois algoritmos também foram iguais.

5.3 Experimento 3

No experimento 3 foi analisado o mesmo banco de dados do experimento 1 alterando-se somente os parâmetros de processamento.

5.3.1 Mineração

Neste experimento os valores para os parâmetros foram os seguintes:

Suporte mínimo: 4
Confiança mínima: 10%
Aplicar restrições: Processamento
Tamanho máximo da regra: Sem limite
Processamento seqüencial: Sim

5.3.2 Apresentação e discussão dos resultados

TABELA 5.5 - Comparação tempo de execução dos algoritmos

Algoritmo *Apriori* [AGR93]	Algoritmo *Mirabit*

Número da Passagem	Tempo (s)	Conjuntos de itens candidatos gerados	Tempo (s)	Conjuntos de itens candidatos encontrados	Conjuntos de itens freqüentes encontrados
1	5		5		281
2	20	39340	13	830	4
3	5	4	6	0	0
Total	**30**	**39344**	**24**	**830**	**285**

TABELA 5.6 - Conjunto de regras de associação encontradas

Número da Passagem	Tempo (s)	Regras encontradas
1	1	16
Total	**1**	**16**

As tabelas 5.3 e 5.4 apresentam os resultados obtidos no experimento 2. Foram encontrados durante a primeira passagem 281 itens freqüentes. O algoritmo *Apriori* [AGR93], no final da primeira passagem, fez a combinação destes 281 itens freqüentes encontrados, gerando os conjuntos de itens candidatos para a segunda passagem. Através de análise combinatória tem-se que $C_{n,p} = (n!)/(p!* (n-p)!)$ ou seja, (281!) / (2! * (281-2)!) = 281! / (2! * 279!) = 39340. Logo, foram gerados 39340 conjuntos de itens candidatos; destes apenas 4 conjuntos tornam-se freqüentes, o que resultou em uma taxa de eficiência de 0,01%. Já no algoritmo MiRABIT foram gerados 830 conjuntos de itens dos quais 4 tornam-se freqüentes, o que resulta em 0,48% dos itens gerados tornaram-se freqüentes. Na terceira passagem não foram encontrados conjuntos de itens candidatos. No final da terceira passagem sobre o banco de dados não são encontrados conjuntos de itens candidatos fazendo com que esta passagem sobre o banco de dados não obtivesse nenhum resultado.

Conforme apresentado na tabela 5.5 verifica-se que o conjunto de itens candidatos gerados no algoritmo *Apriori*, que é gerada pela função *Apriori-gen*, é bem maior que no algoritmo Mirabit. Esta geração de um conjuntos grande de itens candidatos fez com que o tempo de execução do algoritmo *Apriori* também fosse elevado. O algoritmo Mirabit conseguiu fazer uma geração bem mais eficiente de conjuntos de itens candidatos.

Como ocorreu nos experimentos 1 e 2, o algoritmo Mirabit chegou ao mesmo resultado que o algoritmo *Apriori*, ou seja, gerou o mesmo conjunto de regras de associação, no entanto, em 80% do tempo e gerando um conjunto de itens candidatos com 2,10% do tamanho do conjunto gerado pelo *Apriori*. Em virtude de os conjuntos de itens freqüentes gerados pelos dois algoritmos serem iguais, as regras geradas pelos dois algoritmos também foram iguais.

5.4 Experimento 4

No experimento 4 foi analisado o mesmo banco de dados do experimento 1 alterando-se somente os parâmetros de processamento.

5.4.1 Mineração

Neste experimento os valores para os parâmetros foram os seguintes:

Suporte mínimo: 5
Confiança mínima: 10%
Aplicar restrições: Processamento
Tamanho máximo da regra: Sem limite
Processamento seqüencial: Sim

5.4.2 Apresentação e discussão dos resultados

TABELA 5.7 - Comparação tempo de execução dos algoritmos

	Algoritmo *Apriori* [AGR93]		Algoritmo *Mirabit*		
Número da Passagem	**Tempo (s)**	**Conjuntos de itens candidatos gerados**	**Tempo (s)**	**Conjuntos de itens candidatos encontrados**	**Conjuntos de itens freqüentes encontrados**
1	3		3		188
2	20	17578	12	545	4
3	5	4	6	0	0
Total	**28**	**17582**	**21**	**545**	**192**

TABELA 5.8 - Conjunto de regras de associação encontradas

Número da Passagem	**Tempo (s)**	**Regras encontradas**
1	1	8
Total	**1**	**8**

As tabelas 5.7 e 5.8 apresentam os resultados obtidos no experimento 4. Foram encontrados durante a primeira passagem 188 itens freqüentes. O algoritmo *Apriori* [AGR93], no final da primeira passagem, fez a combinação destes 188 itens freqüentes encontrados, gerando os conjuntos de itens candidatos para a segunda passagem. Através de análise combinatória tem-se que $C_{n,p} = (n!)/(p!* (n-p)!)$ ou seja, (188!) / (2! * (188-2)!) = 188! / (2! * 186!) = 17578. Logo, foram gerados 17578 conjuntos de itens candidatos; destes apenas 4 conjuntos tornam-se freqüentes, o que resultou em uma taxa de eficiência de 0,02%. Já no algoritmo MiRABIT foram gerados 545 conjuntos de itens dos quais 4 tornam-se freqüentes, o que resulta em 0,88% dos itens gerados tornaram-se freqüentes. Na terceira passagem não foram encontrados conjuntos de itens candidatos.

Conforme apresentado na tabela 5.7 verifica-se que o conjunto de itens candidatos gerados no algoritmo *Apriori*, que é gerada pela função *Apriori-gen*, é bem maior que no

algoritmo Mirabit. Esta geração de um conjuntos grande de itens candidatos fez com que o tempo de execução do algoritmo *Apriori* também fosse elevado. O algoritmo Mirabit conseguiu fazer uma geração bem mais eficiente de conjuntos de itens candidatos.

Neste experimento, o algoritmo Mirabit chegou ao mesmo resultado que o algoritmo *Apriori*, ou seja, gerou o mesmo conjunto de regras de associação, no entanto, em 38% do tempo e gerando um conjunto de itens candidatos com 0,38% do tamanho do conjunto gerado pelo *Apriori*. Em virtude de os conjuntos de itens freqüentes gerados pelos dois algoritmos serem iguais, as regras geradas pelos dois algoritmos também foram iguais.

5.5 Experimento 5

No experimento 5 foi analisado um banco de dados de uma empresa do comércio varejista de ferragens de Bagé. O banco de dados analisado possui 3133 transações, 6457 itens de transação, e 11843 produtos com um mês de movimento de vendas da empresa.

5.5.1 Pré processamento

O primeiro passo do pré-processamento do banco de dados analisado foi a seleção de um determinado período de vendas para ser minerado pelo protótipo. O subconjunto selecionado consistia das transações efetuadas no período de 01 de outubro de 2000 a 31 de outubro de 2000, totalizando um conjunto de 3133 transações e 6457 itens processados. Não foram encontrados problemas de integridade e completude nos dados.

5.5.2 Mineração

Antes de começar a mineração o usuário deve informar um conjunto de parâmetros para o processo, que são: suporte mínimo, confiança mínima, aplicar restrições, tamanho máximo da regra e processamento seqüencial.

Neste experimento os valores para os parâmetros foram os seguintes:

Suporte mínimo: 3
Confiança mínima: 10%
Aplicar restrições: Processamento
Tamanho máximo da regra: Sem limite
Processamento seqüencial: Sim

5.5.3 Apresentação e discussão dos resultados

TABELA 5.9 - Comparação tempo de execução dos algoritmos

Número da Passagem	Algoritmo *Apriori* [AGR93] Tempo (s)	Conjuntos de itens candidatos gerados	Algoritmo *Mirabit* Tempo (s)	Conjuntos de itens candidatos encontrados	Conjuntos de itens freqüentes encontrados
1	7		7		543
2	124	147153	57	4148	202
3	258	1353400	22	77	46
4	38	163185	62	9	9
5	130	126	224	1	0
Total	**557**	**1516711**	**372**	**4235**	**800**

TABELA 5.10 - Conjunto de regras de associação encontradas

Número da Passagem	Tempo (s)	Regras encontradas
1	4	319
2	3	241
3	2	122
4	0	0
Total	**8**	**682**

As tabelas 5.9 e 5.10 apresentam os resultados obtidos no experimento 5. Foram encontrados durante a primeira passagem 543 itens freqüentes. O algoritmo *Apriori* [AGR93], no final da primeira passagem, fez a combinação destes 543 itens freqüentes encontrados, gerando os conjuntos de itens candidatos para a segunda passagem. Através de análise combinatória tem-se que $C_{n,p} = (n!)/(p!* (n-p)!)$ ou seja, (543!) / (2! * (543-2)!) = 543! / (2! * 541!) = 147153. Logo, foram gerados 147153 conjuntos de itens candidatos; destes apenas 202 conjuntos tornam-se freqüentes, o que resultou em uma taxa de eficiência de 0,1372%. Já no algoritmo MiRABIT foram gerados 4148 conjuntos de itens dos quais 202 tornam-se freqüentes, o que resulta em 4,86% dos itens gerados tornaram-se freqüentes. Na terceira passagem não foram encontrados conjuntos de itens candidatos.

Conforme apresentado na tabela 5.9 verifica-se que o conjunto de itens candidatos gerados no algoritmo *Apriori*, que é gerada pela função *Apriori-gen*, é bem maior que no algoritmo Mirabit. Esta geração de um conjuntos grande de itens candidatos fez com que o tempo de execução do algoritmo *Apriori* também fosse elevado. O algoritmo Mirabit conseguiu fazer uma geração bem mais eficiente de conjuntos de itens candidatos.

Neste experimento, o algoritmo Mirabit chegou ao mesmo resultado que o algoritmo *Apriori*, ou seja, gerou o mesmo conjunto de regras de associação, no entanto, em 66% do tempo e gerando um conjunto de itens candidatos com 0,27% do tamanho do conjunto gerado pelo *Apriori*. Em virtude de os conjuntos de itens freqüentes gerados pelos dois algoritmos serem iguais, as regras geradas pelos dois algoritmos também foram iguais.

5.6 Experimento 6

No experimento 6 foi analisado o mesmo banco de dados do experimento 5 alterando-se somente os parâmetros de processamento.

5.6.1 Mineração

Antes de começar a mineração o usuário deve informar um conjunto de parâmetros para o processo, que são: suporte mínimo, confiança mínima, aplicar restrições, tamanho máximo da regra e processamento seqüencial.

Neste experimento os valores para os parâmetros foram os seguintes:

Suporte mínimo: 4
Confiança mínima: 10%
Aplicar restrições: Processamento
Tamanho máximo da regra: Sem limite
Processamento seqüencial: Sim

5.6.2 Apresentação e discussão dos resultados

TABELA 5.11 - Comparação tempo de execução dos algoritmos

	Algoritmo *Apriori* [AGR93]		Algoritmo *Mirabit*		
Número da Passagem	**Tempo (s)**	**Conjuntos de itens candidatos gerados**	**Tempo (s)**	**Conjuntos de itens candidatos encontrados**	**Conjuntos de itens freqüentes encontrados**
1	6		6		397
2	65	78606	46	3147	107
3	83	198485	20	37	16
4	56	1820	62	1	1
Total	**210**	**278911**	**134**	**3185**	**521**

TABELA 5.12 - Conjunto de regras de associação encontradas

Número da Passagem	**Tempo (s)**	**Regras encontradas**
1	3	179
2	1	88
3	0	0
Total	**4**	**267**

As tabelas 5.11 e 5.12 apresentam os resultados obtidos no experimento 6. Foram encontrados durante a primeira passagem 397 itens freqüentes. O algoritmo *Apriori* [AGR93], no final da primeira passagem, fez a combinação destes 397 itens freqüentes encontrados, gerando os conjuntos de itens candidatos para a segunda passagem. Através

de análise combinatória tem-se que $C_{n,p} = (n!)/(p!* (n-p)!)$ ou seja, (397!) / (2! * (397-2)!) = 397! / (2! * 395!) = 78606. Logo, foram gerados 78606 conjuntos de itens candidatos; destes apenas 107 conjuntos tornam-se freqüentes, o que resultou em uma taxa de eficiência de 0,1361%. Já no algoritmo MiRABIT foram gerados 3147 conjuntos de itens dos quais 107 tornam-se freqüentes, o que resulta em 3,40% dos itens gerados tornaram-se freqüentes. Na terceira passagem não foram encontrados conjuntos de itens candidatos.

Conforme apresentado na tabela 5.11 verifica-se que o conjunto de itens candidatos gerados no algoritmo *Apriori*, que é gerada pela função *Apriori-gen*, é bem maior que no algoritmo Mirabit. Esta geração de um conjuntos grande de itens candidatos fez com que o tempo de execução do algoritmo *Apriori* também fosse elevado. O algoritmo Mirabit conseguiu fazer uma geração bem mais eficiente de conjuntos de itens candidatos.

Neste experimento, o algoritmo Mirabit chegou ao mesmo resultado que o algoritmo *Apriori*, ou seja, gerou o mesmo conjunto de regras de associação, no entanto, em 63% do tempo e gerando um conjunto de itens candidatos com 1,14% do tamanho do conjunto gerado pelo *Apriori*. Em virtude de os conjuntos de itens freqüentes gerados pelos dois algoritmos serem iguais, as regras geradas pelos dois algoritmos também foram iguais.

5.7 Experimento 7

No experimento 7 foi analisado o mesmo banco de dados do experimento 5 alterando-se somente os parâmetros de processamento.

5.7.1 Mineração

Antes de começar a mineração o usuário deve informar um conjunto de parâmetros para o processo, que são: suporte mínimo, confiança mínima, aplicar restrições, tamanho máximo da regra e processamento seqüencial.

Neste experimento os valores para os parâmetros foram os seguintes:

Suporte mínimo: 5
Confiança mínima: 10%
Aplicar restrições: Processamento
Tamanho máximo da regra: Sem limite
Processamento seqüencial: Sim

5.7.2 Apresentação e discussão dos resultados

TABELA 5.13 - Comparação tempo de execução dos algoritmos

Número da Passagem	Algoritmo *Apriori* [AGR93] Tempo (s)	Conjuntos de itens candidatos gerados	Algoritmo *Mirabit* Tempo (s)	Conjuntos de itens candidatos encontrados	Conjuntos de itens freqüentes encontrados
1	6		6		304
2	75	46056	39	2427	61
3	45	35990	19	13	5
4	40	5	63	0	0
Total	**166**	**82051**	**127**	**2440**	**370**

TABELA 5.14 - Conjunto de regras de associação encontradas

Número da Passagem	Tempo (s)	Regras encontradas
1	1	102
2	0	25
Total	**1**	**127**

As tabelas 5.13 e 5.14 apresentam os resultados obtidos no experimento 7. Foram encontrados durante a primeira passagem 304 itens freqüentes. O algoritmo *Apriori* [AGR93], no final da primeira passagem, fez a combinação destes 304 itens freqüentes encontrados, gerando os conjuntos de itens candidatos para a segunda passagem. Através de análise combinatória tem-se que $C_{n,p} = (n!)/(p!* (n-p)!)$ ou seja, (304!) / (2! * (304-2)!) = 304! / (2! * 302!) = 46056. Logo, foram gerados 46056 conjuntos de itens candidatos; destes apenas 61 conjuntos tornam-se freqüentes, o que resultou em uma taxa de eficiência de 0,1324%. Já no algoritmo MiRABIT foram gerados 2427 conjuntos de itens dos quais 61 tornam-se freqüentes, o que resulta em 2,51% dos itens gerados tornaram-se freqüentes. Na terceira passagem não foram encontrados conjuntos de itens candidatos.

Conforme apresentado na tabela 5.13 verifica-se que o conjunto de itens candidatos gerados no algoritmo *Apriori*, que é gerada pela função *Apriori-gen*, é bem maior que no algoritmo Mirabit. Esta geração de um conjuntos grande de itens candidatos fez com que o tempo de execução do algoritmo *Apriori* também fosse elevado. O algoritmo Mirabit conseguiu fazer uma geração bem mais eficiente de conjuntos de itens candidatos.

Neste experimento, o algoritmo Mirabit chegou ao mesmo resultado que o algoritmo *Apriori*, ou seja, gerou o mesmo conjunto de regras de associação, no entanto, em 76% do tempo e gerando um conjunto de itens candidatos com 2,97% do tamanho do conjunto gerado pelo *Apriori*. Em virtude de os conjuntos de itens freqüentes gerados pelos dois algoritmos serem iguais, as regras geradas pelos dois algoritmos também foram iguais.

5.8 Experimento 8

No experimento 8 foi analisado o mesmo banco de dados do experimento 5 alterando-se somente os parâmetros de processamento.

5.8.1 Mineração

Antes de começar a mineração o usuário deve informar um conjunto de parâmetros para o processo, que são: suporte mínimo, confiança mínima, aplicar restrições, tamanho máximo da regra e processamento seqüencial.

Neste experimento os valores para os parâmetros foram os seguintes:

Suporte mínimo: 6
Confiança mínima: 10%
Aplicar restrições: Processamento
Tamanho máximo da regra: Sem limite
Processamento seqüencial: Sim

5.8.2 Apresentação e discussão dos resultados

TABELA 5.15 - Comparação tempo de execução dos algoritmos

	Algoritmo *Apriori* [AGR93]		Algoritmo *Mirabit*		
Número da Passagem	**Tempo (s)**	**Conjuntos de itens candidatos gerados**	**Tempo (s)**	**Conjuntos de itens candidatos encontrados**	**Conjuntos de itens freqüentes encontrados**
1	4		4		230
2	53	26335	34	1487	44
3	32	13244	19	7	1
Total	**89**	**39579**	**57**	**1494**	**275**

TABELA 5.16 - Conjunto de regras de associação encontradas

Número da Passagem	**Tempo (s)**	**Regras encontradas**
1	1	72
2	0	4
Total	**1**	**76**

As tabelas 5.15 e 5.16 apresentam os resultados obtidos no experimento 8. Foram encontrados durante a primeira passagem 230 itens freqüentes. O algoritmo *Apriori* [AGR93], no final da primeira passagem, fez a combinação destes 230 itens freqüentes encontrados, gerando os conjuntos de itens candidatos para a segunda passagem. Através de análise combinatória tem-se que $C_{n,p} = (n!)/(p!* (n-p)!)$ ou seja, (230!) / (2! * (230-2)!) = 230! / (2! * 228!) = 26335. Logo, foram gerados 26335 conjuntos de itens candidatos; destes apenas 44 conjuntos tornam-se freqüentes, o que resultou em uma taxa de eficiência

de 0,1670%. Já no algoritmo MiRABIT foram gerados 1487 conjuntos de itens dos quais 44 tornam-se freqüentes, o que resulta em 2,95% dos itens gerados tornaram-se freqüentes. Na terceira passagem não foram encontrados conjuntos de itens candidatos.

Conforme apresentado na tabela 5.15 verifica-se que o conjunto de itens candidatos gerados no algoritmo *Apriori*, que é gerada pela função *Apriori-gen*, é bem maior que no algoritmo Mirabit. Esta geração de um conjuntos grande de itens candidatos fez com que o tempo de execução do algoritmo *Apriori* também fosse elevado. O algoritmo Mirabit conseguiu fazer uma geração bem mais eficiente de conjuntos de itens candidatos.

Neste experimento, o algoritmo Mirabit chegou ao mesmo resultado que o algoritmo *Apriori*, ou seja, gerou o mesmo conjunto de regras de associação, no entanto, em 64% do tempo e gerando um conjunto de itens candidatos com 3,77% do tamanho do conjunto gerado pelo *Apriori*. Em virtude de os conjuntos de itens freqüentes gerados pelos dois algoritmos serem iguais, as regras geradas pelos dois algoritmos também foram iguais.

5.9 Experimento 9

No experimento 9 foi analisado o mesmo banco de dados do experimento 5 alterando-se somente os parâmetros de processamento.

5.9.1 Mineração

Antes de começar a mineração o usuário deve informar um conjunto de parâmetros para o processo, que são: suporte mínimo, confiança mínima, aplicar restrições, tamanho máximo da regra e processamento seqüencial.

Neste experimento os valores para os parâmetros foram os seguintes:

Suporte mínimo: 7
Confiança mínima: 10%
Aplicar restrições: Processamento
Tamanho máximo da regra: Sem limite
Processamento seqüencial: Sim

5.9.2 Apresentação e discussão dos resultados

TABELA 5.17 - Comparação tempo de execução dos algoritmos

	Algoritmo *Apriori* [AGR93]		Algoritmo *Mirabit*		
Número da Passagem	**Tempo (s)**	**Conjuntos de itens candidatos**	**Tempo (s)**	**Conjuntos de itens candidatos**	**Conjuntos de itens freqüentes**

		gerados		encontrados	encontrados
1	4		4		189
2	41	17766	29	1483	31
3	32	4495	18	4	1
Total	**77**	**22261**	**51**	**1487**	**221**

TABELA 5.18 - Conjunto de regras de associação encontradas

Número da Passagem	Tempo (s)	Regras encontradas
1	1	51
2	0	4
Total	**1**	**55**

As tabelas 5.17 e 5.18 apresentam os resultados obtidos no experimento 9. Foram encontrados durante a primeira passagem 189 itens freqüentes. O algoritmo *Apriori* [AGR93], no final da primeira passagem, fez a combinação destes 189 itens freqüentes encontrados, gerando os conjuntos de itens candidatos para a segunda passagem. Através de análise combinatória tem-se que $C_{n,p} = (n!)/(p!* (n-p)!)$ ou seja, (189!) / (2! * (189-2)!) = 189! / (2! * 187!) = 17766. Logo, foram gerados 17766 conjuntos de itens candidatos; destes apenas 31 conjuntos tornam-se freqüentes, o que resultou em uma taxa de eficiência de 0,1744%. Já no algoritmo MiRABIT foram gerados 1483 conjuntos de itens dos quais 31 tornam-se freqüentes, o que resulta em 2,09% dos itens gerados tornaram-se freqüentes. Na terceira passagem não foram encontrados conjuntos de itens candidatos.

Conforme apresentado na tabela 5.17 verifica-se que o conjunto de itens candidatos gerados no algoritmo *Apriori*, que é gerada pela função *Apriori-gen*, é bem maior que no algoritmo Mirabit. Esta geração de um conjuntos grande de itens candidatos fez com que o tempo de execução do algoritmo *Apriori* também fosse elevado. O algoritmo Mirabit conseguiu fazer uma geração bem mais eficiente de conjuntos de itens candidatos.

Neste experimento, o algoritmo Mirabit chegou ao mesmo resultado que o algoritmo *Apriori*, ou seja, gerou o mesmo conjunto de regras de associação, no entanto, em 66% do tempo e gerando um conjunto de itens candidatos com 6,67% do tamanho do conjunto gerado pelo *Apriori*. Em virtude de os conjuntos de itens freqüentes gerados pelos dois algoritmos serem iguais, as regras geradas pelos dois algoritmos também foram iguais.

5.10 Análise comparativa de desempenho

A seguir é apresentada uma tabela com o resumo dos resultados obtidos nos nove experimentos executados.

TABELA 5.19 – Análise comparativa de desempenho

Ramo de	Suporte	*Apriori*	MiRABIT	Ganho

Negócio do Banco de Dados	Mínimo	Tempo (s)	Conjuntos de itens candidatos	Tempo (s)	Conjuntos de itens candidatos	Tempo %	Espaço %
	2	130	648707	50	2509	61,53	99,61
confecção	3	49	107594	34	1419	30,61	98,68
T_m=1,93	4	30	39344	24	830	20	97,89
	5	28	17582	21	545	25	96,90
	3	557	1516711	372	4235	33,21	99,72
ferragem	4	210	278911	134	3185	36,19	98,85
T_m=2,06	5	166	82051	127	2440	23,49	97,02
	6	89	39579	57	1494	35,95	96,22
	7	77	22261	51	1487	33,76	93,32
Ganho médio em complexidade						33,30	97,57

Tem-se que T_m significa o tamanho médio das transações em cada um dos bancos de dados.

As duas últimas colunas demonstram o ganho de desempenho obtido nas duas dimensões de complexidade analisadas: tempo e espaço. O ganho relativo à complexidade de tempo denota a quantidade de tempo ganha com a utilização do algoritmo MiRABIT em relação ao algoritmo *Apriori*. O ganho relativo à complexidade de espaço denota o quanto menor foi a quantidade de conjuntos de itens candidatos gerada pelo algoritmo MiRABIT em relação ao algoritmo *Apriori*.

Conforme resultados apresentados na tabela 5.19, o algoritmo MiRABIT obteve um ganho médio de tempo de 33,30% em relação ao algoritmo *Apriori* para executar a tarefa de mineração de regras de associação. Em relação ao espaço, o conjunto de itens candidatos gerado pelo algoritmo MiRABIT foi, em média, 97,57% menor do que o conjunto de itens candidatos gerado pelo algoritmo *Apriori*. O algoritmo Mirabit, por gerar um conjunto de itens candidatos menor que o algoritmo *Apriori*, necessita de requisitos de hardware mais modestos para executar a tarefa de mineração de regras de associação. Também é importante salientar que, conforme demonstrado nos experimentos, o resultado final foi exatamente o mesmo, ou seja, ambos algoritmos geraram o mesmo conjunto de regras de associação por obterem o mesmo conjunto de itens freqüentes.

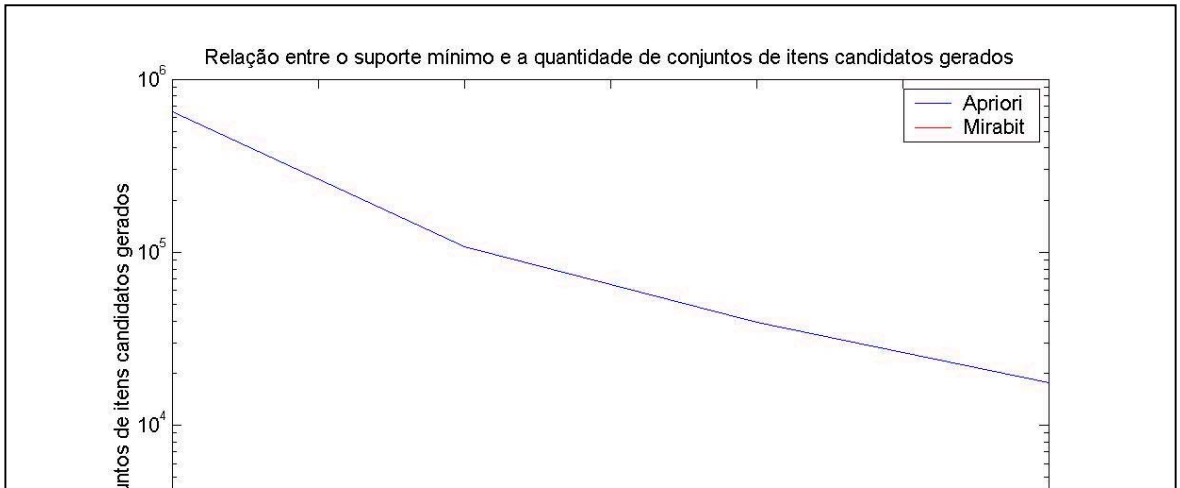


FIGURA 5.1 – Gráfico da quantidade de itens candidatos gerados pelos algoritmos *Apriori* e Mirabit

Conforme demonstrado na figura 2.2, à medida que aumenta o suporte mínimo o tamanho do conjunto de itens candidatos gerados pelo algoritmo *Apriori* diminui. O gráfico da figura 5.1 é um gráfico com valores em forma logarítmica. Conforme pode ser observado na figura 5.1 a diminuição do tamanho do conjunto de itens candidatos gerados pelo algoritmo *Apriori* diminui de forma exponencial, enquanto que com o algoritmo Mirabit a diminuição do número de conjuntos de itens candidatos é de forma quase linear. A diminuição do número de conjunto de itens candidatos do algoritmo *Apriori* se dá de forma exponencial porque a função *Apriori-gen* gera todas as combinações possíveis entre conjuntos de itens através de análise combinatorial. Como o algoritmo Mirabit gera mais eficientemente os conjuntos de itens candidatos que o algoritmo *Apriori*, a quantidade destes conjuntos de itens tende a diminuir de forma quase linear. Por este motivo, à medida que o suporte mínimo aumenta o ganho de espaço obtido pelo algoritmo Mirabit tende a desaparecer. À medida que o suporte mínimo diminui o número de passagens sobre o banco de dados tende para 1. Como a primeira passagem sobre o banco de dados de ambos os algoritmos é idêntica, a medida que o suporte mínimo aumenta o tempo de execução de ambos os algoritmos tenderá a ser o mesmo.

Como pode ser visto na tabela 5.17 o ganho de espaço diminui gradativamente à medida que o suporte mínimo aumenta. Isto já não ocorre com o ganho de tempo pois esta medida depende da quantidade de conjuntos de itens freqüentes encontrados na passagem.

Também pode ser verificado pela tabela 5.17 que o ganho de tempo não é diretamente proporcional ao ganho de espaço. Este fato ocorre por haver um *overhead* para que seja feita a geração mais eficiente de conjuntos de itens candidatos. Como pode ser verificado na discussão individual do resultado de cada experimento, este *overhead* é menor nas passagens iniciais do algoritmo sobre o banco de dados e tende a aumentar à medida que aumenta o número da passagem sobre o banco de dados.

6 Conclusões e trabalhos futuros

Neste trabalho foram apresentados os fundamentos teóricos da descoberta de regras de associação em bancos de dados. Também foram apresentados alguns algoritmos introduzidos por diversos autores utilizando diferentes técnicas para descobrirem regras de associação dentro dos bancos de dados, tais técnicas objetivam principalmente a diminuição do tempo de execução do algoritmo.

Dentre os algoritmos estudados, constatou-se uma preocupação dirigida à primeira fase da mineração de regras de associação, que é a parte mais onerosa processamento. Nesta fase, o algoritmo executa múltiplas passagens sobre o banco de dados e uma das principais características dos algoritmos estudados é que no início cada passagem são gerados todos os seus possíveis conjuntos de itens candidatos. Através da realização de alguns experimentos, detectamos que, nesta primeira fase, um dos passos mais onerosos em termos de tempo de processamento é a identificação prévia dos conjuntos de itens candidatos. Foram realizados experimentos para avaliação de desempenho onde constatou-se que a geração de conjuntos de itens candidatos pode ser otimizada, se executada a cada transação ao invés de ser executada no início da passagem. Esta alteração aumenta o desempenho do processo de mineração de regras de associação em bancos de dados com uma baixa média de itens por transação. Também foi observado, dentre os trabalhos pesquisados, uma preocupação dirigida ao ramo de supermercados, não sendo encontrada na literatura pesquisas dirigidas a outros tipos de atividade. Esta constatação nos motivou a propor um algoritmo dirigido para a descoberta de regras de associação aplicado ao comércio varejista de confecção, que tem como principal característica a não geração de conjuntos de itens candidatos, diminuindo assim o tempo de processamento. Foram realizados testes de desempenho onde o algoritmo proposto demonstrou sua maior eficiência em relação ao algoritmo *Apriori* quando aplicados ao tipo de problema em questão.

Embora o algoritmo *Mirabit* também possa ser aplicado em ambientes onde a quantidade média de itens por transação seja alta, o algoritmo possui um melhor desempenho no ambiente em estudo e em ambientes com uma quantidade média de itens por transação baixa.

Como trabalhos futuros, pretende-se ampliar as funcionalidades da ferramenta com a inclusão de módulos para mineração de padrões seqüenciais e clusterização. Isto irá permitir que a ferramenta consiga resolver um espectro maior de problemas, além de permitir o processamento de arquivos de log de servidores web para identificação dos perfis de comportamento de usuários de um servidor. Como foi constatado durante o desenvolvimento deste trabalho que o algoritmo Mirabit é mais eficiente que o *Apriori* em bancos de dados com baixa média de itens por transação, buscar-se-á fazer com que a ferramenta, após análise do tamanho médio das transações do banco de dados, selecione autonomamente o algoritmo para minerar o banco de dados, executando a tarefa de mineração sempre com o menor tempo possível.

Anexo Regras de Associação

TABELA 1 - Regras de associação encontradas antes do processo de poda

Regra	% Confiança	Melhora
1=>2 | 5	71	-6
2=>1 | 5	83	17
1=>3 | 3	42	-35
3=>1 | 3	75	31
1=>4 | 4	57	-20
4=>1 | 4	80	25
1=>5 | 1	14	-63
5=>1 | 1	50	28
2=>3 | 2	33	-33
3=>2 | 2	50	6
2=>4 | 5	83	17
4=>2 | 5	100	45
2=>5 | 1	16	-50
5=>2 | 1	50	28
3=>4 | 2	50	6
4=>3 | 2	40	-15
3=>5 | 2	50	6
5=>3 | 2	100	78
4=>5 | 1	20	-35
5=>4 | 1	50	28
1,2=>3 | 2	40	-2
1,3=>2 | 2	66	-5
2,3=>1 | 2	100	17
1=>2,3 | 2	28	6
2=>1,3 | 2	33	0
3=>1,2 | 2	50	-5
1,2=>4 | 4	80	38
1,4=>2 | 4	100	0
2,4=>1 | 4	80	-3
1=>2,4 | 4	57	2
2=>1,4 | 4	66	22
4=>1,2 | 4	80	25
1,2=>5 | 1	20	-22
1,5=>2 | 1	100	29
2,5=>1 | 1	100	27
1=>2,5 | 1	14	3
2=>1,5 | 1	16	5
5=>1,2 | 1	50	-5
1,3=>4 | 2	66	9
1,4=>3 | 2	50	8
3,4=>1 | 2	100	20

1=>3,4 | 2	28	6
3=>1,4 | 2	50	6
4=>1,3 | 2	40	7
1,3=>5 | 1	33	-17
1,5=>3 | 1	100	0
3,5=>1 | 1	50	-25
1=>3,5 | 1	14	-8
3=>1,5 | 1	25	14
5=>1,3 | 1	50	17
1,4=>5 | 1	25	5
1,5=>4 | 1	100	43
4,5=>1 | 1	100	20
1=>4,5 | 1	14	3
4=>1,5 | 1	20	9
5=>1,4 | 1	50	6
2,3=>4 | 2	100	17
2,4=>3 | 2	40	0
3,4=>2 | 2	100	0
2=>3,4 | 2	33	11
3=>2,4 | 2	50	-5
4=>2,3 | 2	40	18
2,3=>5 | 1	50	0
2,5=>3 | 1	100	0
3,5=>2 | 1	50	0
2=>3,5 | 1	16	-6
3=>2,5 | 1	25	14
5=>2,3 | 1	50	28
2,4=>5 | 1	20	0
2,5=>4 | 1	100	17
4,5=>2 | 1	100	0
2=>4,5 | 1	16	5
4=>2,5 | 1	20	9
5=>2,4 | 1	50	-5
3,4=>5 | 1	50	0
3,5=>4 | 1	50	0
4,5=>3 | 1	100	0
3=>4,5 | 1	25	14
4=>3,5 | 1	20	-2
5=>3,4 | 1	50	28
1,2,3=>4 | 2	100	0
1,2,4=>3 | 2	50	0
1,3,4=>2 | 2	100	0
2,3,4=>1 | 2	100	0
1,2=>3,4 | 2	40	7
1,3=>2,4 | 2	66	9
1,4=>2,3 | 2	50	10

2,3=>1,4 | 2	100	34
2,4=>1,3 | 2	40	0
3,4=>1,2 | 2	100	20
1=>2,3,4 | 2	28	-49
2=>1,3,4 | 2	33	-33
3=>1,2,4 | 2	50	6
4=>1,2,3 | 2	40	-15
1,2,3=>5 | 1	50	0
1,2,5=>3 | 1	100	0
1,3,5=>2 | 1	100	0
2,3,5=>1 | 1	100	0
1,2=>3,5 | 1	20	-2
1,3=>2,5 | 1	33	8
1,5=>2,3 | 1	100	50
2,3=>1,5 | 1	50	25
2,5=>1,3 | 1	100	50
3,5=>1,2 | 1	50	0
1=>2,3,5 | 1	14	3
2=>1,3,5 | 1	16	5
3=>1,2,5 | 1	25	14
5=>1,2,3 | 1	50	28
1,2,4=>5 | 1	25	0
1,2,5=>4 | 1	100	0
1,4,5=>2 | 1	100	0
2,4,5=>1 | 1	100	0
1,2=>4,5 | 1	20	4
1,4=>2,5 | 1	25	5
1,5=>2,4 | 1	100	43
2,4=>1,5 | 1	20	0
2,5=>1,4 | 1	100	34
4,5=>1,2 | 1	100	20
1=>2,4,5 | 1	14	3
2=>1,4,5 | 1	16	5
4=>1,2,5 | 1	20	9
5=>1,2,4 | 1	50	6
1,3,4=>5 | 1	50	0
1,3,5=>4 | 1	100	0
1,4,5=>3 | 1	100	0
3,4,5=>1 | 1	100	0
1,3=>4,5 | 1	33	8
1,4=>3,5 | 1	25	3
1,5=>3,4 | 1	100	50
3,4=>1,5 | 1	50	25
3,5=>1,4 | 1	50	0
4,5=>1,3 | 1	100	50
1=>3,4,5 | 1	14	3

3=>1,4,5 | 1	25	14
4=>1,3,5 | 1	20	9
5=>1,3,4 | 1	50	28
2,3,4=>5 | 1	50	0
2,3,5=>4 | 1	100	0
2,4,5=>3 | 1	100	0
3,4,5=>2 | 1	100	0
2,3=>4,5 | 1	50	25
2,4=>3,5 | 1	20	-2
2,5=>3,4 | 1	100	50
3,4=>2,5 | 1	50	30
3,5=>2,4 | 1	50	0
4,5=>2,3 | 1	100	50
2=>3,4,5 | 1	16	5
3=>2,4,5 | 1	25	14
4=>2,3,5 | 1	20	9
5=>2,3,4 | 1	50	28
1,2,3,4=>5 | 1	50	0
1,2,3,5=>4 | 1	100	0
1,2,4,5=>3 | 1	100	0
1,3,4,5=>2 | 1	100	0
2,3,4,5=>1 | 1	100	0
1,2,3=>4,5 | 1	50	0
1,2,4=>3,5 | 1	25	0
1,2,5=>3,4 | 1	100	0
1,3,4=>2,5 | 1	50	0
1,3,5=>2,4 | 1	100	0
1,4,5=>2,3 | 1	100	0
2,3,4=>1,5 | 1	50	0
2,3,5=>1,4 | 1	100	0
2,4,5=>1,3 | 1	100	0
3,4,5=>1,2 | 1	100	0
1,2=>3,4,5 | 1	20	4
1,3=>2,4,5 | 1	33	8
1,4=>2,3,5 | 1	25	5
1,5=>2,3,4 | 1	100	50
2,3=>1,4,5 | 1	50	25
2,4=>1,3,5 | 1	20	0
2,5=>1,3,4 | 1	100	50
3,4=>1,2,5 | 1	50	25
3,5=>1,2,4 | 1	50	0
4,5=>1,2,3 | 1	100	50
1=>2,3,4,5 | 1	14	3
2=>1,3,4,5 | 1	16	5
3=>1,2,4,5 | 1	25	14
4=>1,2,3,5 | 1	20	9

5=>1,2,3,4 | 1	50	28

Bibliografia

[AGR93] AGRAWAL, Rakesh; IMIELINSKI, Tomasz; SWAMI, Arun. Mining Associations between Sets of Items in Massive Databases. In: ACM SIGMOD INT. CONFERENCE ON MANAGEMENT OF DATA, 1993. **Proceedings**... Washington D.C.: ACM Press, 1993. p. 207-216.

[AGR93a] AGRAWAL, Rakesh; IMIELINSKI, Tomasz; SWAMI, Arun. Database Mining: A Performance Perspective**. IEEE Transactions on Knowledge and Data Engineering, Special issue on Learning and Discovery in Knowledge-Based Databases**, New York, v5, n.6, p.914-925, 1993.

[AGR94] AGRAWAL, Rakesh; SRIKANT, Ramakrishnan. Fast Algorithms for Mining Association Rules. In: ACM VLDB INT. CONFERENCE ON VERY LARGE DATABASES, 20., 1994. **Proceedings**... Hove: Morgan Kaufmann, 1994. p.487-499.

[AGR96] AGRAWAL, Rakesh et al. Fast Discovery of Association Rules. In: ADVANCES in Knowledge Discovery and Data Mining. Menlo Park: AAAI, 1996. p.307-328.

[AGR96a] AGRAWAL, Rakesh SHAFER, John C. Parallel Mining of Association Rules. **IEEE Transactions on Knowledge and Data Engineering**. New York, v.8, n.6, p. 962-962, 1996.

[BAY99] BAYARDO JR, Roberto J.; AGRAWAL, Rakesh; GUNOPULOS, Dimitrios. Constraint-Based Rule Mining in Large, Dense Databases. In: INT. CONFERENCE ON DATA ENGINEERING, 15., 1999, Sidney. **Proceedings**... [S.l.:s.n.], 1999. p. 188-197.

[BAY99a] BAYARDO JR., Roberto J.; AGRAWAL Rakesh. Mining the Most Interesting Rules. In: ACM SIGKDD INT. CONFERENCE ON KNOWLEDGE DISCOVERY, 5., 1999. **Proceedings**... San Diego: ACM Press, 1999. p. 145-154.

[BER97] BERRY, Michael J.A.; LINOFF, Gordon. **Data Mining Techniques.** New York : John Wiley, 1997. 454p.

[BRU98] BRUSSO, Marcos José. **O Paralelismo na Mineração de Regras de Associação**. Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[CAM2000] CAMARGO, Sandro da Silva. **Mineração de Regras de Associação No**

Problema da Cesta de Compras. Trabalho Individual (Mestrado em Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[CAM2001] CAMARGO, Sandro da Silva; ENGEL, Paulo Martins. MIRABIT: Mineração de regras de associação em bancos de dados de comércio varejista de confecção. **Revista do Ccei**, Bagé, v.5, n.8, p.12-18, 2001.

[CAM2001a] CAMARGO, Sandro da Silva; ENGEL, Paulo Martins. MiRABIT: um novo algoritmo para mineração de regras de associação. In: SEMINÁRIO REGIONAL DE INFORMÁTICA, 11., 2001. **Anais**... Santo Ângelo: Gráfica Venâncio Ayres, 2001. v.1. p.71-72.

[CAM2001b] CAMARGO, Sandro da Silva; ENGEL, Paulo Martins. MiRABIT: um novo algoritmo de mineração para regras de associação. In: OFICINA DE INTELIGÊNCIA ARTIFICIAL, 5., 2001. **Anais**... Pelotas: Educat – Ed. da Universidade Católica de Pelotas, 2001.

[CHE96] CHEN, Ming-Syan; HAN, Jiawei; YU, Philip S.. Data Mining: An Overview from a Database Perspective. **IEEE Transactions On Knowledge and Data Engineering**, New York, v. 8, n. 6, p. 866-883. Dec. 1996.

[FAY96] FAYYAD, U. M. et al. From data mining to knowledge discovery: an overview. In: FAYYAD, U. M. et al. **Advances in Knowledge discovery and data mining**. Menlo Park: MIT Press, 1996. p. 37-54.

[FRE98] FREITAS, Alex A. A Survey of Parallel Data Mining. In: INT. CONFERENCE ON THE PRACTICAL APPLICATIONS OF KNOWLEDGE DISCOVERY AND DATA MINING, 2., 1998. **Proceedings**... Londres: [s.n.], 1998. p.287-300.

[HAN97] HAN, Eui-Hong; KARYPIS, George; KUMAR, Vipin. Scalable Parallel Data Mining for Association Rules. In: ACM SIGMOD INT. CONFERENCE ON MANAGEMENT OF DATA, 1997. **Proceedings**... Tucson: ACM Press, 1997.

[HAN2000] HAN, Eui-Hong; KARYPIS, George; KUMAR, Vipin. Scalable Parallel Data Mining for Association Rules. **IEEE Transactions on Knowledge and Data Engineering**, New York, v. 12, n. 1, p. 377-352, 2000.

[HEU98] HEUSER, Carlos Alberto. **Projeto de Banco de Dados**. Porto Alegre: Sagra-Luzzatto, 1998. 202p.

[HOL94] HOLSHEIMER, M.; SIEBES, A. **Data mining**: the search for knowledge in databases. Amsterdan: Nacional Research Institute for Mathematics and Computer Science in the Nederlands, 1995. (Technical Report CS-R9406). Disponível em: <http://www.cwi.nl/ftp/CWIreports/AA/CS-R9406.ps.Z>. Acesso em: jan. 2001.

[PAR95] PARK, J.-S.; CHEN, M.-S. Chen, YU, P.S. Na Effective Hash Based Algorithm for Mining Association Rules. In: ACM SIGMOD INT. CONFERENCE ON MANAGEMENT OF DATA, 1995. **Proceedings**... San Jose: [s.n.], 1995. p. 175-186.

[PAR98] PARTHASARATHY, Srinivasan; ZAKI, Mohammed J.; LI, Wei. Memory Placement Techniques for Parallel Association Mining. In: KDD CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, 4., 1998. **Proceedings**... New York: AAAI Press, 1998. p. 304-308.

[SRI96] SRIKANT, Ramakrishnan; AGRAWAL, Rakesh. Mining Quantitative Association Rules in Large Relational Tables. **SIGMOD Record**, New York, v.25, n.2, p.1-12, June 1996. Trabalho apresentado na ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1996, Montreal.

[SRI97] SRIKANT, Ramakrishnan; VU, Quoc; AGRAWAL, Rakesh. Mining Association Rules with Item Constraints. In: ACM KDD INT. CONFERENCE ON KNOWLEDGE DISCOVERY IN DATABASES AND DATA MINING, 3., 1997. **Proceedings**... Newport Beach: AAAI Press, 1997. p. 67-73.

[TOS2001] TOSCANI, Laira Vieira; VELOSO, Paulo A. S. **Complexidade de algoritmos**: análise, projeto e métodos. Porto Alegre: Sagra Luzzatto, 2001. 202p.

[ZAK97] ZAKI, Mohammed J.; PARTHASARATHY, Srinivasan; LI, Wei. A Localized Algorithm for Parallel Association Mining. In: ACM SYMPOSIUM ON PARALLEL ALGORITHMS AND ARCHITECTURES, 9., 1997. **Proceedings**... Newport: ACM Press, 1997. p. 321-330.

[ZAK97a] ZAKI, Mohammed J.; PARTHASARATHY, Srinivasan; LI, Wei. New Algorithms for Fast Discovery of Association Rules. In: ACM KDD INT. CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, 3., 1997. **Proceedings**... Newport: AAAI Press, 1997. p. 283-286.