REGRAS DE ASSOCIAÇÃO E CLASSIFICAÇÃO EM AMBIENTE DE COMPUTAÇÃO PARALELA APLICADAS A SISTEMAS MILITARES

Alexandre Soares Alves

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA CIVIL.

Aprovada por:

_____
Prof. Nelson Francisco Favilla Ebecken, D.Sc.

_____
Prof. Alexandre Gonçalves Evsukoff, Dr.

_____
Profa. Marta Lima de Queirós Mattoso, D.Sc.

_____
Profa. Myrian Christina de Aragão Costa, D.Sc.

_____
Prof. Mário Antonio Ribeiro Dantas, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2007

ALVES, ALEXANDRE SOARES

Regras de Associação e Classificação em Ambiente de Computação Paralela Aplicadas a Sistemas Militares [Rio de Janeiro] 2007

XII, 149 p., 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia Civil, 2007)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Regras de Associação e Classificação

1. Mineração de Dados

2. Estruturas de Dados

3. Processamento Paralelo

I. COPPE/UFRJ II. Título (série)

Ao meu pai, Edio , pelo incondicional apoio e incentivo,

à minha mãe Iacy, por sua força e motivação,

e aos meus filhos, Breno, Danilo e Thaís, pela compreensão e carinho.

AGRADECIMENTOS

Agradeço:

ao meu orientador, Professor Nelson Ebecken, pela confiança, estímulo e paciência a mim dedicados, mas sobretudo pelo exemplo de profissionalismo e dedicação sempre presente neste e nos diversos trabalhos que colabora;

aos Professores Alexandre Evsukoff, e Myrian Aragão pelos ensinamentos a mim ministrados em seus cursos, que de forma valiosa contribuíram no desenvolvimento deste trabalho;

ao Instituto de Pesquisas da Marinha (IPqM) pela liberação concedida para tornar possível este trabalho e acreditar no seu resultado;

aos companheiros de trabalho do IPqM, Ângela, Clara, Beth, Maira, Amaury, Falavane, Sahate, Vicente e tantos outros que torceram pelo meu sucesso e me apoiaram sempre que precisei. Mas em especial ao Alexandre Coser, que possibilitou acesso aos equipamentos necessários aos testes desse trabalho; E ao Rafael pela preparação formatação das bases de dado utilizadas na tese.

ao Capitão-de-Fragata(EN) Alexandre de Assis Motta, principal incentivador pelo meu ingresso no doutorado, pelo seu companheirismo, visão científica e pelas palavras sempre inteligentes e equilibradas;

a todos os funcionários da COPPE-PEC pelo eficiente apoio administrativo.

à minha grande amiga e companheira de "aventuras", Selma Foligne, que a exemplo do curso de mestrado, trilhamos juntos o doutorado. Não bastasse sua beleza como incentivo e motivação, sua inteligência e espírito de companheirismo, certamente foram fundamentais para conclusão dessa etapa;

ao meu amigo Luiz Marini, por suportar meu nervosismo e mau-humor, pelas palavras de incentivo e apoio, sempre presentes, impedindo que as adversidades do trabalho e da vida desmotivassem a minha caminhada;

a minha amiga e mãe de meus filhos, Taslene, que com sua calma e paciência natural sempre me apoiou e me encorajou para conclusão de mais essa etapa.

à minha família pela compreensão das longas horas de ausência de um convívio tão prazeroso. Em especial ao meu pai, Edio, pelo seu incondicional apoio; à minha mãe Iacy, pelas incontáveis palavras de incentivo e motivação e aos meus filhos Breno, Danilo e Thaís pelo carinho e compreensão;

aos meus irmãos e parentes queridos, Cristina, Edinho, Terezinha, Viviane, Monique, Marcinho e Morgana que sempre me apoiaram;

a tantos mais que compartilharam comigo desse caminho;

e, acima de tudo, a Deus.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

REGRAS DE ASSOCIAÇÃO E CLASSIFICAÇÃO EM AMBIENTE DE COMPUTAÇÃO PARALELA APLICADAS A SISTEMAS MILITARES

Alexandre Soares Alves

Março/2007

Orientador: Nelson Francisco Favilla Ebecken

Programa: Engenharia Civil

Atualmente, tem sido verificado que a descoberta de conhecimento em grandes bases de dados vem ganhando importância e interesse, sobretudo em áreas estratégicas nas organizações.

Na área militar, os dados e as informações neles contidas, sejam de forma explícita ou implícita, são normalmente confidenciais. Então, o completo domínio de uma ferramenta de mineração de dados é fundamental para garantir a adequação desta a sua aplicação, sem que as informações ou dados sejam expostos.

O presente trabalho apresenta o desenvolvimento de uma ferramenta de mineração de itens freqüentes, em ambiente de computação paralela, para a geração de regras de associação e de classificação. E ainda, um estudo de caso em um sistema militar de contramedidas.

Os experimentos mostram que foram alcançados os objetivos principais desse trabalho. A implementação, em ambiente paralelo, de um classificador baseado em associações e a extração de regras de associação e de classificação da base de dados do sistema de contramedidas contribuir para o aprimoramento, produzindo resultados mais rápidos e melhores que o método original.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

ASSOCIATION AND CLASSIFICATION RULES IN PARALLEL COMPUTATION ENVIRONMENT APPLIED TO MILITARY SISTEM

Alexandre Soares Alves

March/2007

Advisor: Nelson Francisco Favilla Ebecken

Department: Civil Engineering

Nowadays, knowledge discovery in great databases has grown in importance and interest, especially in strategic areas in the organizations.

In the military field, data and information are usually considered classified material, so that the complete understanding of any data mining tool is a requirement to ensure its application adequacy, in which the given information or data must not be disclosed.

This works introduces the development of a frequent item mining tool, in a parallel computational environment, for generation of classification and association rules. A case study in a military countermeasure system is also presented.

Experiments have shown that the main objectives of this work had been reached. The parallel implementation of the proposed association-based classifier algorithm and the association and classification rules extraction from the object of study database, aiming the system performance improvement, has shown to produce better results that the original method.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo I

Introdução

Nos últimos anos houve um crescimento substancial da quantidade de dados armazenados, devido ao aumento da facilidade e a redução dos custos para manter essas informações. Essa imensa quantidade de dados armazenada é inviável de ser analisada por especialistas através de métodos convencionais. Assim, a dificuldade de uma análise mais precisa destes dados colabora para que os mesmos se transformem em apenas um amontoado de informações sem utilidade. Por outro lado, sabe-se que nas grandes quantidades de dados pode existir um enorme potencial de informação, muito embora as informações contidas nos dados não estejam caracterizadas explicitamente, já que sendo estes dados operacionais, não interessam quando estudados individualmente.

Logo, a descoberta de conhecimento em bases de dados vem ganhando grande importância e interesse, pois, as pesquisas nessa área aumentaram e visam à descoberta de tecnologias mais eficientes para a recuperação de informações, procurando encontrar conhecimentos implícitos que possam ser úteis [1]. Então, denomina-se esse processo como mineração de dados, ou seja, a análise de grandes volumes de dados utilizando técnicas computacionais para extração de conhecimento não trivial e potencialmente útil.

1.1. POR QUE ITENS FREQÜENTES ?

Dentre os diversos campos de pesquisa na área de mineração de dados, um dos mais importantes e que mais tem sido alvo de pesquisas é a *mineração de itens freqüentes* [2], [3], [4], isto é, a busca de subconjuntos que aparecem, com certa freqüência, numa seqüência e dados.

A mineração de itens freqüentes aparece como principal subproblema das tarefas de mineração de padrões seqüências e de mineração de regras de associação [2], [3], [4]. Entretanto, outras tarefas de mineração de dados podem ser obtidas como uma aplicação da mineração de itens freqüentes, ao invés de serem implementadas através de seus métodos convencionais. Dentre estas estão a classificação [5], [10], a *clusterização* [6], a *Web Mining* [7], [8] e etc. Além do que muitos algoritmos e técnicas desenvolvidos para mineração de itens freqüentes são empregados em outras áreas de pesquisa. Como exemplo, pode-se citar a busca de seqüências ótimas, para construção de diagramas ordenados de decisão binária (OBDD) [9], que são utilizados na solução de equações *booleanas* e na confecção de circuitos lógicos integrados.

Dentre as tarefas de mineração de dados, as regras de associação e as de classificação são consideradas de grande relevância. Visto que, regras são as formas de conhecimento humano mais expressivas e inteligíveis. O que justifica a importância e os esforços de desenvolvimento de algoritmos, técnicas e ferramentas para a mineração de itens freqüentes.

1.2. MOTIVAÇÃO

A análise de dados ou a descoberta de novos conhecimentos a partir de uma massa de dados operacionais, como já mencionado nesse capítulo, só é possível se os dados forem analisados de forma conjunta. Portanto, sistemas computacionais eficientes são pré-requisitos para mineração de dados.

A maioria das tarefas de mineração de dados, em especial a mineração de itens freqüentes e consequentemente a geração de regras, sofre fortes restrições em suas

implementações devido a diversos fatores, sobretudo, o caráter iterativo da maioria dos algoritmos de mineração de itens freqüentes, aliado a grande quantidade de dados, que limitam as plataformas de *hardware* onde estes podem ser executados. Outra restrição importante é a natureza dos dados, que podem influenciar o problema de explosão combinatória de candidatos a itens freqüentes ou de regras [10], [13].

Apesar da generalidade da maioria dos algoritmos desenvolvidos para as tarefas de associação e classificação é evidente que as implementações mais robustas e eficientes são aquelas onde se levam em conta, além do *hardware* disponível, o conhecimento dos dados e sua natureza [14]. Portanto, acredita-se que a maior motivação para implementações de ferramentas de mineração de dados para aplicações específicas resida no fato de se obter maior domínio do sistema, facilitando, assim, futuras adaptações e acréscimos de funcionalidades, na medida em que estas se mostrarem necessárias.

Sistemas de inteligência computacional e de mineração de dados são utilizados em aplicações militares [15], [16]. Essas técnicas podem ser especialmente importantes na área de desenvolvimento de contramedidas, onde o conhecimento dos parâmetros de uma ameaça, para qual será desenvolvida a contramedida, às vezes, é bastante limitado. Portanto, o uso de técnicas e sistemas de mineração de dados é fundamental na busca de informações que possam contribuir para a eficácia do desenvolvimento.

Dentre as contramedidas utilizadas por navios para se defender de ataques de mísseis está o emprego de foguetes de *chaff* [17], isto é, foguetes com cargas de filamentos metalizados, para a formação de nuvens, com o propósito de confundir ou seduzir o radar do míssil. Entretanto, estas nuvens devem ser formadas em posições chaves, de modo a conseguir desviar, eficazmente, o míssil do seu alvo. Para isto, é necessário um sistema de lançamento de foguetes de *chaff*, que determina através da análise de diversos parâmetros a posição, o momento de abertura da carga e a quantidade de foguetes capazes de desviar a trajetória do míssil. A maior dificuldade no desenvolvimento desse tipo de sistema de contramedida é a falta de informações precisas da ameaça, haja vista que estas, normalmente, são informações secretas. Assim, a mineração de dados de cenários táticos envolvendo diversas condições e comportamentos de ameaças, além de parâmetros meteorológicos e de navios pode levar

a descoberta de importantes informações que contribuam para o aprimoramento desses sistemas de contramedida. Entretanto, a grande quantidade de dados envolvidos, aliada a natureza secreta destes, restringe a utilização das ferramentas de mineração.

A importância da análise computacional dos dados, aliada às restrições das aplicações militares, conforme descrito, motivaram o desenvolvimento de uma ferramenta capaz de realizar a tarefa de mineração de itens freqüentes para geração de regras de associação e de classificação, de modo a permitir a busca de novos conhecimentos para o aprimoramento de sistemas de contramedida. Devido à possibilidade de lidar com grandes bases de dados, a implementação visa à execução em ambiente paralelo, na tentativa de garantir a escalabilidade. Além do mais, o caráter confidencial dos dados exige total conhecimento das estruturas e rotinas que compõe tal ferramenta, de maneira a possibilitar integrações futuras de novas tarefas ou funcionalidades, que dependam de algum modo da interpretação desses dados.

1.3. CONTRIBUIÇÕES

No desenvolvimento dessa tese pode-se citar como principais contribuições:

- A implementação em ambiente computacional paralelo de um classificador baseado em associações, idéia já experimentada com êxito em implementações seriais [10], [11], [12];

- A utilização de uma estrutura de armazenamento de dados mais simples, baseada em classes e seu comportamento em ambiente paralelo; e

- O emprego de tarefas de mineração de dados com a finalidade de aprimorar sistemas antimíssil.

É importante ressaltar que este trabalho de pesquisa representa um esforço na investigação para o desenvolvimento de outras tarefas e/ou metodologias envolvendo mineração de dados aplicada aos problemas inerentes à área militar, sobretudo aos da Marinha do Brasil.

1.4. ORGANIZAÇÃO DA TESE

Esta tese é composta além deste capítulo, de outros cinco capítulos adicionais que estão organizados da seguinte forma:

- O Capítulo II aborda conceitos básicos para o entendimento dos principais assuntos discutidos na Tese. Neste capítulo são apresentadas as regras de associação, suas principais definições e propriedades, a idéia básica que norteia a construção dos principais algoritmos de mineração de regras de associação e o algoritmo APRIORI. É apresentada também, a tarefa de classificação sua concepção clássica e a relação entre regras de associação e regras de classificação.

- O Capítulo III trata da revisão bibliográfica. Ele é iniciado com a descrição de algumas características que influenciam o desempenho dos algoritmos de regras de associação. Em seguida, apresenta um sumário das principais abordagens seqüenciais para o problema e as estratégias conhecidas para geração de regras de classificação baseada em associações, detalhando principalmente o algoritmo CBA. Finalmente, sumariza as estratégias paralelas para mineração de itens freqüentes, destacando os principais algoritmos paralelos e suas características.

- O Capítulo IV apresenta os detalhes do desenvolvimento e da implementação da ferramenta. São descritas as principais fases do processo de desenvolvimento, as estratégias e estruturas de dados adotadas, as dificuldades e limitações encontradas, as ferramentas de apoio, desenvolvidas para testes de corretude e de eficiência dos algoritmos. Além dos detalhes da implementação para um *cluster de PC's*.

- O Capítulo V apresenta o estudo de casos, isto é, a descrição do sistema de contramedidas, seus dados e parâmetros. E em seguida é apresentado o ambiente dos testes, a descrição dos experimentos e os resultados.

- O Capítulo VI apresenta as considerações finais, conclusões, trabalhos futuros e sugestões.

Capítulo II

Regras de Associação e Classificação

Neste capítulo são abordados os conceitos básicos para o entendimento dos principais assuntos discutidos na Tese. Nele são apresentadas as regras de associação, suas principais definições e propriedades. A idéia básica que norteia a construção dos principais algoritmos de mineração de regras de associação e o algoritmo APRIORI, principal algoritmo para essa tarefa. É apresentada, também, a tarefa de classificação, sua concepção clássica e a relação entre regras de associação e regras de classificação.

2.1 REGRAS DE ASSOCIAÇÃO

Essa seção introduz a tarefa de mineração de dados denominada regras de associação, começando pela apresentação do problema formal, seguindo com as propriedades das regras e com a definição de conjuntos freqüentes. A parte final desta seção apresenta uma descrição sucinta das principais características que compõem a maioria dos algoritmos de mineração de regras de associação.

2.1.1 Descrição Geral

Regras de associação representam padrões onde a ocorrência de eventos em conjunto é alta. Pode-se dizer que é a probabilidade de que um conjunto de itens apareça em uma transação, dado que outro conjunto esteja presente. O objetivo de minerar regras de

associação é encontrar todos os conjuntos de itens que freqüentemente ocorrem de forma conjunta na base de dados e formar regras a partir destes conjuntos.

Quanto ao tipo de descoberta, as regras de associação são consideradas um método não supervisionado, e obtém o segundo lugar em percentual de utilização em aplicações, somente atrás da tarefa de classificação [5],[10]. As principais áreas de aplicação de regras de associação são: suporte a decisão em sistemas de diagnósticos; análise de dados de vendas; descoberta de tendências; estudo das informações coletadas durante a venda simultânea de itens, também chamada de "*basket data*", em [2], [3], [4]; dentre outras. Resumindo, as regra de associação disponibilizam uma análise a posteriori das conexões mais comuns e/ou confiáveis entre os dados.

As regras de associação podem ser vinculadas a outras tarefas, como, por exemplo, a *clusterização*, onde as regras de associação são geradas a partir de um conjunto de dados agrupado por algum atributo, além da transação de compra citada em [2]. Ampliando ainda mais suas áreas de aplicação, tem-se, também, a utilização de regras de associações em ferramentas de descoberta de falhas em sistemas mecânicos ou eletros-mecânico, projeto de catálogo de produtos, segmentação de mercado, diagnóstico de alarmes, etc. [55], [56], [57]. Outra utilidade das regras de associação é a mineração realizada em bases de dados taxonômicas, ou seja, aquelas que possuem uma hierarquia entre os itens armazenados. Devido à presença dessa hierarquia, as regras não associam somente itens, mas também classes de itens [14].

As regras de associação foram introduzidas em [2], como resultado de pesquisas conduzidas independentemente no projeto "*Quest*" na *IBM Almaden Research Center* e na Universidade de *Helsinki* [61]. Os algoritmos existentes para essa tarefa buscam encontrar todas as regras que satisfaçam certos parâmetros, e não somente verificar se uma regra em particular é válida. Isso porque a simples verificação tem a limitação de poder desconsiderar regras novas e mudanças nas tendências de um universo hipotético [3]. Entretanto, a tarefa de se descobrir todas as associações freqüentes em bases de dados muito grandes é uma tarefa árdua, pois, o espaço de busca é exponencial em relação ao número de atributos (valores distintos de itens) da base de dados, e em se tratando de milhões de transações, o problema de minimização de acessos E/S se torna essencial. Mesmo assim, a maioria dos algoritmos existentes para resolver tal problema

é iterativa e requerem múltiplas leituras na base de dados. Alguns métodos, especialmente aqueles relacionados a alguma forma de amostragem, podem não corresponder a uma correta representação do universo global, comprometendo, desta forma, os resultados.

De forma mais simplificada, como dito em [3], a mineração de regras de associações (MRA) busca encontrar todos os conjuntos de itens que frequentemente ocorrem de forma conjunta na base de dados, e também as regras formadas entre esses conjuntos.

2.1.2 Conjuntos Freqüentes

Alguns fundamentos teóricos básicos e algumas definições são necessários para o entendimento do problema de mineração de itens freqüentes.

Definição 2.1 - ITEMSET

Para qualquer conjunto X, seu tamanho é o número de elementos em X. Seja N o conjunto de "*n*" números naturais { 1, 2, 3, ..., n }. Cada $x \in N$ é chamado de ***item***. Um subconjunto não vazio de N é chamado um ***itemset***. Finalmente, chama-se conjunto de itens X com cardinalidade K= |X| um "**K-*itemset***".

Definição 2.2 - TRANSAÇÃO

Dado um conjunto I = { I_1, I_2, ..., In } de itens, uma transação T é definida como qualquer subconjunto de itens em I, isto é $T \subseteq I$. Já que T é um conjunto, este não pode conter elementos duplicados, entretanto pode-se assumir, sem perda de generalidade, que tanto os conjuntos de transação T, quanto os demais conjuntos e subconjuntos tratados neste trabalho estão ordenados.

Definição 2.3 - TID

Seja uma base de dados D um conjunto de "*n*" transações, então cada transação nesta base de dados é rotulada com um único identificador de transação, denominado TID.

Diz-se que uma transação T suporta um conjunto $X \subseteq I$ se esta contém todos os elementos de X, isto é, se $X \subseteq T$. Algumas vezes será necessário referir-se ao conjunto de transações que suportam X, logo será adotada a seguinte notação T(X), para denotar o conjunto de TID destas transações.

Definição 2.4 - SUPORTE

Define-se suporte de X, sup(X), como a fração de todas as transações em D que suportam X. E suporte mínimo, **sup_min**, um valor pré-estabelecido que represente o limite mínimo para este parâmetro. O valor estabelecido para o suporte mínimo, tem a finalidade de concentrar a problemática apenas nos conjuntos de itens que ocorrem com uma freqüência suficiente em D, para serem considerados interessantes.

Definição 2.5 - ITEMSET FREQÜENTE e ITEMSET FREQÜENTE MAXIMAL

Um *itemset* X é freqüente se $sup(X) \geq sup\_min$. Uma coleção de *itemsets* freqüentes é denotada como $F(sup\_min, D)$. Um *itemset* freqüente $X \in F(sup\_min, D)$ é maximal se este não pertencer a nenhum superconjunto freqüente.

Há três propriedades dos conjuntos freqüentes que são úteis para a teoria de regras de associação, sobretudo as propriedades (2.2) e (2.3), que formam a fundamentação da maioria dos algoritmos de mineração de regras de associação [63].

Propriede 2.1 - SUPORTE DOS SUBCONJUNTOS:

Se $A \subseteq B$ então o $sup(A) \geq sup(B)$, já que todas as transações em D que suportam B necessariamente, também, suportam A.

Propriede 2.2 - CONJUNTOS INFREQÜENTES:

Se A é um conjunto infreqüente, isto é, $sup(A) < sup\_min$, todo superconjunto B de A é também infreqüente, já que pela propriedade (1), $sup(A) \geq sup(B)$, então $sup(B) < sup\_min$.

Propriede 2.3 - CONJUNTOS FREQÜENTES:

Se B é um conjunto freqüente, isto é, sup(B) ≥ sup_min, todo subconjunto A de B também é freqüente em D. Já que A⊆B, pela propriedade (1), sup(A) ≥ sup(B), portanto sup(B) ≥ sup_min.

Propriede 2.4 - EQUISUPORTES:

Seja $A \subset B \subseteq I$, se o sup(A) = sup(B) $\Rightarrow$ sup($A \cup Z$) = sup ($B \cup Z$), para qualquer $Z \subseteq I$.

2.1.3 Definição Formal de Regras de Associação

Uma regra de associação é uma implicação da forma R: X⇒Y, onde X, antecedente, e Y, conseqüente, são conjuntos de itens disjuntos: X, $Y \subset I$ e $X \cap Y = \varnothing$. Além do mais $Y \neq \varnothing$. Tal regra pode ser entendida como uma predição, já que se uma transação T suporta X, esta também suportará Y com certa probabilidade, a qual é chamada de confiança da regra e denotada como conf (R).

Definição 2.6 - Confiança

A confiança de uma regra R é definida como a probabilidade condicional, onde dada uma transação T que suporta X, T também suportará Y, ou mais formalmente:

$$\text{conf}(\text{R}) = \text{p}\left(\text{Y} \subseteq \text{T} \mid \text{X} \subseteq \text{T}\right) = \frac{p(Y \subseteq T \wedge X \subseteq T)}{p(X \subseteq T)} = \frac{\sup(X \cup Y)}{\sup(X)} \qquad \text{(EQ 2.1)}$$

Definição 2.7 - Regra Forte

O suporte de uma regra R em D é definido como sup($X \cup Y$). A confiança de uma regra revela o quanto esta regra é aplicada, enquanto que o suporte indica o quanto esta regra é confiável. Portanto, para uma regra ser relevante é preciso ter bastante suporte e suficiente confiança. Assim pode-se dizer que uma regra é relevante ou forte se esta apresenta suporte e confiança acima de valores prefixados de confiança mínima e de suporte mínimo, isto é, R é dita forte se conf(R) ≥ conf_min e sup(R) ≥ sup_min. Note que uma condição necessária para a formação de tais regras é que ambos os conjuntos antecedentes e conseqüentes da regra sejam itens freqüentes.

2.1.3.1 Propriedade das regras de associação

O fato de o antecedente e o conseqüente serem conjuntos disjuntos não é absolutamente necessário. A não observância desse preceito não leva a regras absurdas e sim a regras redundantes ou insignificantes. Por exemplo, $X \Rightarrow X$ é trivialmente verdadeiro. Assim $X \Rightarrow X \cup Y$ é equivalente a $X \Rightarrow Y$ e, portanto, desinteressante.

O antecedente X de uma regra pode ser vazio, pois toda transação suporta o conjunto vazio e, portanto toda base de dado satisfaz o antecedente. A confiança de tais regras é igual à freqüência relativa do conseqüente. Entretanto o conseqüente Y não pode ser vazio, pela mesma razão que o antecedente e o conseqüente devem ser disjuntos, isto é, se $Y \neq \varnothing$ para R: $X \Rightarrow \varnothing$, a $conf(R) = \frac{\sup(X \cup \varnothing)}{\sup(X)} = 1$ o que demonstra que as regras $X \Rightarrow X$ e $X \Rightarrow \varnothing$ são equivalentes.

As regras de associação não são, geralmente, transitivas e não fazem composição [63]. Na maioria dos casos se uma regra é válida ou inválida, não se pode inferir a partir da confiança de uma outra regra. Portanto, as seguintes propriedades são estabelecidas:

Propriede 2.5 - COMPOSIÇÃO:
Se $X \Rightarrow Z$ e $Y \Rightarrow Z$ são regras válidas em D, o mesmo não é necessariamente verdade para $X \cup Y \Rightarrow Z$. Considere o caso quando, $X \cap Y = \varnothing$ e existem transações em D que suportam Z se e somente se essas transações suportam também, ou X ou Y, mas nunca ambos. Então o conjunto $X \cup Y$ tem suporte $\varnothing$ e, portanto $X \cup Y \Rightarrow Z$ tem confiança 0%. Pode-se utilizar argumento similar na composição de regras com os mesmos antecedentes, logo $X \Rightarrow Y \wedge X \Rightarrow Z \Rightarrow X \not\Rightarrow Y \cup Z$.

Propriede 2.6 - DECOMPOSIÇÃO:
Se $X \cup Y \Rightarrow Z$ é uma regra válida, $X \Rightarrow Z$ e $Y \Rightarrow Z$ podem não ser válidas. Esta propriedade mostra que não procede à decomposição dos antecedentes. Por exemplo, considere o caso em que Z está presente em uma transação apenas quando X e Y também estão presentes, isto é sup($X \cup Y$) = sup(Z). Assim se o suporte de X ou

suporte de Y é suficientemente maior que o suporte de X ∪ Y, as duas regras individualmente não tem confiança requerida. Veja por exemplo, se:

$$conf\left(X \cup Y \Rightarrow Z\right) = \frac{\sup\left(X \cup Y \cup Z\right)}{\sup\left(X \cup Y\right)} \geq conf\_\min \qquad \text{(EQ.2.2)}$$

e $$conf\left(X \Rightarrow Z\right) = \frac{\sup\left(X \cup Z\right)}{\sup\left(X\right)} \qquad \text{(EQ.2.3)}$$

Caso sup (X) >>> sup (X ∪ Y), pode-se afirmar que conf (X⇒Z) < conf_min. Logo X⇒Z não é uma regra forte, embora X ∪ Y⇒ Z seja por hipótese. Todavia, a decomposição de conseqüentes procede. Se X ⇒ Y ∪ Z é uma regra válida, então X ⇒ Z e Y ⇒ Z também são válidas, já que sup(X ∪Y) > sup(X ∪ Y ∪ Z) e sup(X ∪ Z) > sup(X ∪ Y ∪ Z). Portanto, ambos os suporte e confiança da menor regra aumentam comparados com a regra original.

Propriede 2.7 - TRANSITIVIDADE:

Se X ⇒ Y e Y ⇒ Z são regras fortes, não se pode inferir que X ⇒ Z também seja uma regra forte. Assuma por exemplo, que T(X) ⊂ T(Y) ⊂ T(Z) e que a confiança mínima é dada por conf_min. Seja conf (X ⇒ Y) = conf (Y ⇒ Z) = conf_min, tomando-se por base os valores relativos de suporte, pode-se concluir que conf (X ⇒ Z) = $(conf\_min)^2$, para valores de conf_min < 1, tem-se conf (X ⇒ Z) = $(conf\_min)^2$ < conf_min, logo a regra X ⇒ Z não é forte.

Propriede 2.8 - INFERÊNCIA:

Se uma regra do tipo A ⇒ (L – A) não apresenta confiança mínima, então uma outra regra do tipo B ⇒ (L – B), também não apresenta, desde de que B ⊆ A. Pela propriedade (1) tem-se que sup(B) ≥ sup(A) e pela definição de confiança obtém-se $conf\left(B \Rightarrow \left(L - B\right)\right) = \frac{\sup\left(L\right)}{\sup\left(B\right)} < \frac{\sup\left(L\right)}{\sup\left(A\right)} < conf\_\min$. Desta forma, se a regra (L-C)⇒C é uma regra forte, então (L-D) ⇒ D também é forte para o caso de D ⊆ C e D ≠ ∅.

2.2 IDÉIA BÁSICA DOS ALGORITMOS DE MRA

Essa seção apresenta a idéia básica das duas principais etapas que compõe os algoritmos de mineração de regras de associação. Embora os algoritmos que tratam do problema de mineração de regras de associação sejam diferentes entre si, todos utilizam um esquema básico. Para obter todas as regras de associação, os suportes de todos os itens freqüentes da base de dados têm de ser computados. Portanto, todos os algoritmos para esta tarefa de mineração de dados dividem-se em duas fases:

a) Geração de todos os itens freqüentes;
b) Geração das regras de associação.

2.2.1 Determinação de Itens Freqüentes

Como ilustrado pela figura 2.1 e já mencionado anteriormente, o número de candidatos a conjuntos freqüentes cresce exponencialmente com o número de itens considerados. Um algoritmo guloso pode realizar uma busca exaustiva e testar todos os componentes do espaço de busca para verificar se estes são ou não freqüentes.

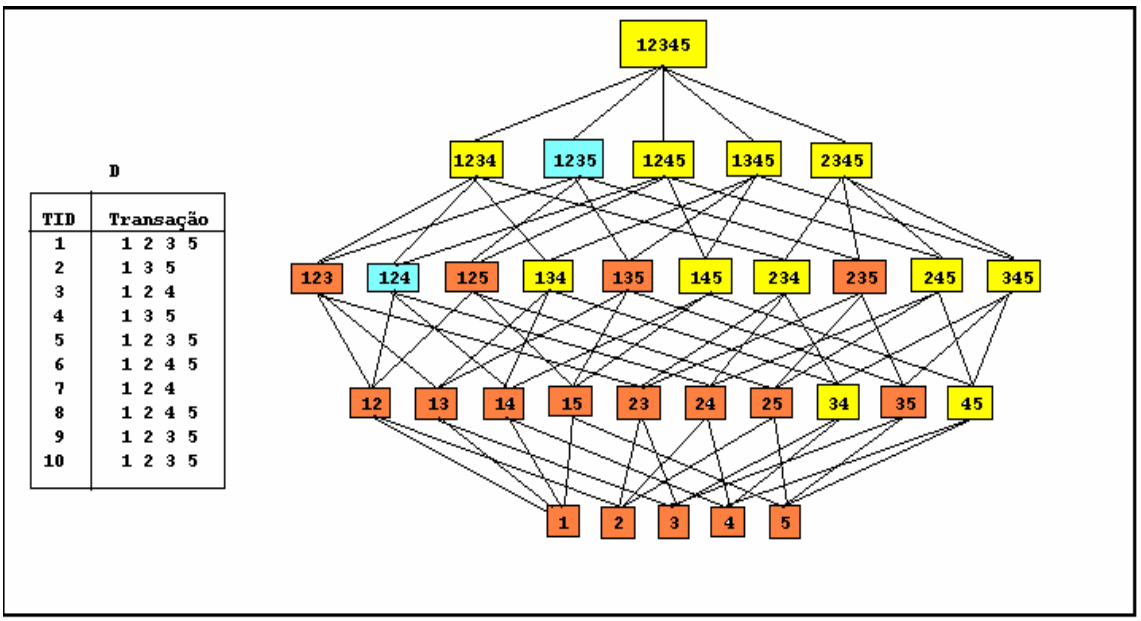


Figura 2.1 : Exemplo de mineração de *itemsets* freqüentes

O método básico seguido pela grande maioria dos algoritmos é criar um conjunto de itens prováveis, chamados de conjunto de candidatos, que podem ou não ser freqüentes. A quantidade de candidatos, a seqüência e a freqüência que são criados dependem particularmente da solução dada a cada algoritmo.

Para descobrir quais destes candidatos são realmente freqüentes e quais são seus suportes exatos, devem ser realizadas contagens sobre a base de dados. Porém como tal tarefa requer um tempo de processamento considerável e normalmente grande quantidade de memória, o objetivo mais óbvio que é perseguido pelos algoritmos é a redução do número de candidatos gerados.

2.2.2 Construção das Regras de Associação

Após a contagem e todos os suportes estarem disponíveis, as regras possíveis podem ser geradas e suas confianças determinadas. Para todo conjunto freqüente X, cada subconjunto é convenientemente escolhido como antecedente de uma regra e os itens restantes constituem o conseqüente. Já que X é freqüente, todos os seus subconjuntos também são freqüentes, de acordo com a propriedade (2.3), e seus suportes são conhecidos. Assim a confiança da regra é calculada e esta é classificada como uma regra forte ou, então, é simplesmente descartada, dependendo do nível de confiança mínimo estabelecido.

Comparando-se com o trabalho de determinação os itens freqüentes pode-se dizer que a tarefa de geração de regras é direta.

2.3 APRIORI

Nesta seção será apresentado de forma detalhada todas as etapas que constituem o principal algoritmo de mineração de regras associação, o APRIORI, considerado a base da grande maioria dos algoritmos que trata desta tarefa de mineração de dados.

2.3.1 O Algoritmo

Este algoritmo foi proposto em [3], para descoberta de regras de associação. Foi o primeiro algoritmo a reduzir significativamente o número de conjuntos candidatos. Isto é, conjuntos de itens possivelmente freqüentes gerados a partir de conjuntos freqüentes até então encontrados, pois quanto maior o número de conjuntos candidatos maior o processamento para encontrar os conjuntos de itens freqüentes. Para isso, o algoritmo se beneficia do princípio de que todos os subconjuntos de um conjunto de itens freqüentes são, necessariamente, freqüentes. Ao gerar um conjunto de itens candidato, pode-se garantir que ele só será um conjunto de itens freqüente se todos os seus subconjuntos também forem freqüentes.

O algoritmo APRIORI faz uso de duas funções, uma para gerar os candidatos e eliminar aqueles que não são freqüentes, e a outra é utilizada para extrair as regras de associação. O primeiro passo do algoritmo é realizar a contagem de ocorrências dos itens para determinar os *itemsets* freqüentes de tamanho unitário (1-*itemsets* freqüentes). Os "K" passos posteriores consistem de duas etapas. Na primeira, os *itemsets* freqüentes F_{K-1}, encontrados na iteração anterior (K-1) são utilizados para gerar os conjuntos de itens potencialmente freqüentes, os *itemsets* candidatos (C_K). Na segunda etapa dessa fase do algoritmo é realizada a contagem dos suportes dos itens candidatos. Então se eliminam aqueles candidatos que não possuem suporte suficiente para serem considerados itens freqüentes, obtendo-se, assim, o conjunto de itens freqüentes (F_K) da iteração corrente. Tais procedimentos serão descritos nos parágrafos seguintes.

A figura 2.2 apresenta o pseudocódigo da parte do algoritmo APRIORI, responsável pela descoberta de itens freqüentes. No primeiro passo do algoritmo (linha 1) simplesmente é contada a ocorrência dos itens na base de dados, para a determinação dos conjuntos freqüentes de tamanho 1 (F_1). Em seguida, diversas iterações são realizadas até que nenhum novo conjunto freqüente seja identificado (linhas 3-11). As iterações subseqüentes são divididas em duas partes. Na primeira parte (linha 5), os conjuntos freqüentes de tamanho K-1 (F_{K-1}) são usados para gerar o conjunto de candidatos (C_K) de tamanho K, onde K corresponde à iteração atual. Na segunda parte (linhas 6 a 9), a base de dados é percorrida e é contado o suporte dos candidatos em C_K. Nessa contagem de suporte, é necessário determinar, os candidatos em C_K que estão

presentes em uma determinada transação. Para isso, é sugerida a utilização de uma árvore *hash* para armazenamento desses candidatos, com o objetivo de diminuir o tempo desse processamento. Após a leitura de todas as transações, os conjuntos candidatos de C_K que possuem suporte maior ou igual ao suporte mínimo são inseridos no conjunto de itens freqüentes (Fk) de tamanho k (linha 10). Na linha 12, o algoritmo determina o conjunto de itens freqüentes resultem da união de todos os conjuntos F_K.

Algoritmo APRIORI

```
1   F1 = conjuntos freqüentes de tamanho 1;
2   k = 1;
3   enquanto ( Fk ≠ ϕ ) faça
4       k = k + 1;
5       Gerar Ck (todos os candidatos de tamanho k) a partir de Fk-1;
6       para cada transação T pertencente a base de dados faça
7           para cada candidato c em Ck faça
8               se todos itens de c pertencem a T então
9               Incrementar o contador de suporte associado a c;
10              Fk = todos os c ∈ Ck com contador ≥ suporte mínimo;
11  fim-enquanto;
12  Resposta = união de todos os conjuntos Fk;
```

Figura 2.2 : APRIORI – Descoberta de itens freqüentes

2.3.2 Geração de Candidatos

Algumas funções do algoritmo merecem explicações adicionais, devido a certas peculiaridades para a obtenção de seus resultados. Entre estas está a geração dos candidatos. Para tal, é necessária a introdução dos conceitos de junção e de poda de candidatos.

Primeiramente, considera-se que os conjuntos possuam seus itens ordenados. No processo de junção, a partir de conjuntos freqüentes gerados no passo K-1, realiza-se a união, dois a dois, dos conjuntos que possuem como interseção os K-2 primeiros

elementos, gerando assim, os conjuntos candidatos de tamanho K, isto é C_K. Para tornar mais claro o entendimento deste processo, suponha, por exemplo, que na iteração K = 4 os conjuntos de itens {1,2,3} e {1,2,4} são verificados como conjuntos freqüentes de tamanho 3. Estes possuem os (K-2) primeiros elementos iguais e, com a aplicação do processo de junção, obtém-se o conjunto candidato {1,2,3,4} de tamanho K = 4, como esquematizado na figura 2.3.

Após um candidato ter sido gerado pelo processo de junção, este passa pelo processo de poda, onde é verificada a propriedade 2.2, isto é, todo subconjunto de um conjunto freqüente é também freqüente. Portanto, o candidato {1,3,4,5} de tamanho K = 4, também mostrado na figura 2.3, que foi gerado pela junção dos conjuntos freqüentes {1,3,4} e {1,3,5} apresenta como seus subconjuntos de tamanho K = 3, os conjuntos {1,3,4}, {1,3,5}, {1,4,5} e {3,4,5}. Porém os subconjuntos {1,4,5} e {3,4,5} não são freqüentes. Logo, valendo-se da propriedade que todo freqüente de tamanho K deve ter todos os seus subconjuntos de tamanho K-1 freqüentes, {1,3,4,5} pode ser descartado do conjunto de candidatos, pois, com certeza, ele não será um conjunto freqüente.

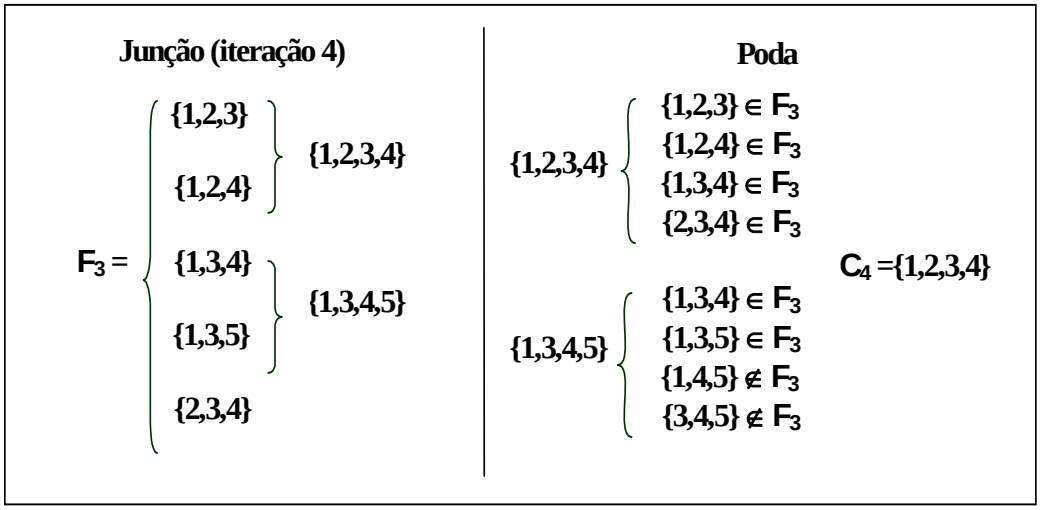


Figura 2.3 : Geração de candidatos no APRIORI

Assim como a junção, a poda de conjuntos candidatos só é realizada a partir da terceira iteração, pois no passo 2 cada conjunto "c" pertencente ao conjunto candidato C_2 é gerado a partir de dois conjuntos freqüentes de tamanho 1, não existindo, assim, um subconjunto de "c" de tamanho 1 que não seja freqüente.

Outra função do algoritmo que merece explicações adicionais é a função que realiza a contagem dos suportes dos itens candidatos. Isto devido à forma na qual os dados são armazenados, pois graças ao grande número de candidatos gerados é necessário armazená-los em estruturas eficientes, que busquem minimizar o tempo de contagem dos suportes dos itens candidatos. Na concepção original do algoritmo APRIORI em [3], foi apresentada uma estrutura de dados denominada árvore *hash*, com o propósito de agilizar o processo de contagem de suportes. Portanto, antes da explicação da função que realiza a tarefa da contagem de suportes, se faz necessário alguns esclarecimentos sobre a árvore *hash*, o que acontece na próxima seção.

2.3.3 Estruturas de Dados

A figura 2.4 ilustra uma árvore *hash*, onde a iteração exemplificada é a terceira, isto é, os conjuntos candidatos têm tamanho K = 3. Além disso, a estrutura se utiliza de uma função *hash*, que resulta em três ramos distintos para cada valor de elemento do conjunto candidato, a saber, $hash_1$={1, 4, 7}; $hash_2$={2, 5, 8} e $hash_3$={3, 6, 9}.

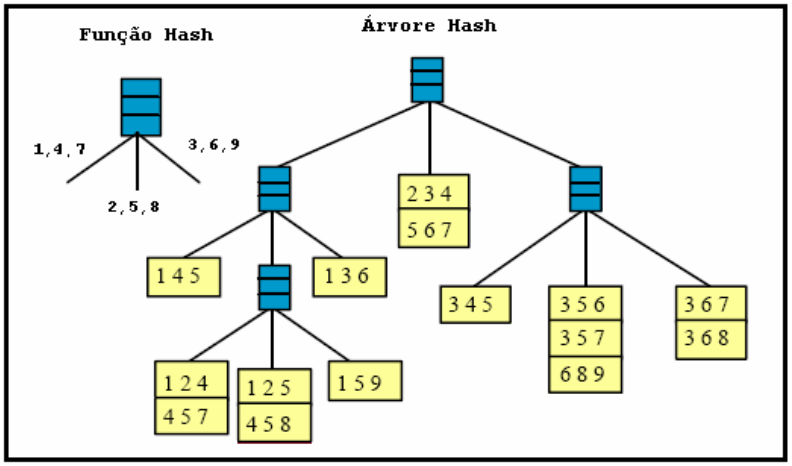


Figura 2.4 : Árvore *Hash*

Um nó interno desta estrutura é simplesmente uma tabela *hash*, cujas entradas apontam para um outro nó. O número de entradas dessas tabelas representa a ordem da árvore. Os nós folhas nesta árvore são formados por uma lista de conjuntos candidatos. A cada conjunto candidato é associado um contador para armazenar a sua freqüência.

Ao ser criada, a árvore *hash* possui apenas o nó raiz sem nenhum conjunto candidato. À medida que os conjuntos candidatos são gerados, estes são inseridos na árvore através de uma função *hash*. Os conjuntos candidatos são inseridos nas folhas, se o número total de conjuntos candidatos da folha não exceder a um determinado limite, definido por um parâmetro chamado "tolerância" ou "saturação" da folha. No caso do número de conjuntos candidatos na folha alcançar o valor desse parâmetro e a profundidade da árvore for menor que K, então este nó folha é transformado em um nó interno. Assim, nós filhos são criados para este novo nó interno e os conjuntos candidatos são redistribuídos de acordo com a função *hash*. Porém, se o último nível, igual a K, for alcançado, o conjunto candidato deve ser inserido no nó folha, independentemente se o número de conjuntos candidatos existentes nesse nó ultrapassa ou não a saturação da folha.

Considere que o conjunto de itens distintos que compõem a base de dados seja o intervalo numérico [1,9]. Suponha que a árvore já possua alguns candidatos como ilustrado na figura 2.5. Suponha ainda, que os parâmetros ordem da árvore e saturação da folha sejam, respectivamente, 3 e 2, e que a função *hash* retorne o resto da divisão do item pela ordem da árvore.

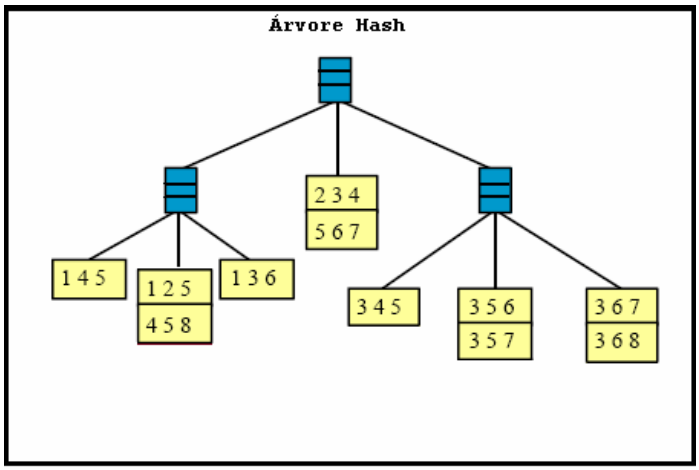


Figura 2.5 : Árvore *Hash* antes da inserção do candidato

A função *hash* é definida da seguinte forma: nó esquerdo contém conjuntos candidatos cujo primeiro item é 1, 4 ou 7; nó central contém conjuntos candidatos cujo primeiro item é 2, 5 ou 8 e o nó direito contém conjuntos candidatos cujo primeiro item é 3, 6 ou 9. Para inserir o novo conjunto candidato { 1, 5, 9 }, a função *hash* é aplicada ao primeiro item que compõem o conjunto obtendo como resultado o nó da esquerda. Em seguida, a função *hash* é aplicada ao próximo item que compõem o conjunto candidato e a folha que contendo os conjuntos candidatos { {1, 2, 4 }, { 4, 5, 8 } } é alcançada. Antes de inserir o conjunto candidato na folha, é verificado o número de conjuntos candidatos já contidos. Com a inserção do conjunto candidato { 1, 5, 9 }, este número ultrapassa o limite especificado. Assim sendo, um novo nó é criado. Os conjuntos candidatos contidos na folha são redistribuídos nos novos nós internos criados. Depois da inserção do novo conjunto candidato { 1, 5, 9 }, a árvore *hash* passa a ser a árvore representada pela como ilustrado na figura 2.6.

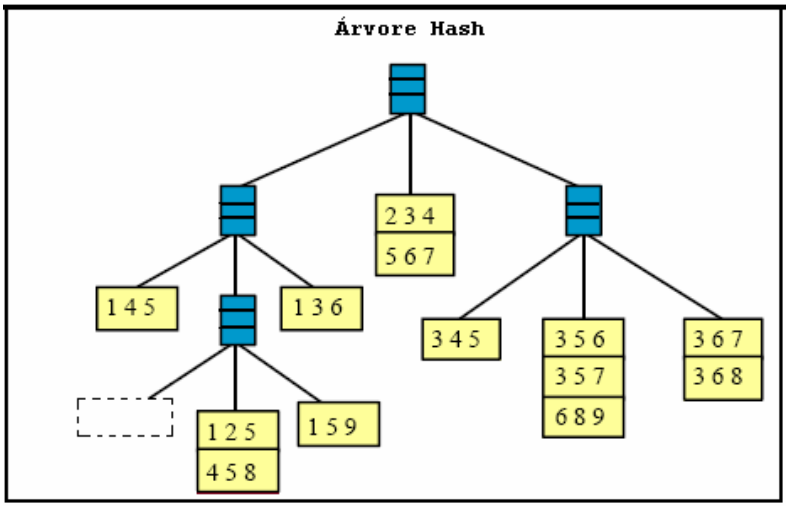


Figura 2.6 : Árvore *hash* após a inserção do candidato {1,5,9}

2.3.4 Contagem de Suporte

Para contagem da freqüência dos conjuntos candidatos, cada transação deve atravessar a árvore, iniciando pelo nó raiz. Isto é feito por meio de aplicação recursiva da função *hash* aos itens da referida transação, até que uma folha seja alcançada. O algoritmo encontra então, todos os candidatos presentes numa transação T, que é composta por uma seqüência ordenada de itens.

A partir do nó raiz a função *hash* é aplicada a cada item da transação. E ao chegar a um nó interno, por meio da aplicação desta função a um item, aplica-se novamente a função *hash* a cada item posterior de T, até que se alcance um nó folha. Verificam-se quais os conjuntos candidatos deste nó folha estão contidos na transação e incrementam-se os seus contadores. Em seguida, marca-se a folha como nó já visitado, para que este não mais seja visitado por esta transação. O processo se repete recursivamente, até que toda a transação seja processada.

A figura 2.7 ilustra como a transação {1,2,3,5,6} atravessa a árvore *hash* na contagem do suporte. O procedimento recursivo tem início, na raiz, com a aplicação da função *hash* ao primeiro item da transação, o item 1. Verifica-se, então, que os conjuntos candidatos iniciados com este item localizam-se no filho esquerdo do nó raiz. Assim, o segundo nível da árvore é alcançado com os itens da transação {2,3,5,6}. Neste caso, alcança-se um nó interno, e a função *hash* é aplicada ao próximo item da transação, o item 2. Então todos os candidatos que iniciam com os itens 1 e 2 estão na sub-árvore localizada no nó central alcançado. O terceiro e último nível da árvore, do exemplo, é alcançado com o processamento do primeiro item da transação {3,5,6}. Assim, ao aplicar a função *hash* no item 3, alcança-se o nó folha com o conjunto candidato {1,5,9}.

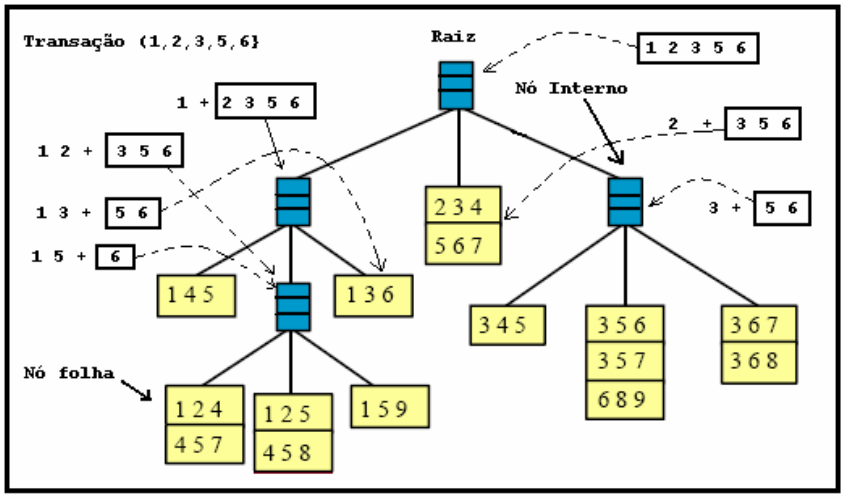


Figura 2.7 : Exemplo da contagem dos suportes

Quando uma folha é alcançada, todos os conjuntos candidatos contidos na folha que pertencerem à transação devem ter seus contadores atualizados. No exemplo em questão, o conjunto candidato {1,5,9} não tem seu contador incrementado, pois o

mesmo não pertence a transação T. Então a folha é marcada como visitada e aplica-se a função *hash* ao próximo item, o item 5. Desta forma, alcança-se outra folha com os conjuntos candidatos { {1,2,5}, {4,5,8} } e o conjunto candidato {1,2,5} tem seu contador incrementado, pois o mesmo pertence a transação. Continuando o processo, aplica-se a função *hash* ao item 6 e uma folha já visitada é alcançada. Neste ponto, todos os itens da transação já foram processados neste nível. Assim o procedimento recursivo retorna ao nível anterior.

Verifica-se que neste segundo nível ainda existem itens da transação a serem processados. Logo, aplica-se a função *hash* ao item 3, que é o item imediatamente seguinte ao item 2 já processado, neste nível, na transação {2,3,5,6}. Novamente, chega-se numa folha e o candidato {1,3,6} tem seu contador incrementado e a folha é marcada. Dando prosseguimento, aplica-se a função *hash* ao próximo item não processado, o item 5. A folha contendo o conjunto candidato {1,4,5} é alcançada e marcada como visitada. Então ao se aplicar a função *hash* ao último item da transação, ainda não processado, o item 6, chega-se a uma folha já visitada. Já tendo processados todos os itens da transação para este nível, então retorna-se a raiz.

Na raiz, o processo repete-se para os demais itens da transação {1,2,3,,5,6} ainda não processados, de modo que todas as folhas da árvore são visitadas e todos os conjuntos candidatos pertencentes a transação tem seu contador incrementado.

Este procedimento é executado para todas as transações que compõem a base de dados e no final desta etapa a árvore *hash* possui todos os conjuntos candidatos e seus respectivos suportes.

2.3.5 Geração de Regras

A geração de regras para qualquer *itemset freqüente* significa encontrar todos os subconjuntos não vazios de F_K. Assim, para todo e qualquer subconjunto *"a"* de *" f "*, produz-se uma regra, do tipo, $R = a \Rightarrow (f - a)$, somente se confiança da regra R é ao menos igual a confiança mínima estabelecida pelo usuário, ou seja, conf (R) ≥ conf_min. Para gerar regras com múltiplos conseqüentes, são considerados todos os subconjuntos. Por exemplo, dado um *itemset ABCD*, considera-se primeiro o

subconjunto *ABC*, seguido de *AB*, etc. Se $ABC \Rightarrow D$ não atinge uma confiança suficiente (confiança < *com_minf*), não é necessário verificar se $AB \Rightarrow CD$. No processo de geração de regras não é preciso revisitar a base de dados, pois a contagem do suporte já foi feita em fases anteriores do APRIORI.

2.3.5.1 Um exemplo de geração de regras de associação

O exemplo a seguir, demonstra como funciona a extração de regras de associação através do APRIORI. Suponha um banco de dados formado somente por um grupo de dados. Suponha, também, que este grupo seja composto por apenas cinco registros, associando-se a cada um desses registros cinco atributos categóricos, conforme mostrado na Tabela 2.1.

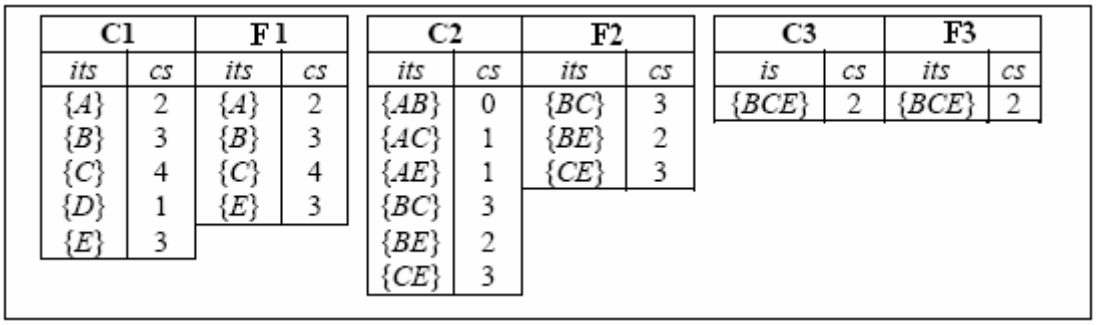
Seja C_K o conjunto de K-*itemsets* candidatos, onde K = 5. Cada membro "c_k" deste conjunto tem dois campos: *itemset* e *contador de suporte*, representados, respectivamente, por *its* e *cs* na figura 2.8. Seja F_K o conjunto dos K-*itemsets* freqüentes. De modo análogo, cada membro deste conjunto também possui *its* e *cs*.

TID	**Itens Categóricos**				
	A	**B**	**C**	**D**	**E**
114-1	1	0	0	1	0
114-2	0	1	1	0	1
114-3	1	0	1	0	1
114-4	0	1	1	0	1
114-5	0	1	1	0	0
freqüência	**2**	**3**	**4**	**1**	**3**

Tabela 2.1 - Tabela de dados

O primeiro passo do algoritmo conta a freqüência com que os itens ocorrem para determinar os 1-*itemsets* freqüentes (última linha da Tabela 2.1). Posteriormente, obtém-se o conjunto de candidatos 1-*itemsets*, C_1, mostrado na figura 2.8. Assumindo um suporte mínimo igual a dois, ou seja, *sup_min* = 40%, F_1 é composto pelos elementos de C_1 com suporte igual ou superior a 40%. No exemplo, somente o *itemset* D não atendeu a esta condição, ficando F_1 composto por {A}, {B}, {C} e {E}. Para descobrir o conjunto dos 2-*itemsets* freqüentes, de modo a continuar satisfazendo ao

suporte mínimo, o APRIORI usa o processo de junção $F_1 * F_1$, descrito na seção 2.3.2, para gerar o conjunto candidato C_2, que consiste de 2-*itemsets*. Por exemplo, {*C*} e {*E*} geram {*CE*}. Mais uma vez, cada ocorrência é computada. No caso {CE} ocorre três vezes. F_2 é determinado com base no suporte de cada candidato de C_2. Agora são excluídos {AB}, {AC} e {AE}, pois têm suporte inferior ao mínimo estabelecido.



C1		F 1	
its	*cs*	*its*	*cs*
{*A*}	2	{*A*}	2
{*B*}	3	{*B*}	3
{*C*}	4	{*C*}	4
{*D*}	1	{*E*}	3
{*E*}	3		

C2		F2	
its	*cs*	*its*	*cs*
{*AB*}	0	{*BC*}	3
{*AC*}	1	{*BE*}	2
{*AE*}	1	{*CE*}	3
{*BC*}	3		
{*BE*}	2		
{*CE*}	3		

C3		F3	
is	*cs*	*its*	*cs*
{*BCE*}	2	{*BCE*}	2

Figura 2.8 : Geração de Candidatos e Itens freqüentes

A geração de C_3 é obtida a partir de F_2 de uma maneira distinta. Os futuros *itemsets c*andidatos devem manter uma ordem lexicográfica, tal que quando a junção $F_2 * F_2$ for realizada, o primeiro item de um *itemset* seja idêntico ao primeiro item do outro *itemset* e assim sucessivamente. Porém, o último item do *itemset* deve ser menor, lexicograficamente, que o último item do outro *itemset.* Esta regra pode ser representada como:

$$\textbf{\textit{itemset\_ p} = \{}p_1, p_2, ..., p_n\textbf{\}, \textit{itemset\_q} = \{}q_1, q_2, ..., q_n\textbf{\}}$$

Tendo como condição necessária que: $p_1 = q_1,\ p_2 = q_2, ..., p_n < q_n$

Na figura 2.8, o *itemset* candidato {BCE}, em C_3, foi formado juntando-se {BC} com {BE}, pois B = B e C < E. Este foi o único conjunto que pôde ser formado, pois não há outra concatenação que satisfaça a condição necessária. A junção {BC}*{CE}, por exemplo, não satisfaz a condição, pois, lexicograficamente, p_1 = B é menor que q_1 = C.

O passo seguinte é descobrir as regras de associação. No caso do exemplo, supondo uma confiança mínima de 60% e mantendo o suporte mínimo em 40%, uma regra provável seria BC $\Rightarrow$ E. Para ela, a confiança é igual a: suporte (BCE) ÷ suporte (BC), cujo resultado é 66%, satisfazendo a condição imposta.

Para a regra BC $\Rightarrow$ E, seu suporte seria o percentual de ocorrências de BCE com relação ao total de dados, que resulta em 40%. Então, esta é uma regra válida. Outra provável regra seria B $\Rightarrow$ CE. Para esta situação, o valor da confiança seria idêntico, pois a razão suporte (BCE) $\div$ suporte (B) também é igual a 66% .

2.3.6 Informações Complementares

Diversas técnica de otimização foram propostas e colocadas em práticas para melhorar o desempenho do algoritmo APRIORI. A seguir serão descritas algumas destas técnicas. Entretanto, primeiramente, serão discutidos alguns detalhes de implementação referentes ao gerenciamento da memória.

2.3.6.1 Gerenciamento da memória

Caso haja espaço na memória principal para estocar os conjuntos de itens freqüentes F_{K-1} e de candidatos C_K , as fases geração e poda dos candidatos podem ser realizadas na memória principal. Do contrário, será preciso armazená-los em disco. Além de ser necessário armazenar os candidatos, na fase de contagem do suporte, é necessário ainda garantir um espaço na memória para as transações que serão testadas para avaliar os contadores de suporte de cada elemento de C_K. Portanto, a seguir será descrito como proceder no caso da memória principal ser insuficiente.

Suponha-se que F_{K-1} caiba na memória principal, mas o conjunto dos candidatos C_K não caiba. Neste caso, a fase de geração candidatos deve ser modificada. Gera-se o número máximo de candidatos que é possível de se armazenar na memória principal. Depois disto, poda-se estes candidatos. Se sobrar espaço, repete-se o processo de geração e poda de candidatos, até que não haja mais espaço na memória principal. Em seguida, inicia-se a fase de contagem dos suportes, varrendo-se uma vez a base de dados para contar os suportes destes candidatos. Aqueles candidatos classificados como freqüentes são armazenados em disco e os demais são suprimidos da memória. Este processo é repetido até que todos os candidatos sejam gerados e classificados como freqüentes ou não. Assim, quando não há espaço para todos os candidatos na memória principal, o algoritmo é obrigado a varrer a base de dados um número de vezes igual ao

número de partições em que deve ser dividido o conjunto dos candidatos, para que cada partição caiba na memória principal.

Agora suponha que o conjunto de itens freqüentes F_{K-1} não caiba na memória principal. Neste caso, não é possível fazer a poda dos candidatos antes de varrer a base para o cálculo do suporte. Entretanto, como os itens são enumerados, é possível associar cada item a um número inteiro positivo, ordená-los, lexograficamente, e armazenar F_{K-1} em disco. Então, carrega-se na memória principal um bloco de *itemsets* de F_{K-1} nos quais todos os K - 2 primeiros elementos são idênticos. Faz-se a junção dos elementos deste bloco e varre-se a base de dados para o cálculo do suporte destes elementos. Os que são freqüentes são armazenados em disco, os demais suprimidos. Repete-se o processo, até que todos os candidatos de tamanho K tenham sido gerados e seus respectivos suportes tenham sido calculados. Como um exemplo, suponha K = 4 e F_3 como ilustrado pela Tabela 2.2.

F_3		
1	2	3
1	2	4
1	2	5
1	2	6
1	2	7
1	3	4
1	3	5
1	3	6

Tabela 2.2 - *Itemset* freqüente F3

Suponha que na memória principal caibam no máximo 3 *itemsets* de tamanho 3, além do espaço que deverá ser reservado para as transações do banco de dados na fase da contagem do suporte. O primeiro bloco que será trazido para a memória será: {{1,2,3}, {1,2,4}, {1,2,5}}. A partir deste bloco serão obtidos os *itemsets* de tamanho 4: {{1,2,3,4}, {1,2,3,5}, {1,2,4,5}}e o suporte desses três *itemsets* é avaliado. Então, o processo se repete, mas desta vez trazendo para a memória principal os *itemsets* {{1,2,3}, {1,2,6}, {1,2,7}}. A partir deste bloco serão obtidos os *itemsets* de tamanho 4: {{1,2,3,6}, {1,2,3,7}, {1,2,6,7}}. No próximo passo, é trazido para a memória principal {{1,2,4}, {1,2,5}, {1,2,6}}. E assim por diante, até que todos os candidatos de tamanho 4 tenham sido gerados e seus respectivos suportes tenham sido calculados.

2.3.6.2 Técnicas de otimização do APRIORI

As técnicas de otimização do algoritmo APRIORI atuam de muitas maneiras distintas. Por exemplo, a utilização da árvore *hash*, para armazenagem de C_K, é uma técnica que diminui o número de candidatos que são testados para cada transação na fase do cálculo do suporte. Também atua na fase de poda, diminuindo o número de subconjuntos a serem testados para cada candidato.

Outra técnica de otimização é a diminuição do tamanho do banco de dados a cada iteração. Pois, uma transação que não contém nenhum *itemset* freqüente de tamanho K, não conterá com certeza nenhum *itemset* freqüente de tamanho K+1. Isto porque todo subconjunto de itens de um (K+1)-*itemset* freqüente deve ser freqüente, como mostrado na seção 2.1.2. Além disto, se uma transação contém um (K + 1)-*itemset* deverá obviamente conter todos os subconjuntos deste *itemset*. Logo, se a transação contiver um (K + 1)-*itemset* freqüente *f*, deverá conter todo K-*itemset* freqüente contido em *f*. Desta maneira, é possível diminuir o banco de dados de transações a cada iteração.

Ainda se pode otimizar o desempenho do APRIORI diminuindo-se o número de varridas do banco de dados. No algoritmo APRIORI clássico, o banco de dados é varrido a cada iteração. Há uma versão otimizada do APRIORI, denominada APRIORI_TID, que varre apenas uma vez a base de dados.

Outra maneira de se otimizar o algoritmo é diminuir o número de candidatos gerados. E isto pode ser conseguido quando se impõem restrições sobre as regras a serem obtidas. Na verdade, existem vários tipos de restrições que podem ser utilizados para se reduzir o conjunto de regras gerado. Por exemplo, se existe algum conhecimento prévio, que possa ser usado para restringir um ou mais itens da base de dados, tal restrição poderá resultar numa diminuição significativa do conjunto de candidatos a itens freqüentes a ser gerado, obtendo-se, assim, uma melhora na aplicação do algoritmo.

2.4 REGRAS DE CLASSIFICAÇÃO

Essa seção apresenta o conceito da classificação, a descrição geral do problema, as estratégias utilizadas pelos principais métodos de classificação e uma breve discussão sobre a relação entre regras de associação e regras de classificação.

2.4.1 Conceituação

Classificação é um problema antigo. O filósofo Aristóteles, considerado o criador do pensamento lógico, já no século IV AC tentou agrupar organismos em duas classes, os benéficos e os maléficos aos seres humanos. Introduziu também, o conceito de classe, ao classificar todas as formas de vida para organizar a rica diversidade em organismos vivos. Nos dias de hoje, os sistemas de classificação tentam encontrar características diferentes entre classes para usá-las na classificação de exemplos desconhecidos. É reconhecido como um problema importante na descoberta de conhecimento implícito em bases de dados [1]. E, nos últimos anos, atraiu grande atenção dos pesquisadores da área de mineração dos dados [10], [18], [19], [20], [21], [22].

O problema de classificação envolve descoberta de relações ou dependências entre variáveis de classe e outras variáveis, usando tais relações para classificar casos desconhecidos. Essas relações são armazenadas como modelos de classificação em forma de regras [23], [24], árvore de decisões [25] ou formulações matemáticas.

Os métodos tradicionais como árvore de decisão [25] e técnicas estatísticas [26], [27], [28] são baseados em heurísticas e algoritmos gulosos para a construção de classificadores. Essas técnicas constroem um conjunto de regras, as quais são usadas para predições de novos casos. As regras codificam relações entre as variáveis de classe e outras variáveis. Recentes estudos em classificação têm proposto maneiras de explorar o paradigma de mineração de regras de associação para realizar a tarefa de classificação. Esses métodos mineram regras de associação de alta qualidade e as usam para construir classificadores [10], [29], [30]. A metodologia é conhecida como classificação associativa. Classificadores associativos apresentam algumas vantagens, a saber:

(1) Os classificadores associativos lidam naturalmente com valores perdidos e "*outliers*", já que trabalham com associações estatisticamente significativas.

(2) A mineração de itens freqüentes captura todo o domínio das relações entre os itens na base de dados.

(3) Existem eficientes algoritmos de mineração de itens freqüentes e consequentemente de regras de classificação associativa.

(4) Os classificadores desenvolvidos sob a luz desta metodologia são robustos, já que apenas itens freqüentes robustos são aproveitados na sua construção.

(5) E estudos de desempenho dessa metodologia [18], [20] têm demonstrado que tais classificadores são altamente precisos e eficientes.

2.4.2 Descrição Geral

Considere uma base de dados relacional, onde cada coluna representa um domínio, chamado atributo, que possui um número finito de valores. Cada linha é chamada de registro, que é um conjunto de valores de atributos. Essa base de dados é chamada de base de treinamento se houver um atributo especial chamado classe. Do contrário, é chamada de base de teste.

A classificação é o processo de predição de classes dos registros em uma base de dados de teste. Considere um padrão como um conjunto de valores de atributos. Uma regra de classificação é a implicação de um padrão com uma classe. A regra é capaz de fazer a predição de um registro se seu antecedente for um subconjunto desse registro. Um classificador baseado em regras é composto de um conjunto de regras ordenadas por alguma norma de precedência e uma classe padrão (default). Para um registro de teste, usa-se a regra de mais alta precedência de modo a verificar se esta é um subconjunto do registro, ou em outras palavras, cobre o registro. Caso afirmativo, o registro em questão é classificado como sendo pertencente à classe representada pelo conseqüente da regra. Se nenhuma regra do classificador for capaz de cobrir o registro, este é classificado com a classe representada pela classe padrão.

2.4.3 Algoritmos Tradicionais de MRC

Os algoritmos de mineração de regras de classificação são usados para descobrir um simples conjunto de regras que melhor cubra uma base de dados de treinamento. Para melhor entendimento desses algoritmos a figura 2.9 ilustra o algoritmo de cobertura de regras [31], que é a base da maioria dos algoritmos de mineração de regras de classificação.

Num conjunto de regras de classificação existe pouca sobreposição entre as regras. Seja devido à remoção dos registros à medida que estes vão sendo cobertos pelas regras, ou devido a registros que suportam apenas uma regra. Assim um conjunto de regras gerado a partir de algum algoritmo baseado no algoritmo de cobertura, representa um conjunto cujas regras cobrem um conjunto disjunto, ou quase disjunto, de registros.

Algoritmo de Cobertura de Regras

Entrada: Uma base de dados de treinamento **D**
Saída: conjunto de regras **R**

```
1   faça  R = ∅
2   enquanto ( D ≠ ∅ )
3       encontre uma regra r de D heuristicamente
4       remova de D todos os registros cobertos pela regra r
5       insira a regra r ao conjunto R
6   fim-enquanto
7   Retorne R
```

Figura 2.9 : Algoritmo de Cobertura de Regras

Os algoritmos de mineração de regras de classificação são geralmente de dois tipos, os algoritmos de conjuntos com cobertura simultânea e os algoritmos com cobertura seqüencial.

Os algoritmos com cobertura simultânea, tais como ID3 e C4.5 [24], [25], primeiramente selecionam o melhor atributo, seleção feita por algum critério heurístico. Então a base de dados é particionada em subconjuntos de dados disjuntos, pela distinção do valor do atributo selecionado como o melhor. Assim, recursivamente, cada subconjunto encontrado é particionado de maneira semelhante à realizada no primeiro particionamento. Esse procedimento termina quando os subconjuntos de dados são puros, isto é, cada subconjunto contém apenas instâncias de uma mesma classe. Todos os atributos selecionados e seus valores distintos formam uma árvore de decisão, onde cada caminho da raiz a uma folha representa uma regra de classificação. Na prática, o conjunto final de regras é submetido a algum processo de poda e a árvore de decisão é otimizada.

Os algoritmos de cobertura seqüencial tais como AQ15 [32] e CN2 [33] buscam a melhor regra, isto é, a regra com a maior cobertura dos dados da base de treinamento. Esses algoritmos separam estas instâncias de registros e recursivamente buscam pela próxima melhor regra nos dados restantes da base de treinamento, até que não haja mais nenhum registro na base de treinamento. Neste processo também é utilizado algum critério heurístico para a escolha da melhor regra.

Existem diversos algoritmos de mineração de regras de classificação baseado no algoritmo de cobertura. Eles se distinguem pelo critério heurístico utilizado seja na escolha de atributos, seja na escolha de regras. Discussões adicionais sobre os algoritmos de mineração de regras de classificação são referenciadas em [34], [35].

A maioria dos algoritmos de mineração de regras de classificação apresenta alguns pontos fracos, a saber:

1) Sendo algoritmos baseados em alguma heurística, podem perder alguma regra ótima global.

2) Como as regras são retiradas de uma base de dados de treinamento podem ter problemas de ajuste, decorrente da amostra dos dados.

3) Normalmente estes algoritmos necessitam acessar a base de dados várias vezes, o que representa uma restrição para mineração em bases de dados de grande escala.

Nesta tese é apresentado um algoritmo com particionamento da base de dados para execução em ambiente computacional paralelo, visando possibilitar a aplicação desta tarefa em grandes bases de dado.

2.4.4 Métodos de Classificação

A tarefa de Classificação é um campo de pesquisa que envolve diverso tópicos, tais como, árvore de decisão [36], [25]; redes neurais [37], [38]; algoritmos genéticos [39]; classificação *bayesiana* [40]; etc. Nessa tese, os estudos serão concentrados em classificação baseado em associações, ou, como também é conhecida, classificação associativa [10].

2.4.4.1 Conjunto robusto de regras

Em bases de dados reais é comum existir atributos cujos valores foram perdidos. Um sistema robusto deve ser capaz de lidar com esse problema. Dados perdidos podem acontecer em bases de dados de treinamento ou em bases de dados de teste.

Para as bases de treinamento existem alguns métodos propostos. Sendo que o mais comum é quando há falta de valores de algum atributo, a solução mais simples é completar os valores perdidos pelo valor mais freqüente do atributo. Mais precisamente, a freqüência dos valores dos atributos é levantada para cada classe e o valor perdido é completado com o valor mais freqüente do atributo em cada classe. Esse método é usado em [41]. Outra solução é completar o valor perdido com um valor estimado baseado na probabilidade de freqüência entre diversos valores do atributo. Esse é usado em [25]. Enquanto que para as bases de teste onde há falta de valores em atributos, não existe nenhum estudo prévio, para solucionar o problema, segundo [54].

Dize-se então, que um conjunto de regras é mais robusto que outro, quando este pode fazer predições mais precisas, em uma base de teste que possua valores perdidos de atributos, do que o outro conjunto.

2.4.4.2 Busca sistemática

Um método de busca heurística encontra apenas um conjunto de possíveis regras que se ajuste com a base de treinamento. Mas o conjunto encontrado pode não ser o melhor. Embora seja possível extrair um conjunto de regras através de amostragens de registros da base de treinamento [42], [43], não existe nenhuma garantia de ser este o conjunto de regras ótimo. Essa fraqueza, das buscas heurísticas, tem sido notada e alguns algoritmos foram desenvolvidos para a busca de melhores regras.

Alguns algoritmos de busca sistemática tais como [44], [45], [46], [47] utilizado em algoritmos com estruturas de árvore de enumeração [48] podem encontrar de forma mais eficiente o melhor conjunto de regras de classificação. Entretanto, essas metodologias estão sujeitas a explosão combinatória para bases de dados com muitos atributos.

O algoritmo FSS [49] reorganiza a árvore de busca dinamicamente, de modo que os estados que levam a soluções diferentes da ótima pode ser maximamente podados. O algoritmo OPUS [50] é um melhoramento do FSS, pela introdução do algoritmo A* [51]. Ele é implementado com um mecanismo de poda baseado em aprendizagem de máquina. Embora OPUS seja bem eficiente em seu mecanismo de poda, este algoritmo busca apenas um conjunto de regras por vez. Sendo assim, torna-se muito custoso em termos computacionais, para encontrar o conjunto ótimo de regras. Quando a busca é apenas de um conjunto de regras satisfatório, esse algoritmo pode ser adaptado para mineração de regras de associação como em [52]. Todavia, essa versão modificada do algoritmo OPUS produz regras com alto suporte, o que não é bom para predição. Quando esse algoritmo é usado para geração de todas as regras de associação, apresenta desempenho inferior ao APRIORI, como mostrado em [53]. Além do mais o OPUS modificado necessita ler a base de dados tantas vezes quantos forem o número de diferentes antecedentes do conjunto de regras encontrado. O que não é eficiente quando a base de dados não residir na memória principal.

2.4.5 Relação entre Regras de Associação e Regras de Classificação

Aparentemente, regras de associação e regras de classificação são distintas nos seguintes aspectos:

1) Regras de Classificação são geradas a partir de uma base de dados relacional, para solucionar problemas de classificação. Enquanto que regras de associação são originalmente geradas a partir de uma base de dados transacional, para solucionar problemas de cesta de mercado.

2) Regras de classificação sempre têm conseqüentes pré-especificados (as classes) que nunca aparecem no antecedente de nenhuma outra regra de classificação. Enquanto que as regras de associação normalmente não possuem conseqüentes pré-especificados e o conseqüente de uma regra de associação pode ser o antecedente de uma outra regra de associação.

3) O objetivo da mineração de regras de classificação é obter um conjunto simples e preciso de regras. Enquanto que o objetivo da mineração de regras de associação é encontrar todas as regras que satisfaçam alguns parâmetros.

4) Os algoritmos de mineração de regras de classificação frequentemente utilizam critérios heurísticos e por isso não podem garantir que o conjunto de regras encontrado seja ótimo. Enquanto que a mineração de regras de associação utiliza métodos de regras sistemáticas que podem produzir conjuntos ótimos de regras.

A despeito dessas diferenças, as tarefas de mineração de regras de classificação e as de mineração de regras de associação são intimamente relacionadas. Por exemplo, uma base de dados relacional pode ser mapeada para uma base de dados transacional, quando os atributos numéricos são discretizados. Um conjunto de regras de classificação é um subconjunto de regras de associação com alguns conseqüentes especificados, quando o suporte mínimo é baixo.

Algoritmos de mineração de regras de classificação têm requisitos implícitos de precisão mínima, o que pode ser contornado com ajustes no valor do suporte mínimo em mineração de regras de classificação baseado em regras de associação. Portanto, é possível utilizar técnicas de mineração de regras de associação para resolver problemas de classificação. Em [10], [11] e [12] é mostrado a construção de um classificador baseado em associação, o CBA, que apresenta resultados mais precisos que os classificadores tradicionais. Outros trabalhos nessa direção aparecem em [18].

Embora tenha-se visto que a classificação baseada em regras de associação apresenta vantagens sobre os métodos de classificação tradicionais, é importante ressaltar que as bases de dados relacionais são mais densas do que as bases de dados transacionais. Isto é, nas bases relacionais existe mais correlação entre os dados. O que eleva o risco de explosão combinatória nas aplicações de mineração de regras de associação. Além do que, a distribuição de classes é desigual na maioria das bases de dados relacionais. Portanto, a precisão e eficiência do classificador associativo (ou baseado em regras de associação) são altamente dependentes de ajustes nos valores dos parâmetros suporte mínimo e confiança mínima. Outra importante limitação apresentada pelos classificadores associativos é que estes só lidam com atributos discretos. Logo, a aplicação deste tipo de classificador em bases de dados que apresentem atributos contínuos, exige uma etapa de pré-processamento, a discretização destes atributos. Este processo, também pode influenciar fortemente a precisão do classificador.

Capítulo III

Banco de Dados e Algoritmos

Este capítulo apresenta os trabalhos correlatos aos problemas estudados nessa tese. Começa com a descrição de algumas características que influenciam o desempenho dos algoritmos de mineração de regras de associação. Em seguida apresenta um sumário das principais abordagens seqüenciais para o problema. Então, apresenta as estratégias conhecidas para geração de regras de classificação baseada em associações, detalhando principalmente o algoritmo CBA. E, finalmente, é feita uma revisão das estratégias paralelas para mineração de itens freqüentes, destacando os principais algoritmos paralelos e suas características.

3.1 METODOLOGIA DE GERAÇÃO DE CANDIDATOS

Essa seção apresenta uma discussão de como a metodologia utilizada na geração de *itemsets* candidatos influência o desempenho dos algoritmos de mineração de regras de associação.

Os algoritmos de mineração de regras de associação podem diferir na maneira pela qual geram os novos candidatos a itens freqüentes. A geração de candidatos, baseada numa investigação completa do espaço de busca, garante que todos os subconjuntos de itens mais freqüentes são gerados e testados. Por isso, esta é a opção mais adotada pelos algoritmos de mineração de regras de associação. Entretanto, em

função de garantir maior velocidade na geração dos candidatos, algumas abordagens utilizam-se de métodos heurísticos, que não investigam todo o espaço de busca para a geração de candidatos. Entretanto, estes algoritmos podem deixar de gerar regras importantes, já que nem todos os *itemsets* são enumerados.

Também é possível uma busca randômica para localizar o *itemset* freqüente maximal, já que alguns algoritmos, ao invés de fazer uma enumeração completa de todos os *itemsets*, fazem uma enumeração apenas dos *itemsets* maximais. Logo, pode-se notar que outro ponto importante e que merece destaque no processo de busca de itens freqüentes e, conseqüentemente, na mineração de regras de associação é a solução particular adotada pelos diferentes algoritmos para a geração dos candidatos.

3.2 REPRESENTAÇÃO DO BANCO DE DADOS

Nessa seção é feita uma explanação de como os dados podem ser formatados para sua utilização nos algoritmos de mineração de regras de associação, apontando as vantagens e desvantagem de cada tipo de formatação de dados.

No processo de descoberta de conhecimento em bancos de dados, a segunda etapa, o pré-processamento, fornece o conjunto de dados já em um formato apropriado, que será utilizado pelos algoritmos de mineração de regras de associação. Tal formato, que precisa ser considerado na projeção de sistemas de mineração, pode ser, basicamente, de dois tipos: horizontal ou vertical [55], [60], como mostra a figura 3.1.

3.3.1. Formato Horizontal

Neste tipo de formato, os dados estão organizados por transação, isto é, cada transação é composta por uma lista de itens, que são representados por algum identificador único do item. Cada transação possui um TID, identificador da transação.

Um grande problema da representação dos dados em formato horizontal é que o número de colunas pode ficar muito grande. Para alguns casos, por exemplo, o número de itens pode exceder a cem mil, neste caso, itens similares precisam ser agrupados para

a diminuição do número de colunas a valores razoáveis. Além do mais, ao se utilizar este formato de representação, não é possível calcular o suporte de um determinado item ou conjunto de itens sem fazer uma passagem completa por toda a base de dados.

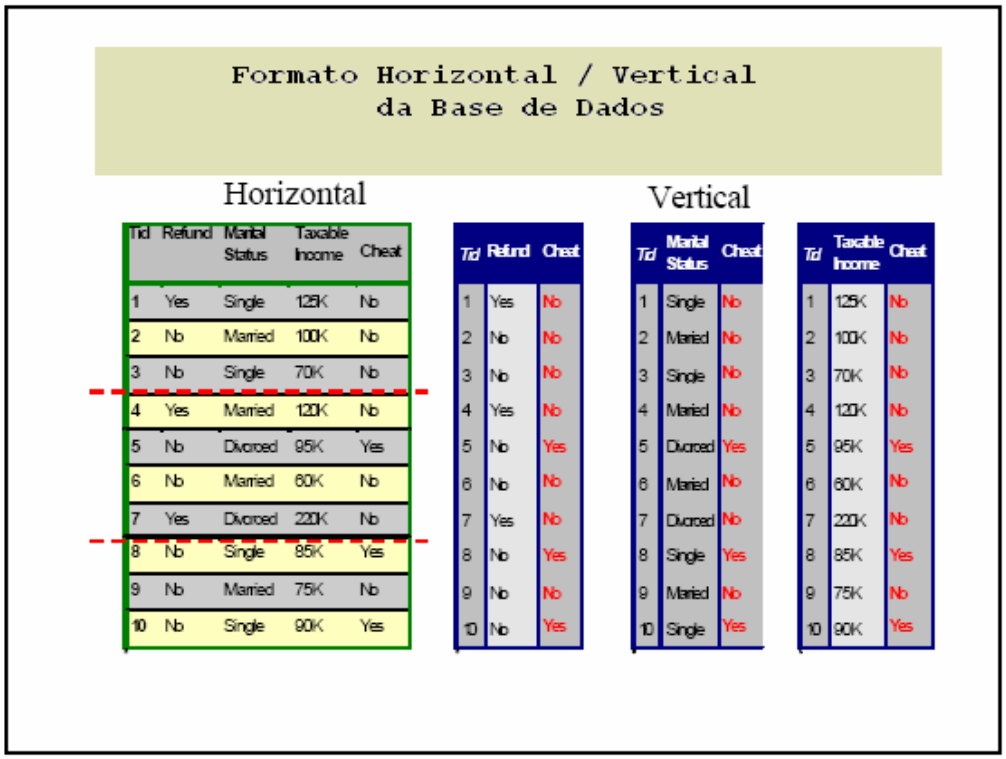


Figura 3.1 : Formato horizontal e vertical do banco de dados

3.3.2. Formato Vertical

O formato vertical representa os dados utilizando uma lista de itens, onde cada item é seguido por sua lista de transações. Ao contrário do formato horizontal, não é necessária a passagem por todo o banco de dados para calcular o suporte de um item. Como cada lista de itens possui todas as informações relevantes sobre um item, pode-se determinar o suporte pelo número de transações que compõem a lista. Para determinar o suporte para dois itens, basta realizar, através de uma operação simples de intersecção, a contagem das transações contidas tanto em um quanto em outro item.

Como desvantagem do formato vertical, porém, está o fato de que ele torna mais oneroso o exame de todos os *itemsets* freqüentes pequenos. Isto se deve ao fato de que a lista de transações utilizada não fornece nenhuma informação sobre a associação entre

itens. Por exemplo, para um banco de dados no qual seiscentos 1-*itemsets* são encontrados na primeira passagem. A combinação de todos estes conjuntos vai gerar $600^2/2$ candidatos, o que é igual a 180.000 candidatos. Portanto serão necessárias 180 mil interseções, o que consumirá a maior parte do processamento [62].

3.3 ESTRUTURAS DE ARMAZENAMENTO DE DADOS

Essa seção comenta a influência das estruturas de dados nos algoritmos de mineração de regras de associação, e apresenta as duas principais formas de armazenamento de dados utilizadas por esses algoritmos.

É de grande influência para os algoritmos de mineração de regras de associação a forma pela qual os dados coletados são armazenados, pois, devido as grandes quantidades de dados envolvidas neste problema, a forma de acesso a estes dados pode, inclusive, inviabilizar o método proposto, o que já justificaria a utilização de estruturas de dados eficientes para a manipulação destes.

Processos internos aos algoritmos geram grandes quantidades de dados, na sua maioria *itemsets* candidatos a tornarem-se *itemsets* freqüentes através da contagem dos suportes realizada nas bases de dados. Consequentemente, os processos de geração de candidatos e o de contagem dos suportes são diretamente dependentes da quantidade de dados, das estratégias utilizadas em cada processo e, sobretudo, das estruturas de dados, que podem proporcionar formas otimizadas de se realizar tais tarefas. Portanto, as estruturas de armazenamento de dados são, para esta tarefa de mineração de regras, um dos objetos de pesquisa mais importante.

Algumas formas distintas de estruturas de dados foram propostas para essa aplicação. Entretanto, dentre as estruturas de dados utilizadas para essa tarefa, as de maior destaque em uso são a árvore *hash*, utilizada primeiramente pelo APRIORI [3] e a árvore de prefixos, em [63].

3.4.1. Árvore *Hash*

A figura 3.2 ilustra a árvore *hash*, já apresentada no capítulo 2. A maioria dos algoritmos baseados no algoritmo APRIORI utiliza essa estrutura para armazenagem dos conjuntos candidatos e contagem de seus suportes. Na verdade, os candidatos são armazenados nas folhas. O propósito dos nós internos é apenas direcionar, através de uma tabela *hash*, a busca de um candidato a folha adequada, de modo a otimizar a tarefa de contagem dos suportes.

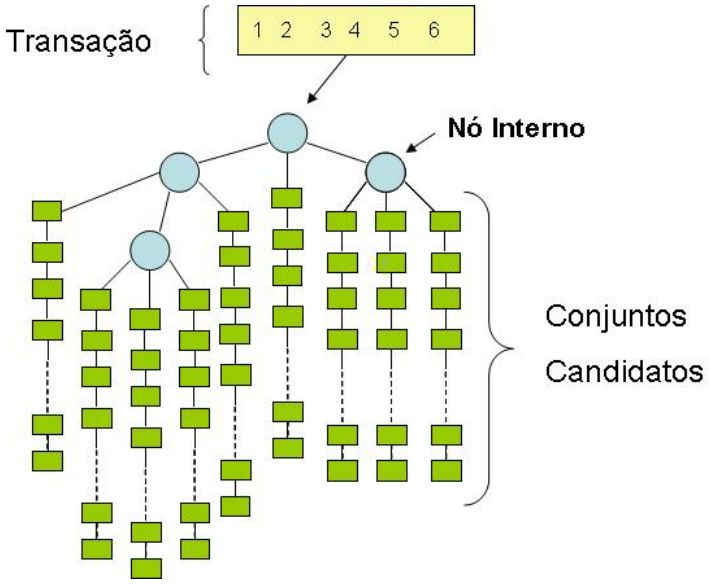


Figura 3.2 : **Árvore *Hash***

3.4.2. Árvore de Prefixos ou *Trie*

A árvore de prefixos ou *trie* é uma estrutura especializada no armazenamento de cadeias de caracteres e suporta uma rápida busca de padrões. A aplicação principal dessa estrutura é na recuperação de informação. De fato, o nome *trie* vem da palavra *retrieval* (recuperação). As operações primárias de consulta suportadas por *tries* são busca de padrões e busca de prefixos, daí o outro nome, árvore de prefixo.

Diferentemente da árvore *hash*, a árvore de prefixo é uma estrutura que não faz nenhuma distinção entre um nó interno ou um nó folha. Nesta estrutura os nós não

contêm conjuntos, mas apenas informações sobre os conjuntos, por exemplo, a contagem de suporte. Cada aresta da árvore é rotulada com um item e cada nó contém a informação para o conjunto de itens que rotulam as arestas de seu caminho até a raiz. A figura 3.3 mostra um exemplo, onde o conjunto { 1, 3, 4 } é marcado com um caminho hachurado. Os itens num certo conjunto X podem ser compreendidos, como uma descrição do caminho, a partir da raiz para alcançar o nó X. A ordenação dos itens é requerida de modo a se evitar a ambigüidade; do contrário, muitos nós corresponderiam ao mesmo conjunto.

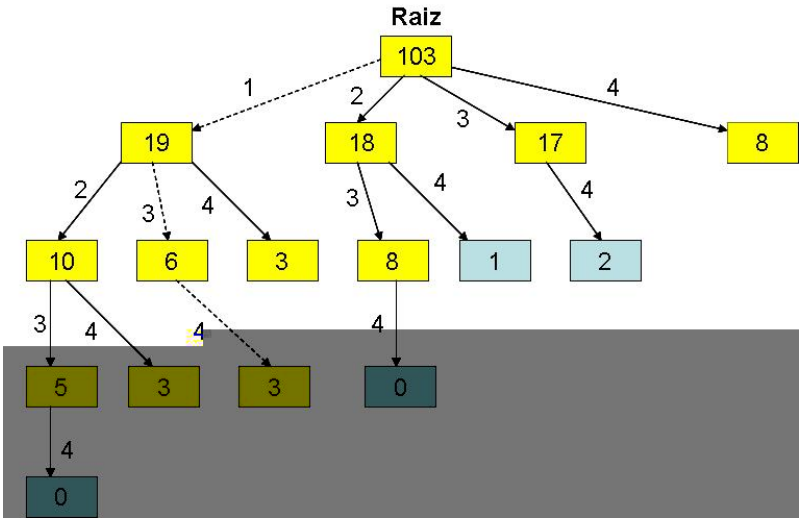


Figura 3.3 : **Árvore de Prefixo**

Outra diferença entre as duas estruturas é que a árvore de prefixos armazena ambos os conjuntos, itens freqüentes e candidatos, na mesma árvore. Uma vez computado o suporte de um candidato, altera-se o valor de seu contador. Entretanto, este, simplesmente, mantém-se na sua posição na árvore, ao ser considerado um *itemset* freqüente. Na árvore de prefixos todo nó contém contadores de suporte para seu conjunto correspondente. A raiz representa o conjunto vazio. Logo, seu contador é igual ao número de transações da base de dados, já que todas as transações suportam o conjunto vazio. Os conjuntos candidatos têm os contadores zerados antes de serem processados. A figura 3.3 mostra uma árvore completa, contendo todos os subconjuntos

de { 1, 2, 3, 4 }. Mas sob condições normais apenas os conjuntos freqüentes e conjuntos candidatos seriam armazenados na árvore.

3.4 METODOLOGIA DE BUSCA.

Esta seção apresenta as três principais abordagens para a tarefa de busca de itens freqüentes, adotadas pela maioria dos algoritmos de mineração de regras de associação.

A busca de itens freqüentes é a tarefa de maior complexidade computacional, dentre as demais tarefas que compõe os algoritmos de mineração de regras de associação. Daí, desde a sua introdução em 1993, como em [2], muitos pesquisadores vem envidando esforços na pesquisa de algoritmos mais eficientes para o problema. Consequentemente, muitos algoritmos com diferentes estratégias já foram propostos [3], [13], [65], [67]. Entretanto, independente das pequenas peculiaridades de cada algoritmo, pode-se identificar três principais abordagens com idéias centrais distintas.

A primeira, e a mais explorada delas, utilizada em [3], propõe um percurso iterativo e por níveis a todos os *itemsets*. Durante cada iteração, correspondente a um nível, um conjunto de candidatos é gerado através da junção de conjuntos freqüentes descobertos durante uma iteração prévia. Em seguida, é feita a contagem dos suportes dos *itemsets* candidatos e os *itemsets* considerados infrequentes são descartados.

A segunda abordagem é baseada na busca de *itemsets* freqüentes maximais [68]. Nessa abordagem não é necessário examinar explicitamente os *itemsets* simples, reduzindo potencialmente o número de candidatos examinados para encontrar os *itemsets* freqüentes. Entretanto, a determinação de *itemsets* freqüentes maximais não é suficiente para a geração das regras de associação, já que não se conhece o suporte associado a cada subconjunto dos *itemsets* maximais. Pode-se dizer, portanto, que é este o principal aspecto negativo dessa estratégia, tendo em vista que é necessária uma varredura completa na base de dados a fim de se computar a freqüência de cada subconjunto.

E a terceira abordagem baseia-se no conceito de "*closed set*" que tem sua descrição formal em [66] e foi introduzido para o problema de mineração de regras de associação em [64], [65]. Nesta estratégia não é preciso, necessariamente, examinar todos os *itemsets* freqüentes. São examinados apenas os chamados FCI ou " *Frequent Closed Itemset*", que é o menor conjunto representativo de todos os *itemsets* freqüentes, sem perda de informação. Assim, todos os *itemsets* freqüentes, bem como seus suportes, são gerados a partir dos FCI, sem necessidade de acessar a base de dados. Logo, nem todos os *itemsets* são explicitamente examinados e nenhuma varredura adicional na base de dados é necessária para a geração de regras de associação.

3.5 ALGORITMOS SERIAIS.

Nesta seção é feita uma breve descrição dos principais algoritmos seriais para as tarefas de mineração de regras de associação e de classificação baseado em associações, onde é detalhado o algoritmo CBA. Este algoritmo juntamente com o algoritmo APRIORI, detalhado no capítulo 2, servem de base para a implementação apresentada nesta tese.

3.5.1. Principais Algoritmos de MRA

O primeiro algoritmo para mineração de itens freqüentes foi o **AIS** [2]. Este algoritmo constrói conjuntos candidatos a partir dos itens que compõe as transações da base de dados, o que resulta em um grande número de conjuntos candidatos gerados. O algoritmo **SETM** [62], proposto no mesmo ano, foi desenvolvido em SQL e utilizava o mesmo princípio do AIS para a geração de candidatos.

O grande marco das pesquisas nesta área, surgiu com a introdução do **APRIORI** [3], que reduziu drasticamente o espaço de busca. Este algoritmo utiliza um padrão de busca "*bottom-up*" no espaço de busca e trabalha com dados no formato horizontal. Além disto, enumera todos os itens freqüentes, é do tipo iterativo e conta os *itemsets*, de comprimento definido em cada iteração, sobre a base de dados, como detalhado no capítulo 2. O desempenho deste algoritmo depende, entre alguns fatores, da geração de candidatos e sua estrutura de armazenamento de dados. O algoritmo faz tantas leituras da base de dados quanto a maior cardinalidade dos conjuntos de itens freqüentes. É tido até hoje, como um dos melhores algoritmos para mineração de regras de associação.

Surgiram então, algumas variações do APRIORI, como o **APRIORI-TID** [3]. A idéia desse algoritmo é criar uma relação especial (candidatos x transação), utilizando formato vertical de dados, de modo a diminuir o número de inspeções à base de dados. Embora, essa estratégia reduza este número para apenas 2 varreduras da base de dados, o consumo de memória para manter a relação de candidatos por transação pode ser muito elevado. Então, foi proposto o **APRIORI-HIBRIDO** [69]. Esse algoritmo baseia-se na idéia de que não é necessário o uso do mesmo algoritmo em todos os passos sobre os dados. Logo, nas etapas iniciais ele utiliza a estratégia do APRIORI e a certo momento de sua execução passa a utilizar a estratégia do APRIORI-TID.

O sucesso do APRIORI foi tão grande que além das variações descritas, este serviu de base para um grande número de algoritmos desenvolvidos para a tarefa de mineração de regras de associação.

O algoritmo **DHP** (*Dynamic Hashing and Pruning*), proposto em [70] e [71], é na verdade uma extensão do APRIORI. Sua estratégia tem como principal característica a redução do número de candidatos e da base de dados. Ela utiliza uma tabela *hash* para o armazenamento da pré-contagem de suportes. Deste modo, numa próxima iteração é necessário contar apenas o suporte de candidatos que se encontram nas células *hash*. O uso desta técnica permite a remoção prévia de muitos candidatos que se tornariam infrequentes. A redução da base de dados é feita progressivamente a cada passo do algoritmo com o objetivo de reduzir o tempo de sua leitura.

Um outro algoritmo, também baseado no APRIORI, denominado **PARTITION** minimiza acessos de E/S ao ler a base de dados somente duas vezes [72]. Este particiona a base de dados em pequenos pedaços, os quais podem ser processados na memória principal. Na primeira leitura, são gerados todos os potenciais conjuntos de itens freqüentes, os quais são confirmados na segunda leitura através da contagem de seus suportes. Após a leitura de cada partição são formadas listas verticais, com todas as transações onde cada item aparece. Então são gerados todos os *itemsets* freqüentes locais. Une-se, assim, todos os

sobre a base de dados para se obter o suporte de todos os candidatos e formar o conjunto global de *itemsets* freqüentes.

Ainda baseado no APRIORI, entretanto visando investigar a influência de estruturas de dados na performance do algoritmo, algumas idéias foram propostas. Entre elas está o algoritmo **SEAR** (Sequencial Efficient Association Rules) [63], idêntico ao APRIORI proposto em [3], apenas a estrutura de armazenamento é diferente. Neste algoritmo, os candidatos são armazenados numa árvore de prefixos, detalhada anteriormente neste capítulo, e não na estrutura original, a árvore *hash*. Também proposto em [63] o algoritmo **SPEAR** (*Sequential Partitioning Efficient Association Rules*), é similar ao SEAR, mas utiliza a técnica do particionamento da base de dados. Diferentemente do PARTITION, que utiliza o formato vertical de dados, o SPEAR utiliza formato horizontal. Mas também, faz apenas duas varreduras na base de dados. Primeiramente, coleta os itens potencialmente freqüentes e então em uma segunda investigação da base de dados obtém os suportes globais destes candidatos. O objetivo da implementação é de avaliar os benefícios intrínsecos do particionamento dos dados. Foi concluído em [63] que o particionamento dos dados não ajudou, devido ao alto custo de processamento das múltiplas partições e devido ao surgimento de grande quantidade de falsos positivos, isto é, *itemsets* freqüentes locais, que na verdade eram infrequentes quando analisados sob o ponto de vista global.

Como uma generalização do APRIORI, pode-se citar o algoritmo **DIC** (*Dynamic Itemset Counting*) proposto por [73]. Este conta dinamicamente candidatos de tamanhos variados à medida que a leitura da base acontece, conseguindo assim reduzir o número de acesso aos dados. Em particular, uma outra abordagem para minimizar o custo de E/S desse processamento é trabalhar somente com uma amostra da base de dados. A base de dados é dividida em partições de mesmo tamanho de modo que cada partição caiba totalmente na memória. Então, para a partição 1, o algoritmo coleta os suportes dos itens. Os itens localmente freqüentes geram conjuntos candidatos de tamanho 2. Logo, ao ler a partição 2, o algoritmo obtém os suportes para todos os candidatos correntes, isto é, os itens e os candidatos de tamanho 2. Este processo se repete para todas as partições restantes. O algoritmo então, inicia a contagem de candidatos de tamanho k enquanto está processando a partição k na primeira inspeção da base de dados. Após a última partição ter sido processada o algoritmo retorna a partição 1. Neste

ponto o suporte global dos candidatos é conhecido, uma vez que o algoritmo retornou a base de dados e alcançou a partição onde eles foram gerados pela primeira vez. O DIC é efetivo na redução do número de consultas a base de dados, se a maioria das partições for homogênea, isto é, possuírem uma distribuição de itens freqüentes similar. Caso os dados não sejam homogêneos, este algoritmo pode gerar muitos falsos positivos e então, realizar mais inspeções à base de dados que o próprio APRIORI. Pode-se utilizar a técnica de particionamento randômico da base de dados para minimizar o problema causado pela má distribuição dos dados.

Mesmo com diferentes peculiaridades os algoritmos citados até agora aplicam a mesma estratégia, isto é, buscam os *itemsets* freqüentes através um percurso iterativo e por níveis. Entretanto, como visto na seção 3.5, existem outras abordagens para tratar o problema de mineração de regras de associação. Por exemplo, estratégias que buscam diretamente os *itemsets* freqüentes maximais e a partir destes extraem seus subconjuntos. Entre os algoritmos que utilizam esse tipo de abordagem, pode-se destacar o **ALL-MFS** (*ALL Maximal Frequent Sets*) [74], que utiliza uma busca randômica para descobrir os *itemsets* freqüentes maximais; o **MAXMINER**, que usa eficientes técnicas de poda para rapidamente reduzir o espaço de busca [75] e também o **GENMAX** [78], entre outros.

Como a descoberta de *itemsets* freqüentes maximais não é suficiente para o problema de geração de regras de associação, já mencionado na seção 3.5, alguns algoritmos utilizam uma estratégia híbrida. Uma família desses algoritmos, apresentada em [55], [56], [57], tanto buscam todos os *itemsets* freqüentes como também buscam os *itemsets* freqüentes maximais. E dependem do formato da base de dados, da técnica de decomposição adotada, além do método de busca escolhido. São eles: **ECLAT** (*Equivalence Class Transformation*), **MAXECLAT**, **CLIQUE** e **MAXCLIQUE**. Estes algoritmos se caracterizam pelo uso dos dados no formato vertical; por adotarem uma abordagem teórica baseada em reticulados e grafos para decompor o espaço de busca original em menores pedaços (subreticulados). Neste caso, duas técnicas são utilizadas: partição por prefixo ou por clique maximal; e ainda, pela independência entre a tarefa de decomposição e a busca por padrões freqüentes. Eles precisam basicamente de uma única leitura na base de dados, minimizando assim custos de E/S. E são particularmente eficientes quando os *itemsets* freqüentes a serem descobertos são longos.

Outro algoritmo, também de grande relevância para esta tarefa, é o **FP-GROWTH** introduzido em [76]. Ele utiliza uma abordagem distinta das demais, pois processa primeiramente os *itemsets* mais freqüentes. Além de realizar a busca de *itemsets* freqüentes sem a geração de candidatos. Este algoritmo usa um esquema de eliminação recursiva de *itemsets* e adota como estrutura de dados uma variação da árvore de prefixo, denominada *FP-tree*. Desta forma, economiza uma considerável quantidade de memória para armazenamento das transações da base de dados. As vantagens oferecidas pelo algoritmo são parcialmente decorrentes do processo de ordenação dos dados em ordem decrescente de freqüência, o que reduz o tamanho da representação da base de dados e, consequentemente, o tempo de processamento, pois permite que os itens mais freqüentes sejam processados de forma mais eficiente.

Na maioria das bases de dados transacionais, especialmente para o problema de "*Cesta de Mercado" (Market Basket)*, os dados são pouco associados. Entretanto, para outras bases de dados onde os algoritmos de regras de associação podem ser aplicados ou nos casos onde o suporte mínimo é muito baixo, o número de *itemsets* freqüentes pode ser muito elevado. Tal fato levou os pesquisadores a adotarem uma outra abordagem para lidar com o problema, isto é, a utilização de estratégias para condensar os dados. Estas estratégias são baseadas no conceito de "*closed set*" [66]. Nos últimos anos foram desenvolvidos alguns algoritmos para mineração de regras de associação, utilizando essa estratégia. Pode-se citar como representantes deste grupo de algoritmos, o **A-CLOSED** [65] e **CHARM** [58], que apresentam como característica principal a mineração de FCI (*Frequent Closed Itemsets*), mencionado na seção 3.5. Esses algoritmos utilizam formato vertical de dados, metodologia híbrida para realizar busca e fazem apenas uma varredura na base de dados para todo o processo de geração de regras de associação.

3.5.2. Principais Algoritmos de MRC por Associação

A construção de classificadores baseados em regras de associações envolve quatro etapas distintas. Mineração do conjunto completo de regras de classificação por associação (ou RCA's), poda ou redução desse conjunto de RCA's, construção do classificador e verificação de sua precisão e eficiência.

Os algoritmos de mineração de regras de associação encontram todas as regras numa base de dados, que satisfaçam os limites de suporte e confiança. Como mostrado no capítulo 2, essas regras são da forma $X \Rightarrow Y$, onde X é o antecedente da regra e Y o conseqüente. As RCA's são definidas como regras de associação onde seus conseqüentes são variáveis de classe. Então, para uma regra de associação ser considerada uma RCA em uma base de dados, ela deve satisfazer duas condições: (1) os valores de suporte e de confiança da regra devem ser iguais ou superiores aos parâmetros suporte mínimo e confiança mínima, respectivamente; (2) o conseqüente da regra deve ser uma variável de classe.

Os algoritmos para mineração de regras de classificação geralmente são derivados de modificações de algoritmos conhecidos para mineração de regras de associação. Uma vez fixados os parâmetros suporte mínimo e confiança mínima, a saída de todos os algoritmos de mineração de regras de classificação é a mesma, isto é, o conjunto completo das RCA's. Assim, a primeira etapa, a mineração de regras, não altera a precisão do classificador, embora esta esteja sujeita aos valores de suporte mínimo e confiança mínima pré-estabelecidos. Portanto, pode-se dizer que a precisão dos classificadores associativos é dependente, sobretudo, da etapa do algoritmo responsável pela seleção das regras que compõe o classificador.

Existem muitos algoritmos eficientes para mineração de regras de classificação, como é visto a seguir. Porém, como o classificador paralelo apresentado nesta tese é baseado no algoritmo CBA, sua apresentação será mais detalhada do que os demais algoritmos.

3.5.2.1. O algoritmo CBA

O algoritmo **CBA** [10] foi o primeiro a definir um processo de construção de um classificador utilizando regras de associação. Este algoritmo consiste em duas partes, um gerador de regras, denominado **CBA-RG**, baseado no algoritmo APRIORI para a mineração das regras de associação e um construtor de classificador, denominado **CBA-CB**.

A operação chave do CBA-RG é encontrar todas RCA's que tenham suporte acima ou igual do suporte mínimo. O suporte de uma RCA é o número de casos, em uma base de dados D, que contém o antecedente da regra e é rotulado pelo seu conseqüente. As RCA's que satisfazem ao suporte mínimo são denominadas de RCA's freqüentes, enquanto o restante são as infreqüentes. Para todas as RCA's freqüentes que têm o mesmo conseqüente, a RCA com a mais alta confiança é escolhida como regra provável (ou RCA freqüente provável), e é a representante deste subconjunto de regras. Se existir mais do que uma RCA com a mesma mais alta confiança, seleciona-se uma delas randomicamente. A figura 3.4 mostra o algoritmo CBA-RG.

As linhas 1 e 2 representam o primeiro passo do algoritmo. Contam-se as ocorrências de itens e de classes para se determinar F_1, conjunto de item-regras[1] freqüentes de tamanho 1 (linha 1). A partir do conjunto F_1 são geradas as RCA_1 através da função *genRules*( ) (linha 2).

Para cada passo subseqüente, diz-se passo k, o algoritmo realiza quatro operações principais. Os item-regras freqüentes F_{k-1} encontrados na iteração anterior são usados para gerar os item-regras candidatos C_k, através da função *candidateGen*( ) (linha 4). Então, varre-se a base de dados e atualiza-se o suporte dos itens-regras candidatos em C_k (linhas 5-11). Após os novos itens-regras frequentes terem sidos identificados, forma-se o conjunto F_k (linha 12). O algoritmo então, produz as regras RCA_k, através da função *genRules*( ).

[1] Denomina-se item-regra a representação numérica de uma RCA. Um item-regra de tamanho k é na verdade um conjunto de inteiros com k+1 elementos, onde os k primeiros elementos representam o antecedente da regra e o último elemento o consequente, ou classe da regra .

Algoritmo CBA - RG

```
1    F1
2    RCA = genRules( F1 )
3    for ( k = 2 ; Fk-1 ≠ ∅ ; K ++ )
4        Ck = candidatoGen ( Fk-1 )
5        para cada caso d ∈ D faça
6            Cd = ruleSubset (Ck ,d)
7            para cada candidato c ∈ D faça
8                c.condsupCounter++
9                se d.class = c.class então c.rulesupCount ++
10           fim
11       fim
12   Fk = { c ∈ Ck | c.rulesupCount ≥ supmim}
13   RCA = genRules( Fk )
14   fim
15   RCAs= ∪k RCAk
```

Figura 3.4 : Algoritmo CBA-RG

A função *candidateGen*( ) é similar a função de geração de candidatos do algoritmo APRIORI. A função *ruleSubset*( ) toma um conjunto de itens-regras candidatos, C_k, e um caso "*d*", para encontrar todos os itens-regras em C_k cujo antecedente é suportados por "*d*". Essa operação e as demais nas linhas 7-9, também são similares as realizadas no algoritmo APRIORI. A diferença é que, neste caso, é necessário incrementar o contador de suporte dos antecedentes (*condsupCounter*) e o contador de suporte dos item-regras (*rulesupCount*) separadamente, enquanto que, no APRIORI, apenas um contador de suporte é atualizado. Porém, isto permite que as confianças das regras sejam calculadas.

A segunda etapa do algoritmo CBA é a construção do classificador. Para isso foi desenvolvido o CBA-CB, algoritmo de construção do classificador, que utiliza as RCA's produzidas pelo CBA-RG. Para produzir o melhor classificador dentre todo o conjunto de RCA's encontrado pelo CBA-RG seria necessária a avaliação de todos os subconjuntos de RCA's possíveis em uma base de dados de treinamento, de modo a

selecionar o subconjunto de RCA's com a seqüência correta de regras, que fornecesse o menor erro. Todavia, existem 2^m subconjuntos de RCA's, onde "*m*" é o número de regras, não mencionando as diferentes possíveis seqüências de regras. Isto é claramente inviável, haja vista que o número de regras sempre é muito elevado. O algoritmo CBA-CB é heurístico, mas como mostrado em [10] o classificador produzido por este algoritmo apresenta desempenho muitas vezes superior a maioria dos classificadores construído sob outros paradigmas.

Antes da apresentação do algoritmo CBA-CB, é necessário definir a ordenação total das regras geradas. Esta ordenação é utilizada pelo algoritmo para a seleção de regras que compõe o classificador.

Definição: Dada duas regras r_i e r_j. Para $r_i \gg r_j$, diz-se que r_i precede r_j ou que r_i tem maior precedência que r_j se:

1. a confiança de r_i for maior que a de r_j; ou
2. as confianças são iguais, mas o suporte de r_i é maior que o de r_j; ou ainda
3. ambos, suportes e confiança, são iguais, mas r_i foi gerada primeiro que r_j.

Seja *R* o conjunto de regras geradas e *D* o conjunto de dados de treinamento. A idéia básica do algoritmo CBA-CB é escolher um conjunto de regras com as mais altas precedências em *R* para cobrir *D*. Assim, o classificador C tem o seguinte formato: $C = < r_1, r_2, r_3, ..., r_n$, classe-padrão>, onde $r_i \in R$ e $r_a \gg r_b$ se $b > a$. Ainda, nas classificações de casos desconhecidos, a primeira regra de C que cobrir o caso o classificará. E na hipótese de nenhuma regra conseguir cobrir o caso, este será classificado como a classe padrão. A figura 3.5 ilustra o algoritmo CBA-CB, que possui 3 etapas distintas:

A primeira etapa trata da ordenação do conjunto de regras geradas, de acordo com a relação de precedência "»". Isto garante a seleção das regras de mais alta precedência para o classificador (linha1).

A segunda etapa do algoritmo, (linhas 2-14), seleciona regras no conjunto R_{sort} para o classificador, seguindo a seqüência de ordenação. Para cada regra "*r*" $\in R_{sort}$, busca-se em *D* encontrar todos os casos cobertos por "*r*" (linhas 5). Marca-se "*r*" se

esta regra classificar corretamente, pelo menos, um caso $d \in D$. Se "*r*" for marcada, então significa que é uma regra potencial a fazer parte do classificador (linha 5-7). Os casos cobertos por "*r*" são removidos de *D* (linha 10). Uma classe padrão também é selecionada, ou seja, é escolhida como classe padrão a classe majoritária nos dados de teste restante. Isso significa que se o algoritmo parar de selecionar mais regras, para o classificador, a última classe padrão escolhida é a que integrará o classificador (linha 11). É, então, computado e gravado o número total de erros cometido pelo classificador corrente, compostos pelas regras selecionadas até o momento, além da classe padrão escolhida. Esse número total de erro é a soma dos erros cometidos, no conjunto de dados de treinamento, por todas as regras que compõe o classificador corrente, inclusive os erros cometidos pela classe padrão. Quando não houver mais regras a serem testadas ou nenhum caso de treinamento sobrando, a etapa de seleção de regras é encerrada.

Algoritmo CBA-CB

entrada: conjunto de regras: **R**

```
1     R_sort= ordena(R)
2     para cada regra r ∈ R_sort em sequência faça
3         temp = ∅
4         para cada caso d ∈ D faça
5             se d satisfaz a condição de r então
6             armazene d.id em temp e marque r se classificou d
7         fim para
8         se r é marcado então
9             insere r no fim do classificador C
10            remova todos os casos de D que possuam id em temp
11            selecione uma classe padrão para C corrente
12            calcule o número total de erros de C
13        fim se
14    fim para
15    ache a primeira regra p em C com o menor número de erro
      e descarte todas as regras em C após p
16    insere a classe padrão associada a p no final de C
17    fim
```

Figura 3.5 : Algoritmo CBA-CB

Finalmente, a terceira etapa do algoritmo, (linhas 15-17), é a responsável em descartar as regras selecionadas para o classificador que não melhoram a eficiência do mesmo, ou seja, a primeira regra que possuir o menor número de erros gravado é a regra de corte. E todas as demais regras após a regra de corte podem ser descartadas, pois apenas produzem mais erros. Assim, as regras não descartadas e a classe padrão da última regra formam o classificador.

3.5.2.2. Outros algoritmos de MRC

Os bons resultados obtidos pela idéia de usar a mineração de regras de classificação baseadas em associações, apresentada pelo CBA motivou o incremento das pesquisas nesta área. Surgiram, então, diversos outros algoritmos de mineração de regras de classificação baseados em associações.

No ano seguinte a apresentação do CBA, em 1999, foi proposto o **LB** (*Large Bayes Classifier*) em [77], um classificador associativo que trata o problema de combinar regras de associação para construção de classificadores mais sobre a perspectiva estatística. Na fase de treinamento do LB, os itens freqüentes com seus "*suporte-classse"* são minerados. O "*suporte-classe"* é uma variação da noção usual de suporte e estima a probabilidade de que um padrão ocorra em uma determinada classe. Para classificar um padrão, o LB constrói, em tempo de execução, um modelo de classificação local, focando um contexto particular. Este modelo de classificação consiste de uma fórmula que computa uma probabilidade condicional. Para a determinação desta fórmula, o LB utiliza *itemsets* freqüentes.

Em 2000, foi proposto o **ADT** (*Association-based Decision Trees*) em [78]. Este algoritmo tem o propósito de, primeiramente, minerar todas as RCA's sem estabelecer limites como suporte mínimo ou confiança mínima. Em seguida, utiliza esse conjunto de regras para construir uma árvore de decisão, daí o nome do algoritmo. Então, realiza a poda da árvore utilizando técnicas de poda para árvores de decisão. Os mais altos níveis da árvore guardam as regras mais gerais, enquanto as regras mais específicas são colocadas nos níveis mais baixos. A poda é realizada de baixo para cima (*bottom-up*), para evitar que boas regras gerais sejam cortadas. Quando uma regra-filho é cortada, seus pais tornam-se folhas. Para decidir se deve ou não podar uma regra, compara-se a

estimativa de erro após o corte com a estimativa antes do corte. Então, se o corte da regra melhorar a estimativa de erro, está é descartada.

O algoritmo **CMAR** (*Classification based on Multiple Association Rules*) foi proposto em 2001 [79]. É, na verdade, uma variação do CBA. Também apresenta duas etapas, a de geração de regras e a de classificação, utiliza uma variação do método *FP-Growth* para geração das RCA's, ordena e poda as RCA's de maneira similar ao CBA. Entretanto, a forma pela qual é realizada a classificação difere do CBA, pois, o CMAR não faz a classificação baseado em apenas uma regra. O algoritmo defende que a classificação feita por apenas uma regra não é um modo robusto de previsão. Assim, dado um caso de teste, as regras escolhidas são divididas em grupos de acordo com suas classes. Isto é, todas as regras de um grupo compartilham da mesma classe e cada grupo de regras apresenta uma classe distinta.. Para cada grupo de regras, o efeito combinado de todas as regras deste grupo é calculado por um procedimento de pesos das regras, de modo que a previsão da classe baseia-se em múltiplas regras.

O **CPAR** (*Classification based on Predictive Association Rules*) apresentado em 2003 por [80], tenta combinar as vantagens da classificação associativa e as da classificação tradicional. A geração de regras neste algoritmo é similar a feita pelo algoritmo FOIL (*First-Order Inductive Leaner*) [24], um algoritmo guloso que ensina regras a distinguir os exemplos positivos dos exemplos negativos. E, repetidamente, busca pela melhor regra corrente, e então, remove todos os exemplos positivos cobertos pela regra, até que toda a base de dados de treinamento seja coberta. Na mineração de regras do CPAR, ao invés de remover um exemplo, após sua cobertura por uma regra, estabelece-se um peso para cada exemplo, e a cada cobertura esse peso é decrementado. Tal processo permite que exemplos positivos sejam cobertos várias vezes. Desta forma, é gerado um grande conjunto de RCA's.

Após a mineração de todas as RCA's, cada regra é avaliada quanto ao seu poder de predição. Isto é feito através da estimativa de erro esperado de *Laplace*. Para classificar um padrão, o CPAR escolhe a melhor regra de cada classe, usando o seguinte procedimento: seleciona todas as regras cujos antecedentes satisfazem o padrão; desse conjunto de regras selecionadas, acha a melhor regra de cada classe segundo o método de *Laplace*; e, então, compara a média de precisão estimada para a melhor regra de cada

classe, escolhendo a classe com a mais alta precisão estimada, como a classe do padrão de teste.

Outro algoritmo para classificação associativa proposto em 2004, foi o **CorClass** (Correlated Association Rule Mining for Classificatino) [82]. Este algoritmo descobre todas as RCA's adaptando a técnica apresentada em [81], e então, aplica esse conjunto de regras para a predição de casos desconhecidos. A principal vantagem do algoritmo é que se comparado a outros algoritmos para classificação associativa, o CorClass encontra diretamente o conjunto das melhores RCA's, através do emprego de um algoritmo de *branch-and-bound*. Enquanto que os demais algoritmos para esta tarefa primeiramente descobrem todas as regras de associação que satisfaçam aos parâmetros suporte mínimo e confiança mínima, só então pós-processam essas regras, de modo a selecionar o conjunto de RCA's que compõe o classificador.

Verifica-se, através dos algoritmos apresentados, que são muitos os esforços no sentido de contribuir de alguma forma para a metodologia de classificação por regras de associação. Isto leva a crer que existe realmente um grande potencial nesta área de pesquisa. E que ainda surgirão muitos outros estudos nessa área.

3.6 ALGORITMOS PARALELOS

Nessa seção é apresentada uma discussão das principais características dos algoritmos paralelos, suas vantagens e desvantagens tanto em relação a plataforma, quanto ao tipo de paralelismo. Em seguida, são apresentados alguns dos principais algoritmos paralelos desenvolvidos para mineração de regras de associação.

3.6.1. Principais Características

A utilização da computação paralela nos métodos de mineração de regras de associação visa aliviar estes métodos do gargalo imposto pela solução seqüencial, fornecendo escalabilidade para o trato com grandes conjuntos de dados, além de melhorar o tempo de resposta desses algoritmos. Entretanto, alcançar bons desempenhos em sistemas multiprocessados não é tarefa trivial. O desafio principal inclui minimização da

sincronização e da comunicação, balanceamento de carga de trabalho, descoberta de bons arranjos de dados, além da minimização dos acessos a disco, que é de especial importância para mineração de regras de associação.

Um projeto paralelo apresenta três componentes principais: plataforma de *hardware*, tipo de paralelismo e estratégia de balanceamento de carga.

Quanto à plataforma de *hardware*, duas abordagens dominantes para o uso de multiprocessamento têm emergido: a arquitetura de memória distribuída, onde cada processador tem uma área de memória privada, e a arquitetura de memória compartilhada, onde os processadores acessam uma área de memória comum.

A estratégia de memória compartilhada apresenta muitas propriedades desejáveis. Cada processador tem acesso igual e direto a todo sistema de memória. Portanto, os programas paralelos são mais simples de se implementar nestes sistemas.

Uma outra solução para o multiprocessamento é construir um sistema de muitas unidades, cada uma contendo um processador e uma memória local independente. Nestes sistemas de memória distribuída, para um processador acessar dados na memória de um outro processador, deve haver troca de mensagens entre os processadores.

Embora uma arquitetura de memória compartilhada ofereça mais simplicidade de programação, a largura de banda finita do barramento comum pode limitar a escalabilidade. Um sistema de memória distribuída, com estratégia de troca de mensagens, melhora a solução para o problema da escalabilidade, pela eliminação do barramento comum, entretanto, perde-se na simplicidade dos programas paralelos.

A otimização do desempenho buscada pela aplicação da estratégia paralela depende da arquitetura básica, pois, nos sistemas de memória distribuída, a sincronização está implícita na passagem de mensagens. Assim o objetivo torna-se a otimização da comunicação. Entretanto, para os sistemas de memória compartilhada a sincronização origina-se nos sinalizadores e nas barreiras. Logo, o objetivo é otimizar este ponto.

A decomposição de dados é muito importante para sistemas de memória distribuída, mas não para os sistemas de memória compartilhada. Entretanto, os processos paralelos de entrada e saída (*E/S*) são simples na arquitetura distribuída, mas podem ser problemáticos em sistemas de memória compartilhada, que tipicamente realiza de forma serial processos de entrada e saída (*E/S*). O maior desafio para encontrar um bom desempenho em sistemas distribuídos é achar uma boa decomposição de dados entre os nós e minimizar a comunicação.

Dois paradigmas principais para a exploração dos algoritmos paralelos são o paralelismo dos dados e o paralelismo das tarefas. Nos algoritmos de mineração de regras de associação, o paralelismo dos dados se dá quando a base de dados é fisicamente particionada entre os processadores, nos sistemas distribuídos, ou logicamente particionada, nos sistemas compartilhados. Já a paralelização de tarefas correspondem ao caso onde os processadores, por exemplo, realizam diferentes cálculos, independentemente, tais como contar um conjunto disjunto de candidatos, mas ter ou necessitar de acesso a todo o banco de dados. Os sistemas compartilhados têm acesso a todos os dados, mas nos sistemas distribuídos o processo de acessar a base de dados pode envolver seletivas replicação ou comunicação explícita da porção local. O paralelismo híbrido, que combina ambos, paralelismo de dados e de tarefas, é também possível e talvez desejável para explorar todos os métodos de paralelismo disponíveis em mineração de regras de associação.

Outro componente importante nos algoritmos paralelos é o balanceamento de cargas, que pode ser de dois tipos: estático ou dinâmico. No balanceamento de carga estático, inicialmente, divide-se o trabalho entre os processadores usando uma heurística de custo qualquer. Nenhum dado subseqüente ou movimento de computação está disponível para corrigir o desbalanceamento de carga.

O balanceamento dinâmico busca solucionar este problema através da retirada de tarefas de um processador sobrecarregado e atribuindo estas tarefas a um outro processador, com menos trabalho a realizar. Movimento de computação também acarreta movimento de dados, porque o processador responsável por uma tarefa de computação precisa dos dados associado a esta tarefa. Portanto, balanceamento de carga dinâmico incorre num custo adicional para o movimento do trabalho e dos dados. E

também para um mecanismo que seja capaz de detectar o desbalanceamento. Então, para que o balanceamento dinâmico seja essencial, deve existir um grande desbalanceamento de carga, que justifique seu emprego.

O balanceamento de carga é especialmente importante em ambientes multi-usuários com transientes de cargas e em plataformas heterogêneas, que tem diferentes processadores e diferentes velocidades de redes. Essa espécie de ambiente inclui servidores paralelos e *clusters* (grupos) heterogêneos. A maioria dos algoritmos de mineração de regras de associação existente usa apenas balanceamento estático de carga [59], que é herdado do particionamento inicial da base de dados entre os nós existentes. Isto porque esses algoritmos assumem que terão um ambiente dedicado e homogêneo.

3.6.2. Principais Algoritmos

Um dos primeiros algoritmos paralelos para mineração de regras de associação foi o **PEAR** (*Parallel Efficient Association Rules)* [63]. Este algoritmo é, na verdade, uma versão paralela do SEAR e, portanto, baseado nos algoritmos APRIORI e PARTITION. No PEAR, em cada iteração cada processador gera uma árvore de prefixos de candidatos a partir do conjunto global de itens freqüentes do passo anterior. Assim, cada processador tem uma cópia inteira do mesmo conjunto de candidatos. Cada nó, então, coleta os suportes locais, e por meio de uma primitiva de redução de soma, se obtém o suporte global em cada processador.

Outro algoritmo paralelo de mineração de regras de associação é o **PPAR** (*Partitioned Parallel Association Rules*) [63]. De fato, este algoritmo é uma paralelização sugerida do SPEAR. Ao contrário de sua versão serial, o PPAR usa dados no formato horizontal. Neste algoritmo, cada processador acha seu conjunto local de itens freqüentes, de todos os tamanhos, em apenas uma passagem sobre a base de dados local. Em seguida, os conjuntos freqüentes locais são replicados para todos os demais processadores. Então, cada processador calcula os suportes destes candidatos globais. E, finalmente, após a difusão desses candidatos globais, é, então, obtido o conjunto freqüente global.

Também baseado numa estratégia de mineração de regras de associação serial, o **PDM** (*Parallel Data Mining*) [70] é a versão paralela do DHP (*Dynamic Hashing and Pruning*). Neste algoritmo, cada processador computa o suporte local de conjuntos freqüentes de tamanho 1, e faz uma estimativa para os conjuntos freqüentes de tamanho 2 através de uma tabela *hash*. Então, por meio de uma mensagem do tipo *todos-para-todos,* das contagens de suporte local obtém-se a contagem global para os conjuntos freqüentes de tamanho 1. Já que a tabela *hash* dos conjuntos freqüentes de tamanho 2 pode ser muito grande, trocar diretamente contagem através de mensagens pode ser muito custoso. Assim, é utilizado um método de otimização que troca apenas as células, as quais é garantido serem freqüentes. Entretanto, esse método requer duas rodadas de comunicação. Em seguida, o PDM gera candidatos locais usando a tabela *hash* do conjunto global de itens freqüentes de tamanho 2. Apenas neste passo é utilizada a tabela *hash* para a geração de candidatos, pois nos passos subseqüentes os candidatos são gerados diretamente do conjunto freqüente da iteração anterior, como no APRIORI. Os candidatos são gerados em paralelo, cada processador gera seu próprio conjunto local, que são trocados através de mensagens de *todos-para-todos*, de modo a se construir o conjunto de candidatos global. Então, o PDM obtém a contagem local para todos os candidatos e as troca entre todos os processadores para determinar o conjunto de itens freqüentes global. Este algoritmo paraleliza a geração de candidatos através de uma mensagem de *todos-para-todo,* para construir o conjunto de candidatos global. Logo, o custo da comunicação no PDM, pode tornar esta paralelização ineficiente.

Muitos algoritmos paralelos também se baseiam no APRIORI, devido a seu sucesso nas aplicações seqüenciais. Em [83] foram propostos três algoritmos paralelos baseado no APRIORI. Estes são denominados por: Distribuição de Contagem, Distribuição de Dados e Distribuição de Candidatos.

O algoritmo de Distribuição de Contagem, **DC**, é uma simples paralelização do APRIORI, onde a base de dados é particionada entre os processadores. Neste algoritmo, todos os processadores geram uma árvore *hash* completa de candidatos, a partir do conjunto de itens freqüentes da iteração anterior. Portanto, cada processador, independentemente, pode computar os suportes parciais dos candidatos de suas partições locais da base de dados. Então, através de uma primitiva de comunicação do tipo redução de soma e da troca do suporte local com os outros processadores, é obtida a

contagem ou suporte global. Ao invés de combinar diferentes árvores *hash*, é preciso comunicar apenas a contagem parcial, já que cada processador tem uma cópia da árvore inteira. Tão logo o conjunto freqüente global tenha sido determinado numa iteração, cada processador constrói uma árvore completa de candidatos, em paralelo. Este processo se repete até que todos os conjuntos freqüentes sejam encontrados. Neste algoritmo a comunicação é minimizada, já que apenas suportes são trocados entre os processadores. É importante lembrar que neste algoritmo os processadores podem varrer assincronamente e em paralelo suas partições de dados locais. Entretanto, devem ser sincronizados a fim de se desenvolver a contagem global dos candidatos.

O algoritmo de Distribuição de Contagem apresenta como característica mais atraente não haver troca de dados entre os processadores, apenas troca de suportes. Portanto, os processadores podem operar independentemente e assincronamente enquanto lêem os dados. Todavia, a desvantagem desse algoritmo é que ele não explora a real capacidade de memória do sistema. Suponha que cada processador tenha uma memória de tamanho | M |. O número de candidatos que pode ser computado em uma iteração do algoritmo é determinado por | M |. Se o número de processadores aumenta de 1 para N, o sistema passa a ter uma quantidade total de memória igual a N x | M |, mas a quantidade de candidato computada em uma iteração do algoritmo permanece a mesma, já que cada processador computa uma quantidade idêntica de candidatos.

O algoritmo de Distribuição de Dados, **DD,** é projetado para explorar melhor a quantidade total de memória do sistema, à medida que o número de processadores aumenta. Neste algoritmo, cada processador conta de maneira mutuamente exclusiva os candidatos. Assim, se o número de processadores aumenta, um número maior de candidatos pode ser computado em uma iteração do algoritmo. Porém, a desvantagem deste algoritmo é que cada processador deve divulgar para os demais sua partição de dados em cada iteração do algoritmo. Portanto, este algoritmo só é viável em sistemas que apresentam comunicação muito rápida. Logo, é fácil verificar que esse algoritmo sofre com o excesso de comunicação e, portanto, seu desempenho é baixo comparado com o algoritmo de Distribuição de Contagem.

Ambos os algoritmos de Distribuição de Contagem e de Distribuição de Dados requerem sincronização dos processadores no final de uma iteração, seja para troca de

suportes ou de itens freqüentes, respectivamente. Se a carga de trabalho não estiver perfeitamente balanceada, pode fazer com que todos os processadores esperem por aquele processador que for o último a terminar sua tarefa, em cada iteração do algoritmo. Já o algoritmo de Distribuição de Candidatos procura livrar-se das dependências entre os processadores, de modo que estes possam proceder independentemente sem necessidade de sincronização no fim de cada iteração.

O algoritmo de Distribuição de Candidatos, **DCdt**, divide os candidatos durante as iterações, de modo que cada processador pode gerar conjuntos disjuntos de candidatos independentemente de outros processadores. A divisão dos candidatos utiliza uma heurística baseada no suporte, de modo que cada processador recebe uma quantidade igual de trabalho. Ao mesmo tempo, a base de dados é seletivamente replicada, de modo que um processador pode gerar contagem global independentemente. A escolha da iteração de redistribuição envolve uma negociação entre desacoplamento das dependências do processador tão logo seja possível e aguarda até que um balanceamento de carga suficiente seja alcançado. Nos experimentos de [83], o reparticionamento é feito na 4ª iteração. Depois que a única dependência que um processador tinha em outro processador era a poda de candidatos, cada processador assincronamente difunde o conjunto freqüente local para os outros processadores durante cada iteração. Se essa informação de poda chega a tempo, ela é usada; do contrário, esta é armazenada para a próxima iteração. Cada processador deve, ainda, examinar seus dados locais uma vez por iteração. Porém, esse algoritmo se mostrou pior em desempenho que o algoritmo de Contagem Distribuída, já que ele paga o alto custo da redistribuição da base de dados enquanto examina sua partição local da base de dados repetidamente.

Em [61] foram propostos quatro algoritmos paralelos baseados no APRIORI, que são muito similares aos três algoritmos citados anteriormente. O **NPA** (*Non Partition Apriori),* que é essencialmente o mesmo algoritmo que o de Distribuição de Contagem, com a diferença que a redução de soma ocorre num processador mestre. O **SPA** (*Simply Partitioned Apriori*) que é bem similar ao algoritmo de Distribuição de Dados e o **HPA** (*Hash Partitioned Apriori),* similar ao de Distribuição de Candidatos. Neste último, cada processador gera candidatos a partir de conjuntos freqüentes de uma iteração prévia e aplica uma função *hash* para determinar um processador residente para

os conjuntos de candidatos gerados. Se um processador é a residência de um conjunto candidato, este insere o candidato na árvore *hash* local; do contrário, ele descarta o candidato. Também, foi proposto uma variante do HPA, chamada de **HPA-ELD** (*Extreme Long Duplicate – HPA*). A motivação desta proposição é que, muito embora seja possível dividir os candidatos igualmente entre os processadores, alguns candidatos são mais freqüentes que outros. Portanto, seus processadores locais são, consequentemente, sobrecarregados, enquanto os outros processadores têm pouca carga de trabalho. O HPA-ELD resolve isto replicando os *itemsets* extremamente freqüentes por todos os processadores e os processa utilizando uma estratégia de balanceamento dinâmico de carga muito limitada. Em [61] os experimentos confirmam que o HPA-ELD tem desempenho superior aos outros algoritmos propostos. Todavia, o SPA, HPA e o HPA-ELD foram utilizados apenas para a segunda iteração. Nas demais iterações restantes utilizaram o NPA. Isto sugere que a melhor abordagem é um algoritmo híbrido, ou seja, um algoritmo que use HPA-ELD nas iterações onde o número de candidatos é elevado e caiba na memória, e, quando o número de candidatos diminuir a valores viáveis para o tamanho da memória disponível, usa-se o NPA.

Outros dois métodos de mineração de regras de associação baseados no algoritmo de Distribuição de Dados foram propostos em [84], pois foi observado que o algoritmo de Distribuição de Dados tem uma operação muito custosa para enviar porções da base de dados local para todos os outros processadores. Além disso, embora este algoritmo divida os candidatos igualmente entre os processadores, ele falha em dividir o trabalho realizado em cada transação. Assim, no **IDD** *(Intelligent Data Distribution),* é usada uma estratégia baseada em anel do tipo de todos-para-todos, que é menos custosa para enviar aos demais processadores a porção local da base de dados e ainda passa a executar a abordagem de Distribuição de Contagem tão logo os dados caibam na memória. O segundo algoritmo proposto é o **HD** (*Hybrid Distribuition*) que é uma combinação do algoritmo DC e do IDD. Neste algoritmo, os processadores são divididos em G grupos de iguais tamanhos, onde cada grupo é considerado um superprocessador. O algoritmo de Distribuição de Contagem (DC) é usado entre os G superprocessadores, enquanto que cada processador em um grupo usa IDD. A base de dados é horizontalmente particionada ente os superprocessadores e os candidatos são particionados entre os processadores em um grupo. O algoritmo ainda ajusta o número de grupos dinamicamente para cada iteração. As vantagens do HD são que ele reduz o

custo de comunicação da base de dados para 1/G e tenta manter os processadores ocupados, especialmente durante as últimas iterações.

Em [85] foi proposto uma versão paralela do FDM (*Fast Data Mining*), chamada de **FPM** (*Fast Parallel Mining*). O problema do FPM é que este algoritmo requer duas rodadas de troca de mensagens em cada iteração. Uma para a determinação dos suportes globais e outra para a difusão dos *itemsets* freqüentes. Este esquema de comunicação pode degradar a performance do algoritmo. Entretanto, este algoritmo apresenta uma métrica de distribuição de itens freqüentes entre as partições muito interessante e eficiente para certos conjuntos de dados. Os experimentos mostram que em determinados casos foi obtido uma eficiência bem superior ao algoritmo de Distribuição de Contagem.

Também é importante lembrar que em [60] foram apresentadas os algoritmos **PARECLAT**, **PARMAX**, **PARCLIQUE** e **PARMAXCLIQUE**, que são na verdade paralelização de suas versões seriais. Entretanto, diferentemente dos algoritmos paralelos mencionados anteriormente nesta seção, estes quatro algoritmos foram desenvolvidos e experimentados em sistemas híbridos, isto é, sistemas que apresentam tanto componentes de memória compartilhada, quanto componentes de memória distribuída. Porém, muito embora existam alguns algoritmos de mineração de regras de associação para sistemas de memória compartilhada, nesta tese a ênfase foi dada aos algoritmos para sistemas com memória distribuída. E, neste caso, é possível notar que estes algoritmos, na sua maioria, são na verdade baseados em poucas estratégias diferentes. Por exemplo, O PEAR, o PDM, o NPA e o FPM são similares ao DC, do mesmo modo que o SPA e o IDD são similares ao DD, enquanto que o HPA e o HPA-ELD são similares ao DCdt.

Capítulo IV

A Proposta

Neste capítulo são descritas as principais fases do processo de desenvolvimento da implementação apresentada nesta tese. São apresentadas, ainda, as estratégias e estruturas de dados adotadas, as dificuldades e limitações encontradas, os detalhes da paralelização dos algoritmos e as ferramentas de apoio que foram desenvolvidas para os testes de corretude e de eficiência das implementações.

4.1 PRINCIPAIS REQUISITOS

Nesta seção são apresentados os principais requisitos da implementação, bem como os algoritmos escolhidos para a implementação das tarefas de mineração de regras de associação e de classificação.

Um dos objetivos desta tese, é a aplicação de tarefas de mineração de dados para o aprimoramento de um sistema de contramedidas antimíssil, que será melhor detalhado no capítulo 5. A meta inicial deste desenvolvimento é a implementação dos algoritmos de mineração de regras de associação e de classificação para aplicação nos sistemas militares supracitados.

O algoritmo APRIORI foi o escolhido como o algoritmo base de nossa implementação, pois como discutido no capítulo 3, este algoritmo norteia a maioria das implementações bem sucedidas para a realização da tarefa de mineração de regras de associação, embora existam outras abordagens interessantes para o problema. Além do mais, estudos mostram que a classificação associativa produz resultados bastante interessantes. Portanto, a adoção do APRIORI facilita a implementação do algoritmo CBA, responsável pela tarefa de classificação.

Outro requisito a ser considerado é o tamanho das bases de dados onde serão realizadas as tarefas de mineração de dados. Como as bases de dado da aplicação vêm crescendo na medida em que novos dados são coletados, a tendência é o processamento de grandes massas de dados. Assim, na tentativa de garantir a escalabilidade, essa implementação é desenvolvida para execução em ambiente de computação paralela. E como plataforma de *hardware* é utilizado um *cluster* de PC's.

Diversas estratégias paralelas para o algoritmo APRIORI e suas variações são apresentadas no capítulo 3. E de acordo com [83], a paralelização dos dados, embora mais simples, apresenta melhores resultados que a paralelização de tarefas nas implementações desse algoritmo. Por isso, foi usado como estratégia de paralelização o particionamento da base de dados, tanto para a mineração de itens freqüentes, quanto para a construção do classificador.

As tarefas de mineração de dados, sobretudo a de geração de regras de associação, podem ser melhor desenvolvidas quando existe conhecimento prévio sobre os dados a serem minerados, possibilitando, por exemplo, a hierarquização destes dados em grupos ou a introdução de restrições que influenciem nos resultados das associações encontradas. Logo, é de grande interesse o total domínio das implementações desses algoritmos. A fim de possibilitar futuras alterações de código que permitam ingerências sobre os dados, foram implementadas todas as rotinas, funções e estruturas de dados utilizadas na codificação dos algoritmos. A linguagem utilizada para implementação foi o C padrão ANSI, devido a sua flexibilidade e eficiência.

A implementação integral dos algoritmos traz facilidades para futuras alterações ou acréscimo de funcionalidades. Porém, muitos autores proclamam que a

implementação tem grande influência na eficiência dos algoritmos de mineração de regras, e que as habilidades de programação são tão ou mais importantes que os conhecimentos na área de mineração de dados. Entretanto, nesta fase do trabalho, o compromisso é com uma implementação que apresente uma boa eficiência, não havendo a preocupação de se obter um desempenho ótimo.

4.2 INICIANDO A IMPLEMENTAÇÃO

Esta seção descreve detalhes do processo de desenvolvimento da implementação, a experiência com os modelos das estruturas de dados utilizadas, as principais dificuldades e as soluções adotadas.

4.2.1. As Diretivas

A mineração de regras de associação, como já mencionado anteriormente, pode ser dividida em dois subproblemas principais: a busca dos itens freqüentes e a geração das regras de associação. O subproblema de busca de itens freqüentes é, do ponto de vista computacional, muito mais complexo do que a geração de regras. Por essa razão, a maioria das pesquisas desta área é direcionada à busca de itens freqüentes. Logo, concentrou-se na implementação de um algoritmo baseado no APRIORI que executasse a tarefa de mineração de itens freqüentes de forma eficiente, objetivando sua aplicação em bases de dados militar.

Um dos requisitos da implementação é a sua execução em ambiente paralelo. Porém, como a estratégia escolhida foi o paralelização dos dados, decidiu-se iniciar o desenvolvimento com uma implementação serial, o que facilita a depuração da codificação dos algoritmos.

A mineração de itens freqüentes pode ser descrita de forma simplista, como um problema de enumeração e contagem de dados. Todavia, existem detalhes importantes nas implementações de algoritmos envolvendo grandes quantidades de dados, que não podem ser relegados, pois inviabilizam a sua execução. No caso do algoritmo

APRIORI, alguns cuidados devem ser tomados, principalmente na escolha das estruturas de dados utilizadas.

4.2.2. Representando os Dados

Inicialmente a implementação do algoritmo foi realizada fielmente como descrito em [3], inclusive com a utilização de uma árvore *hash* como estrutura de armazenamento dos *itemsets* candidatos. Nesta implementação utilizou-se um vetor de *bits* como estrutura básica de representação para os elementos dos *itemsets*. Como estes são codificados por inteiros positivos, a utilização de um vetor de *bits* é uma boa alternativa para economia de memória.

Na fase de depuração, foram utilizadas pequenas base de dados de testes para facilitar a correção de erros e, por isso, não havia comprometimento com a eficiência da implementação. Entretanto, após esta fase, quando a implementação foi submetida a maiores bases de dados, percebeu-se que essa representação de dados exigia muitas comparações tanto na fase de geração de candidatos quanto na fase de contagem dos suportes. Então, embora o consumo de memória fosse baixo, a eficiência do algoritmo era muito comprometida.

De modo a buscar uma representação mais econômica em termos de memória, para os elementos dos *itemsets*, experimentou-se algumas formas de codificação para estes conjuntos de dados. Dentre estas, a que mostrou resultados mais interessantes foi a representação destes conjuntos através do produto de números primos.

Nesta representação, cada item da base de dados é codificado como um número primo único, e todos os conjuntos de itens, incluindo as transações da base de dados, são representados pelo produto dos números primos de seus componentes. Com isto houve economia de memória, já que cada conjunto de candidatos gerado é representado apenas pelo valor do produto de seus componentes. Além do mais, a fase de contagem dos suportes dos candidatos é beneficiada, pois uma simples operação de divisão entre o produto de primos representante de uma transação e o produto de primos representante de um conjunto candidato é suficiente para verificar se o candidato pertence ou não a transação.

Como tudo tem dois lados, as desvantagens dessa representação de conjuntos de itens são as seguintes: (1) Necessidade de armazenamento de uma tabela de primos para codificação dos itens, pois não há nenhum procedimento automático de geração de números primos; e (2) limitação do valor do produto em relação ao valor máximo suportado pelo tipo de dado da linguagem. Diante dessas restrições, resolveu-se abandonar este modo de representação de conjuntos de itens, muito embora esse modelo possa vir a ser aproveitado em alterações futuras desta implementação, quando a quantidade de itens distintos das bases de dados da aplicação estiver mais estabelecida[2].

De modo a livra-se das restrições da representação de conjuntos por produtos de primos e da baixa eficiência obtida com a representação destes conjuntos por vetores de *bits*, os elementos dos *itemsets* foram, então, representados por um vetor de inteiros. Embora a utilização desse modelo de dados não represente uma economia de memória tão boa quanto à obtida pelo modelo de produtos de números primos, a eficiência dessa implementação do algoritmo APRIORI melhorou muito se comparada à implementação que usa vetor de *bits*.

As três principais funções do algoritmo APRIORI são: (1) a geração de *itemsets*, candidatos; (2) a contagens dos suportes dos candidatos e (3) a formação dos *itemsets* freqüentes. Estas funções são compostas de algoritmos básicos, como por exemplo, a busca de elementos em um conjunto de dados ou, ainda, a ordenação de um conjunto de dados. Devido à quantidade de dados do problema e ao caráter iterativo do algoritmo, mesmo a implementação desses algoritmos básicos deve obedecer a critérios de complexidade computacional mínima. Do contrário, a eficiência da implementação é comprometida.

Embora as implementações do algoritmo APRIORI desta fase do desenvolvimento contassem com algoritmos básicos em suas formas otimizadas, a representação dos dados influenciou de maneira definitiva na eficiência das implementações. Isto demonstra que o desenvolvimento de implementações para grandes massas de dados não é um problema tão trivial e merece atenção aos detalhes.

[2] O conhecimento sobre os dados pode ser usado no aprimoramento das implementações de tarefas de mineração de dados, sobretudo, em casos de desenvolvimento para aplicações específicas.

4.2.3. Ferramentas de Apoio

As bases de dados que servem de entrada para esta implementação do algoritmo APRIORI são arquivos planos contendo uma tabela de dados. Cada linha da tabela representa uma transação, que contém uma lista de números inteiros positivos separados por um espaço em branco ou algum outro caracter, como ilustra a figura 4.1.

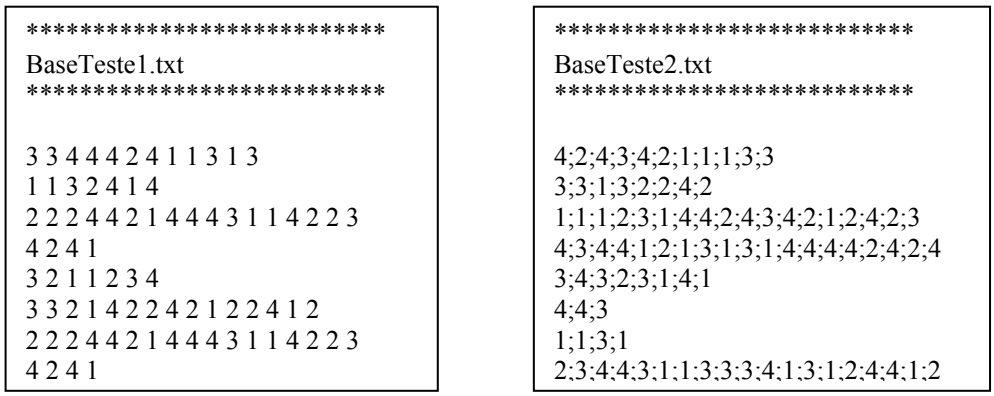


Figura 4.1 : Exemplos de Arquivos de Base de Dados

Como uma transação é apenas uma lista de eventos, esses podem aparecer em qualquer ordem e em qualquer quantidade. Mas esse tipo de informação não interessa para a mineração de itens freqüentes. É necessário então, um trabalho de pré-processamento dos dados, de modo que estes possam ser submetidos à implementação do algoritmo APRIORI. Cada transação deve ser armazenada num conjunto, onde os dados são organizados em ordem crescente e os elementos repetidos descartados. Só após essa fase é que estes dados são submetidos ao algoritmo para a realização da tarefa de mineração de itens freqüentes.

Para a depuração das rotinas de pré-processamento dos dados e das demais rotinas que compõem essa implementação do algoritmo APRIORI foi necessária a utilização de bases de teste com pequenas quantidades de dados, onde fosse possível verificar a corretude das rotinas implementadas. Entretanto, vislumbrou-se que seria interessante ter algumas bases de testes com maiores quantidades de dados para a fase das primeiras avaliações de eficiência da implementação. Além disso, estas bases deveriam estar em formatos que pudessem ser submetidas a outras implementações já

consagradas do APRIORI, de modo a possibilitar a comparação dos resultados e comprovar a corretude da implementação em desenvolvimento.

Assim, iniciou-se o desenvolvimento de uma ferramenta que gerasse bases de teste, de qualquer tamanho e em vários formatos diferentes, para utilização destas por outras implementações do algoritmo. Porém, para a execução do algoritmo era necessário estabelecer previamente diversos parâmetros, tais como valor de suporte mínimo, número de máximo de conjuntos freqüentes, nome do arquivo da base de dados e etc. Então, foi desenvolvida uma interface para facilitar a introdução destes parâmetros e esta foi integrada á implementação do algoritmo, como ilustra a figura 4.2. Promoveu-se, ainda, a integração desta interface com uma outra desenvolvida para a ferramenta de geração de bases de teste, de modo a facilitar todo o processo de depuração da implementação.

Figura 4.2 : Interface para geração de itens freqüentes

A partir da interface mostrada na figura 4.2, é possível, através do acionamento do botão "*gerar base de dados,*" abrir a tela com a interface mostrada na figura 4.3, para geração de um arquivo de base de teste. Nesta interface pode-se definir o número total de transações da base de teste, a quantidade máxima de itens numa transação, o número

máximo de itens a ser considerado, o caracter de separação dos itens e o nome do arquivo a ser gerado. É importante ressaltar que a missão desta ferramenta é de gerar dados apenas para a fase de depuração, pois, embora os dados gerados sejam aleatórios, não existe nenhum cuidado que justificasse a utilização desses dados para a avaliação efetiva da eficiência da implementação.

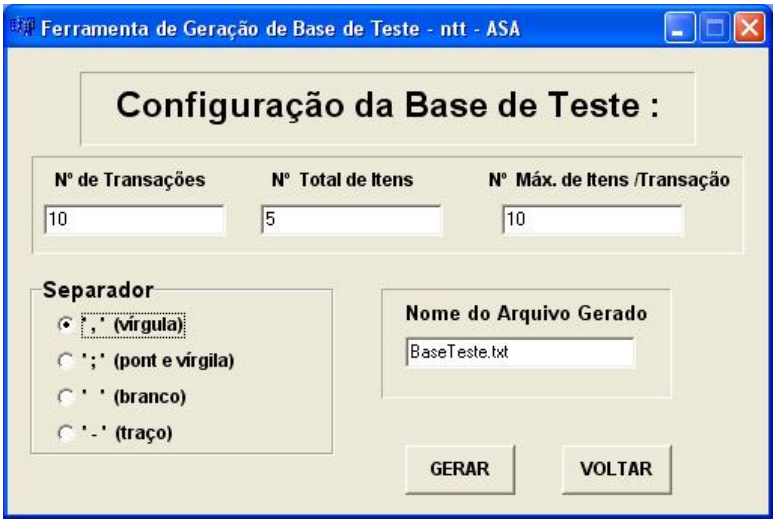


Figura 4.3 : Interface para geração de arquivos de bases de testes

4.3 MELHORANDO O DESEMPENHO

Esta seção apresenta as primeiras avaliações de eficiência, discute alguns aspectos a serem otimizados na implementação, mostra como o armazenamento e a manipulação dos dados afeta a eficiência do algoritmo e apresenta, ainda, a idéia de uma estrutura de armazenamento de dados mais simples, que foi adotada nesta implementação do algoritmo.

4.3.1. As Primeiras Avaliações

As caracteríscas de uma base de dados, ou seja, seu tamanho, quantidade de itens e, sobretudo, a natureza dos dados influenciam o desempenho do algoritmo, pois a quantidade de dados gerados pode ser muito diferente. Por exemplo, as bases de dados

transacionais costumam ser esparsas[3], enquanto que as bases relacionais, utilizadas na tarefa de classificação são muito densas, pois nestas bases os dados estão mais correlacionados.

Assim, como a implementação desenvolvida para a mineração dos itens freqüentes é utilizada também na implementação do classificador proposto, achou-se que seria prudente a realização de testes em bases genéricas, muito embora essas implementações objetivem uma aplicação específica. Além do mais, espera-se que as bases de dados da aplicação sofram um crescimento gradual, à medida que novos dados estão sendo coletados.

Nos primeiros testes de eficiência foram usadas algumas bases[4] de dados conhecidas para avaliação desta classe de algoritmo, disponíveis num repositório de dados para mineração de itens freqüentes [86]. Percebeu-se que quando os valores de suporte mínimo eram muito baixos ou a quantidade de itens da base de dados fosse elevada a eficiência da implementação caía bastante. Isto se deve, principalmente, ao aumento da quantidade de *itemsets* candidatos gerados.

4.3.2. Gargalo do APRIORI

Diante do baixo desempenho nos testes de algumas instâncias do problema, verificou-se que a causa principal era a grande quantidade de candidatos gerados, não só pelo consumo de memória, mas, sobretudo, pelo tempo gasto em duas fases do algoritmo: a fase de geração de novos candidatos e a fase de contagem dos suportes destes candidatos.

4.3.2.1. Melhorando a geração de candidatos

Na geração de *itemsets* candidatos, a cada novo conjunto de itens gerados, antes de ser considerado um *itemset* candidato, ele é submetido a um processo de poda baseado na propriedade 2.2, apresentada no capítulo 2, isto é, quando um novo *itemset* de tamanho

[3] Neste caso considera-se esparsas as bases onde há pouca ocorrência de eventos simultâneos em relação ao tamanho das bases, isto é, pouca correlação entre os dados de uma mesma base.
[4] Alguns exemplos de bases de dados utilizadas: T10I4D100K, T40I10D100K , CHESS, CONNECT, MUSHROOM, e etc.

K é gerado, realiza-se uma verificação para avaliar se algum dos seus subconjuntos de tamanho (K-1) é infreqüente. Então, ele só é considerado um *itemset* candidato se todos os seus subconjuntos são freqüentes, caso contrário o conjunto de itens gerado é descartado.

Conseguiu-se melhorar o tempo gasto pelo processo de poda com a criação do algoritmo de poda mostrado na figura 4.4. Neste algoritmo, ao invés de se gerar todos os subconjuntos de um conjunto de itens e submeter esses subconjuntos ao processo de verificação, contam-se apenas as ocorrências de subconjuntos freqüentes no novo conjunto de itens gerado.

Algoritmo de Poda

entrada: ConjItem
saída: 1 candidato
-1 descarta

```
1    Lê entrada X_K = ConjItem
2    zera contador: ContFreq =0
3    enquanto ContFreq < K faça
4        para cada elemento Y_K-1 de F_K-1
5            se Y_K-1 ∈ X_K então ++ ContFreq
6    se ContFreq == K então retorna 1
7    retorna -1
```

Figura 4.4 : Algoritmo de Poda

Cada novo conjunto de itens gerado é carregado em X_K e submetido ao algoritmo de poda da figura 4.4 (linha 1). O contador de freqüência *ContFreq* é zerado (linha 2). Então, é realizada uma varredura dos *itemsets* freqüentes de F_{K-1}. A cada ocorrência de um *itemset* freqüente em X_K o contador *ContFreq* é incrementado. Este procedimento é repetido até que *ContFreq* atinja uma contagem igual a K, ou terminem os *itemsets* freqüentes em F_{K-1} (linhas 3-5). O algoritmo termina com a avaliação do valor de *ContFreq*: se este for inferior a K significa que pelo menos um dos subconjuntos de X_K é infreqüente, ou seja, X_K deve ser descartado; caso contrário, X_K é considerado como um *itemset* candidato.

Em geral, o procedimento de poda é benéfico ao desempenho do algoritmo, pois reduz o número de *itemset* candidatos gerados. Entretanto, se a base de dados é muito densa, o procedimento de poda torna-se um desperdício de tempo, pois neste caso, a maioria dos conjuntos verificados são realmente *itemsets* candidatos. Assim, o número de reduções não compensa o tempo gasto para a verificação dos conjuntos de itens. Portanto, resolveu-se deixar o procedimento de poda como uma opção do usuário, pois, dependendo da base de dados minerada, pode ou não ser vantajoso o uso da poda.

4.3.2.2. Melhorando a contagem de suportes

A fase de contagem de suportes dos *itemsets* candidatos apresenta um tempo de execução muito superior que as demais fases do algoritmo. Por isso, esta fase é considerada o grande gargalo do APRIORI. Portanto, de modo a se obter melhorias no desempenho da implementação, passou-se a buscar formas de reduzir o tempo gasto na contagem dos suportes.

Na fase de contagem dos suportes cada transação da base de dados varre a estrutura de armazenamento dos *itemsets* candidatos, incrementando os contadores de suporte de cada *itemsets* candidato pertencente à transação corrente. Logo, a estrutura de armazenamento influencia de forma decisiva no tempo de execução desta fase.

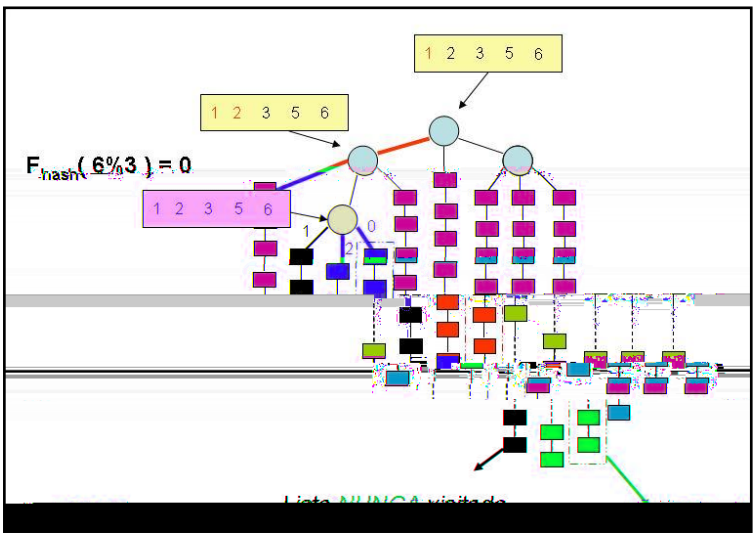


Figura 4.5 : Contagem de Suportes com Árvore *Hash*

Na implementação proposta utiliza-se uma árvore *hash* como apresentado em [3]. Esta estrutura evita a verificação de listas de *itemsets* candidatos que não pertencem à transação corrente, como ilustra a figura 4.5. Isto representa uma redução no número total de *itemsets* candidatos submetidos a cada transação. Porém, esta redução pode ser melhorada através de uma distribuição melhor destes *itemsets* candidatos na árvore. Para isto, é necessário o ajuste de dois parâmetros da estrutura, a função *hash* e a saturação dos nós.

Realizou-se testes variando estes dois parâmetros. E, para facilitar a execução destes testes, incorporou-se outras funções *hash* à implementação e modificou-se a ferramenta de apoio, permitindo a escolha prévia do valor de saturação dos nós e/ou da função *hash* a ser utilizada, como ilustra a figura 4.6.

Figura 4.6 : Ferramenta modificada

Os resultados dos testes mostraram que realmente a variação destes dois parâmetros exerce grande influência no desempenho da fase de contagem dos suportes. Entretanto, a variação do valor do parâmetro saturação tem menos peso no desempenho do algoritmo do que a variação da função *hash* utilizada.

O valor da saturação altera apenas a altura da árvore *hash*. Nessa estrutura, os *itemsets* candidatos são armazenados nas folhas e um nó interno representa apenas uma alternativa de caminho para se chegar a uma folha, onde efetivamente é realizada a contagem dos suportes. A altura máxima da árvore *hash* é estabelecida pelo número da iteração corrente. Por exemplo, para a K-ésima iteração, cada folha da árvore *hash* que ultrapassar o limite de saturação poderá transformar-se num nó interno, desde que seu nível seja menor que K. No caso da folha já pertencer ao nível K, esta armazenará qualquer quantidade de *itemsets* candidatos, independentemente do limite de saturação. Logo, o parâmetro saturação promove uma maior distribuição dos *itemsets* candidatos para grandes valores de K. Porém, nestes casos, a geração dos *itemsets* diminui sensivelmente e, por isso, não é grande a influência do parâmetro saturação na redução do tempo de contagem de suportes, excetuando nas bases de dados que apresentam grandes quantidades de *itemsets* extensos.

Já a função *hash* atua mais na largura da árvore, e, portanto, pode promover uma maior distribuição de *itemsets* candidatos desde as primeiras iterações, influenciando muito mais efetivamente no tempo de contagem dos suportes. Entretanto, para o alargamento da árvore, há um aumento de consumo de memória. Isto leva a uma relação de compromisso que limita a atuação da distribuição de dados.

A estratégia de busca imposta pela árvore *hash* é boa. Entretanto, para grandes quantidades de *itemsets*, formam-se longas listas de dados nas folhas da árvore e, por isso, o rendimento é baixo. Mesmo com as tentativas de uma melhor distribuição dos dados na estrutura, o desempenho do algoritmo não melhora muito. F. *Bondon* em [87] alerta para o comprometimento da eficiência de implementações do APRIORI, devido ao uso de árvores *hash*.

Contudo, decidiu-se não abandonar, ainda, a utilização de árvore *hash* na implementação. Ao invés disso, procurou-se buscar uma outra maneira de armazenar os dados nas folhas da árvore *hash*, pois uma análise simplificada mostra que a complexidade de tempo para a realização da busca de um dado nestas listas é, no pior caso, igual a O(n), onde "n" é o número de *itemsets* da lista.

Então, substituiu-se as listas de *itemsets* das folhas por uma estrutura de busca denominada *SkipList*, apresentada em [88], ilustrada na figura 4.7. Nestas estruturas, os dados, ordenados em ordem crescente, são distribuídos randomicamente em listas de níveis. A busca realizada em uma *SkipLists* apresenta, em média, complexidade de tempo igual a O(lgn), onde "n" é o número de elementos na estrutura.

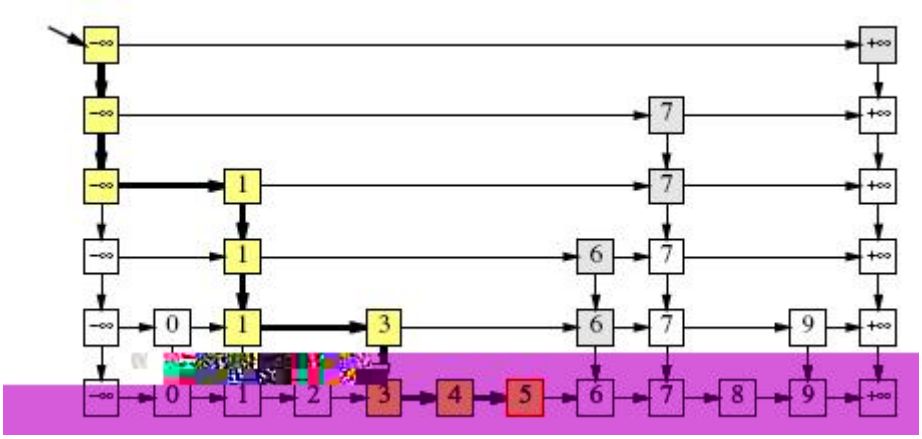


Figura 4.7 : Busca em uma *SkipList*

A incorporação das *SkipLists* na árvore *hash* aumentou o desempenho do algoritmo em várias bases de dados. Entretanto, o consumo de memória aumentou o que inviabiliza a utilização desta estrutura em algumas bases de dados, sobretudo, quando o valor do suporte mínimo é muito baixo.

Nesta altura do desenvolvimento havia algumas soluções particulares para melhora do desempenho do algoritmo. Estas poderiam ser adotadas com algumas adaptações para as bases de dados da aplicação alvo. Entretanto, nenhuma das soluções encontradas apresentam melhora expressiva no desempenho do algoritmo que justifique a sua adoção.

As pesquisas apontavam como a solução mais provável para o problema a utilização de uma árvore de prefixos como estrutura de armazenamento de dados. Em [87], são referenciadas diversas implementações bem sucedidas do algoritmo APRIORI que utilizam árvore de prefixo. A adoção desta estrutura forçaria mudanças mais profundas na codificação da implementação. Além do mais, como não se estava

perseguindo nenhum desempenho ótimo, uma estrutura mais simples, mas que fosse capaz de garantir um bom desempenho na execução da tarefa de mineração de itens freqüentes nas bases de dado da aplicação alvo, seria preferível, tendo em vista a previsão da incorporação na implementação de outras funcionalidades e/ou tarefas de mineração de dados.

4.3.2.3. A idéia da lista de classes

Ao se analisar os *itemsets* candidatos, percebe-se que seria possível classificá-los em grupos distintos. E, como exemplificado na figura 4.8, as listas de *itemsets* candidatos das folhas de uma árvore *hash* armazenam *itemsets* de diferentes grupos.

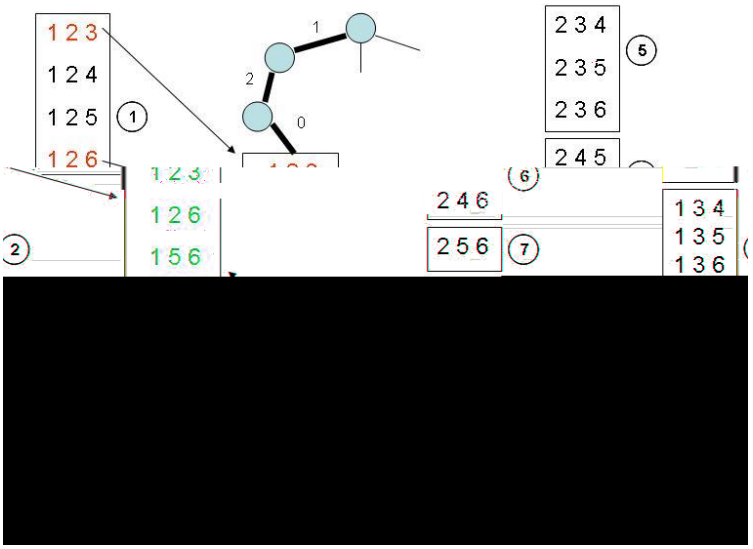


Figura 4.8 : Armazenamento de *itemsets* de grupos distintos

No exemplo da figura 4.8 a função *hash* adotada é a "$f(x) = x \% 3$" (ou módulo de 3). Para o armazenamento de um *itemset* cada elemento do conjunto, na seqüência de sua ordenação, é submetido a essa função *hash*, cujo resultado indica um caminho, até que se chegue a uma folha e este seja armazenado.

Assim, começa a desconfiança de que pudesse haver alguma vantagem na separação destes *itemsets* em grupos distintos, que poderia ser aproveitada para melhorar o desempenho do algoritmo.

Para a formação de um *itemset* candidato de tamanho K são necessários dois *itemsets* freqüentes de tamanho K-1, que possuam os K-2 primeiros elementos iguais. Então, cada junção de dois *itemset* freqüente forma uma classe de *itemsets* candidatos, como a figura 4.9 exemplifica.

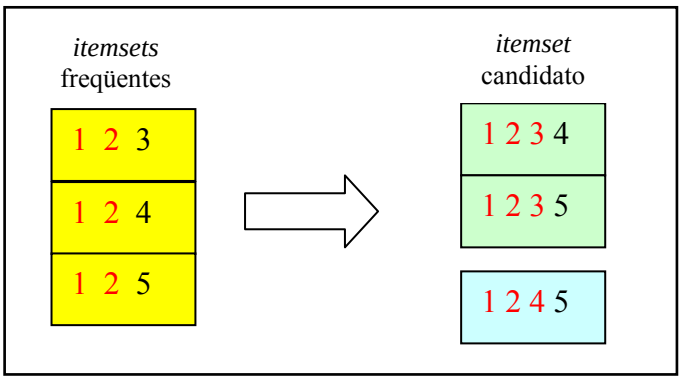


Figura 4.9 : Exemplo de formação de um *itemset* candidato

Foi denominado de prefixo os K-1 primeiros elementos de um *itemset* candidato e de corpo o último elemento do *itemset*. Então, os *itemsets* foram separados em grupos ou classes de prefixos. Em seguida, foi criada uma representação por lista de classes para esses *itemsets* candidatos, como ilustrado na figura 4.10.

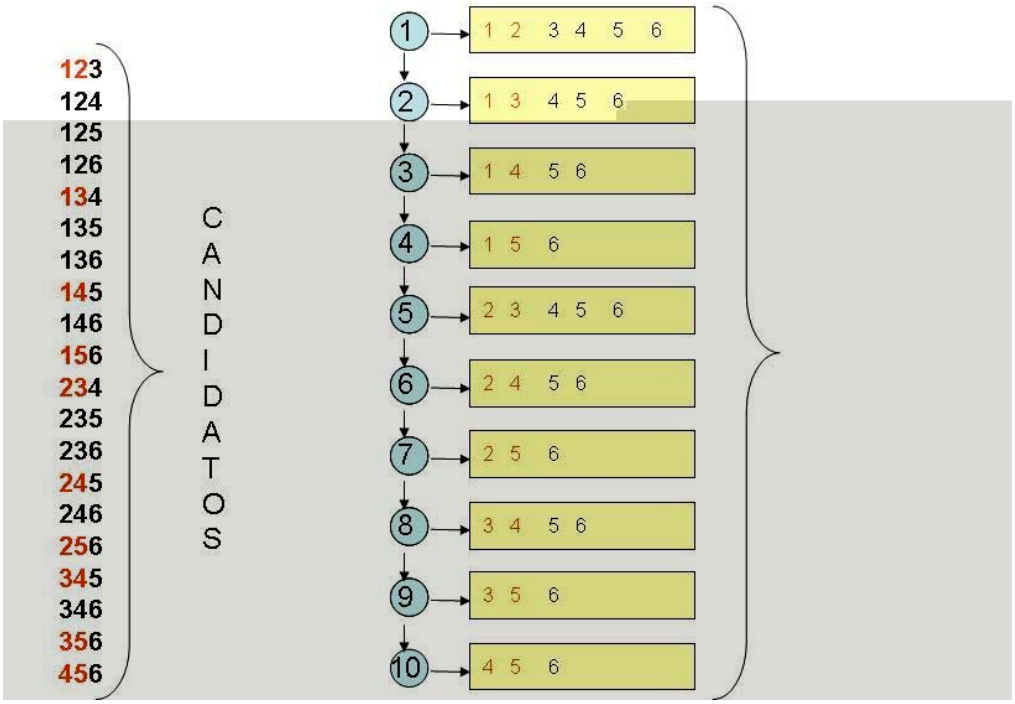


Figura 4.10 : Representação de *itemsets* candidatos por lista de classes

Na estrutura de lista de classes, cada nó guarda as informações referentes aos *itemsets* de sua classe. Como o prefixo é a parte comum, este só é armazenado uma vez em cada classe. Os diferentes *itemsets* da classe são diferenciados apenas pelo seu corpo e estes têm associado um contador de suportes, como ilustrado pela figura 4.11.

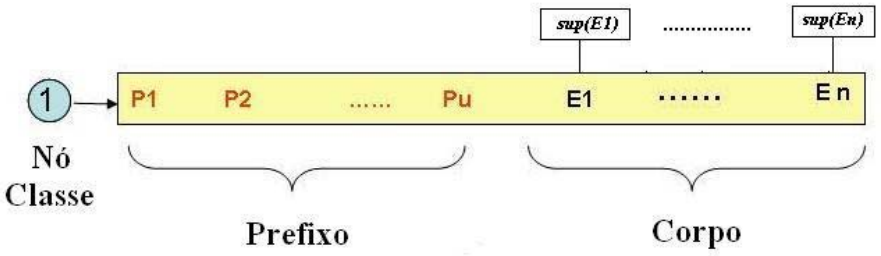


Figura 4.11 : Representação de um nó da Lista de Classes

Esta representação de *itemsets* candidatos é uma forma condensada de representar estes conjuntos de dados. Portanto, promove uma economia no consumo de memória para o armazenamento de candidatos no algoritmo. Além do mais, reduz o número de testes para o reconhecimento de um *itemset*, se comparado com o número de testes gastos na representação clássica de *itemsets* por conjuntos, como é mostrado na figura 4.12.

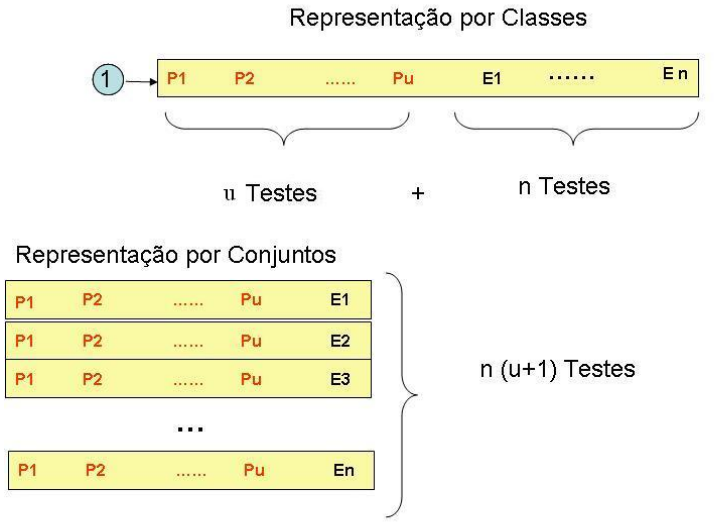


Figura 4.12 : Economia de testes

Foi desenvolvido um algoritmo para a realização da contagem dos suportes dos *itemsets* candidatos, de forma a reduzir o tempo desta tarefa. O algoritmo apresentado na figura 4.13 é submetido a todos os conjuntos de itens das transações que compõem a base de dados.

Algoritmo de Contagem de Suportes

entrada: Transação $_x$: ConjItem

```
1    Lê entrada X [ n ] =  ConjItem
2    para  todo noClasse faça
3       i = n - 1
4       j = 0
5       se  X [ i ]  ≥  noClasse.corpo[ j ]  então
6          se comparaPrefixo ( noClasse , X ) então
7             i = posi ( prefixo +1 )
8             enquanto  ( i  <  n )  faça
9                enquanto ( noClasse.corpo[ j ]  ≤  X [ i ] ) faça
10                  se  noClasse.corpo[ j ]  ==  X [ i ] então
11                     sup ( noClasse.corpo[ j ] ) ++
12                     i ++
13                     j ++
14                  senão  j ++
15               fim enquanto
16               i  ++
17            fim enquanto
18      noClasse=noClasse.prox
19   fim para
20   fim
```

Figura 4.13 : Algoritmo de Contagem de Suporte para Lista de Classes

No passo inicial do algoritmo os itens da transação são armazenados em ordem crescente em X (linha 1). Em seguida, o algoritmo varre a lista de classes atualizando os contadores de suporte de cada nó que pertencem à transação (linhas 2- 19). No início desse processo, o algoritmo verifica se o último item da transação é menor do que o primeiro elemento do corpo do nó que está sendo analisado. Em caso afirmativo, não existe nenhum elemento do corpo suportado pela transação. Então, o algoritmo passa

para o próximo nó da lista (linha 2-5); do contrário, há chances dos *itemsets* do nó pertencerem à transação. Então, primeiramente, a função *comparaPrefixo*( ) verifica se o prefixo do nó está contido na transação. Caso a transação não suporte o prefixo, o algoritmo passa para o nó seguinte da lista de classes. Porém, se o prefixo for suportado pela transação, a função *posi*( ) carrega o indexador "i" com a posição imediatamente superior a posição do último elemento do prefixo em X (linha 7). Desta forma, apenas os itens de X maiores que os elementos do prefixo são analisados. Então, cada item restante de X é comparado com cada elemento de corpo do nó. E, sempre que houver coincidência entre um item de X e um elemento de corpo do nó, este tem seu contador de suporte incrementado, indicando que o *itemset* pertence à transação (linhas 8-17).

A exemplo do que já havia sido feito anteriormente, foi desenvolvida nova interface integrada à implementação do algoritmo com utilização de lista de classes, de modo a facilitar a realização dos testes de depuração. A figura 4.14 mostra a nova interface para introdução dos parâmetros e execução do algoritmo.

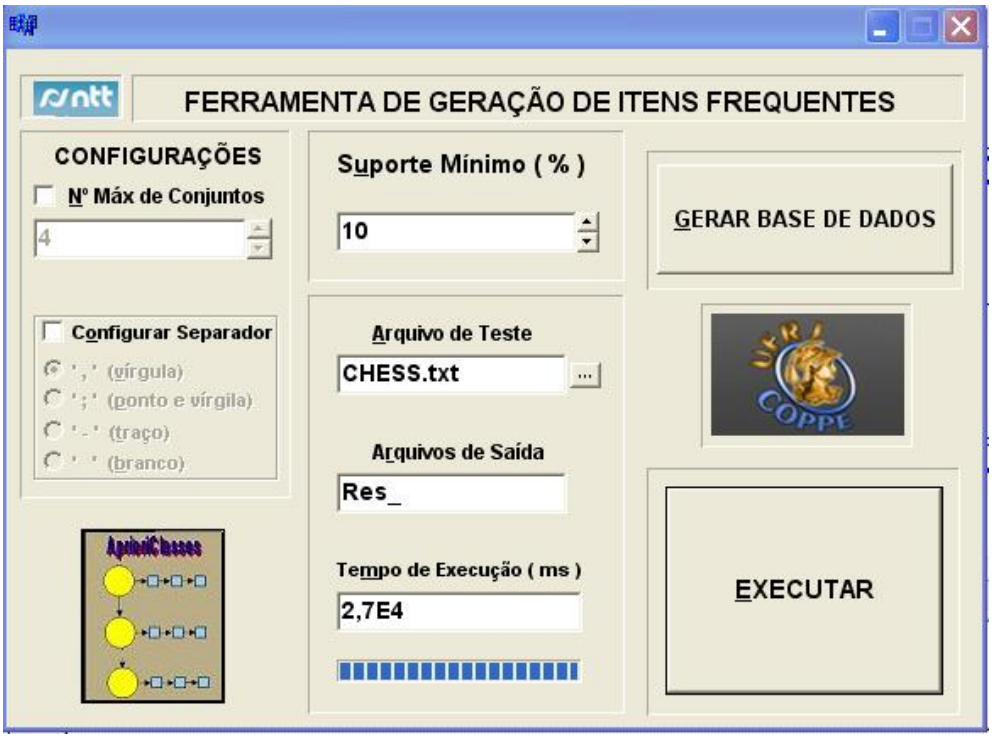


Figura 4.14 : Interface para implementação do APRIORI com lista de classes

4.3.2.4. Alguns resultados parciais

Realizou-se alguns testes com as três implementações do algoritmo e em bases de dados com características distintas. Esses experimentos foram realizados numa máquina com processador Pentium IV de 2GHz com 512 Mbytes de memória RAM e sistema operacional Windows XP. A fim de testar a real contribuição das estruturas no desempenho do algoritmo, não foi utilizada a função de poda na fase de geração dos *itemsets* candidatos e limitou-se em cinco o número de iterações, visto que em algumas instâncias experimentadas, a quantidade de dados envolvidos era superior ao limite de memória disponível.

Dentre as bases utilizadas nestes experimentos está a CHESS, uma base produzida através de formulações matemática, considerada densa, composta por 3.196 transações e 75 itens. Outra base utilizada no experimento foi a MUSHROOM que contém características de várias espécies de cogumelos e é considerada uma base esparsa para grandes valores de suporte mínimo. Entretanto, à medida que estes valores são reduzidos, essa base apresenta um incremento considerável em sua densidade. Ela é composta por 8.125 transações e 119 itens. Utilizou-se também a base sintética T10I4D100K, uma base de grande porte, porém muito esparsa, composta por 100.000 transações e 1000 itens. E, finalmente, foi utilizada a Chaff-SLDM2, uma base da aplicação alvo com dados produzidos através de um simulador de ambiente de Chaff, composta por 94.949 transações e 202 itens. Excetuando as bases de dados militar, que têm caráter sigiloso, as demais bases de dados utilizadas nos experimentos estão disponíveis em [86].

A figura 4.15 mostra o comportamento das três implementações para base de dados CHESS. A implementação com *SkipList* teve melhor desempenho que a implementação com árvore *hash*, apenas quando o número de *itemsets* candidatos era baixo. Porém, a implementação por lista de classes foi a que apresentou melhor desempenho entre as demais implementações nesta base de teste.

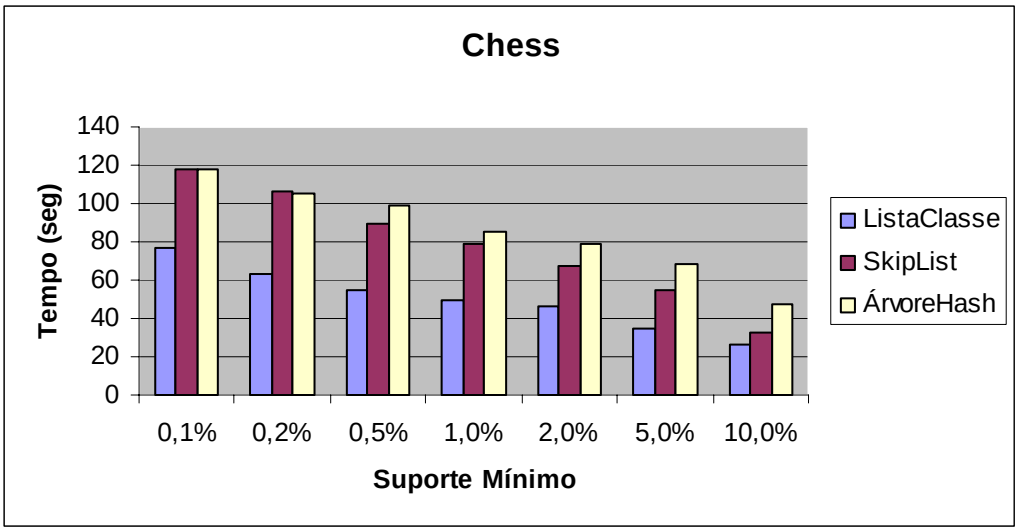


Figura 4.15 : Comportamento das implementações do APRIORI na base CHESS

A figura 4.16 mostra que a implementação por lista de classes também foi a que apresentou melhor desempenho, para base MUSHROOM.

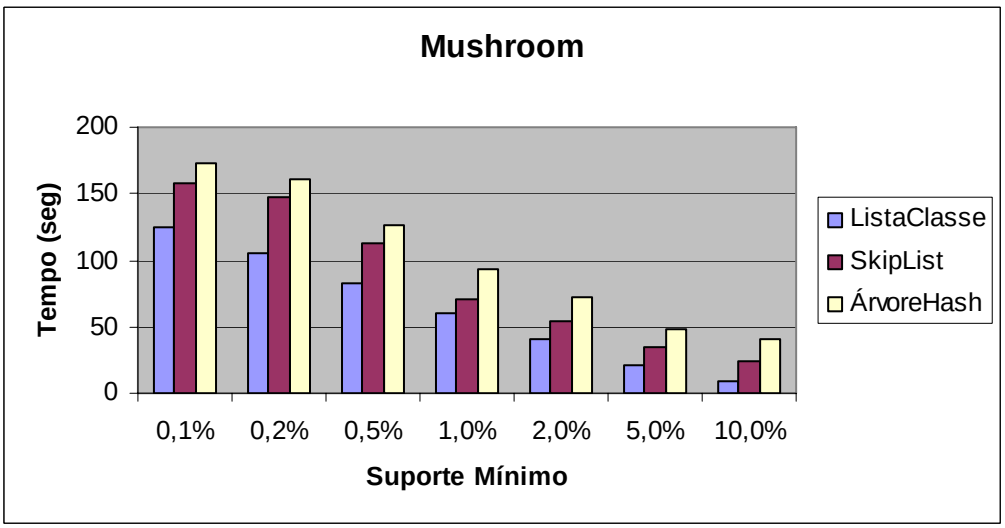


Figura 4.16 : Comportamento das implementações do APRIORI na base MUSHROOM

O mesmo ocorreu para as bases T10I4D100k e Chaff-SLDM2, como ilustra os gráficos da figura 4.17 e o da figura 4.18, respectivamente. Isto demonstra que, independentemente da base de teste utilizada, a implementação do algoritmo apresentou melhor desempenho com o uso de lista de classes como estrutura de armazenamento de *itemsets* candidatos.

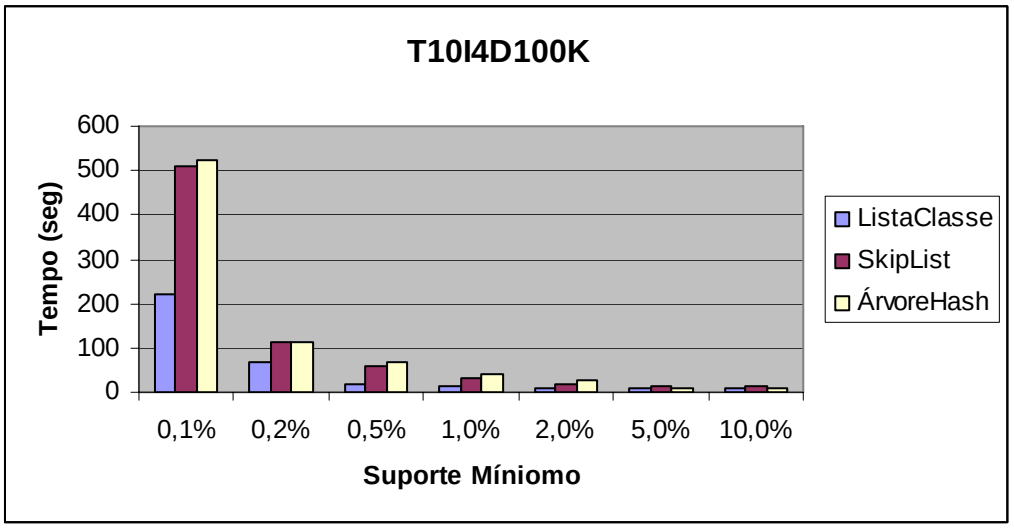


Figura 4.17 : Comportamento das implementações do APRIORI na base T10I4D100K

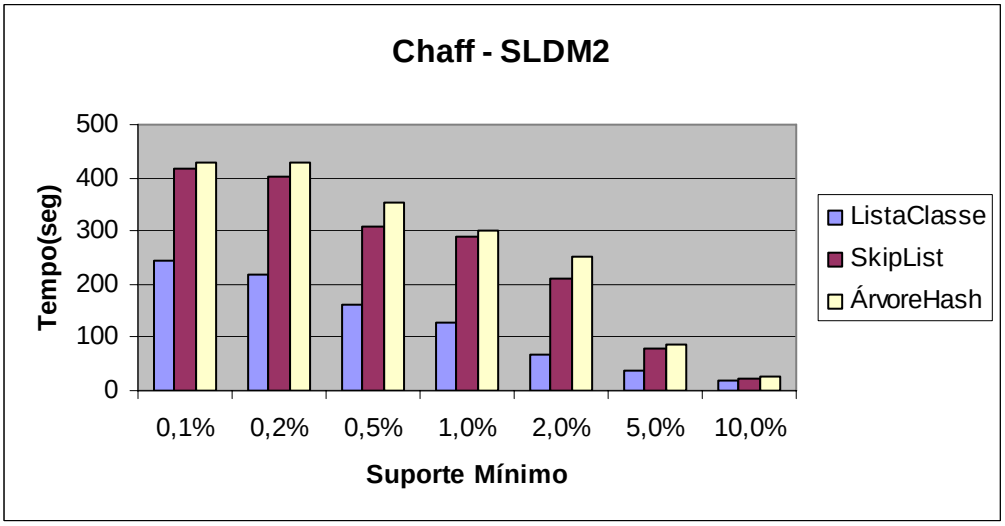


Figura 4.18 : Comportamento das implementações do APRIORI na base Chaff-SLDM2

4.4 IMPLEMENTAÇÃO PARALELA

Nesta seção são mostrados os detalhes da implementação paralela para o algoritmo APRIORI e também são apresentados alguns resultados de experimentos realizados com essa implementação.

4.4.1. A Estratégia de Paralelização do Apriori

A implementação paralela do APRIORI visa a sua execução em ambiente de arquitetura distribuída, isto é, um conjunto de computadores interligados em rede, onde cada computador, ou nó, é composto por um ou mais processadores, com sua própria memória e seu próprio disco. E a comunicação entre esses computadores é feita através de trocas de mensagens com a utilização de *software* especial para sistemas distribuídos. As primitivas de comunicação utilizadas na implementação fazem parte da biblioteca de comunicação MPI (*Message Passing Interface*) [89].

A estratégia desenvolvida para a paralelização do algoritmo APRIORI baseou-se no algoritmo de distribuição de contagem apresentada em [83] e citada no capítulo 3 desta tese. Nesta implementação, toda a base de dados é particionada e armazenada na memória principal de cada máquina. E, em cada iteração do algoritmo, todos os processadores executam o mesmo código. A paralelização dos dados é a forma mais simples de paralelização. Para o algoritmo APRIORI, devido, sobretudo ao seu caráter iterativo essa estratégia é bastante eficiente em sistemas distribuídos, pois reduz o tempo de comunicação entre as máquinas.

Na fase inicial do algoritmo proposto é realizado o particionamento da base de dados e o armazenamento das partições na memória das diferentes máquinas. A figura 4.19 apresenta o algoritmo de particionamento utilizado na implementação. Nesse algoritmo todos os processadores lêem o arquivo que contém a base de dados. A variável *numProc* contém a quantidade de máquinas que executam o algoritmo. As variáveis *idProc* e *idPartição* identificam as máquinas e as partições de dados, respectivamente, através de seus valores inteiros e não negativos. A base de dados e a

quantidade de máquinas são as entradas do algoritmo. A variável de identificação das partições *idPartição* é zerada, indicando o rótulo da primeira partição (linha1). A base de dados é submetida à função *ContaTransação*( ), para definição da quantidade total de transações contidas no arquivo da base de dados. A quantidade total de transações da base é dividida pelo valor de *numProc*, e então, é armazenado em *numElementos* a quantidade de transações de cada partição de dados (linha 2). Esse valor é carregado no contador de transações das partições, *contaPartição* (linha3).

Algoritmo de Particionamento da Base de Dados

entrada: arquivo de Base de Dados: **Base.txt**
quantidade de máquinas : **numProc**

```
1    idPartição =0
2    numElementos= ContaTransação( ) / numProc
3    contaPartição = numElementos
4    para todas as transações da Base de Dados faça
5        se ( contaPartição = 0 ) então
6            idPartição ++
7            contaPartição = numElementos
8        se ( idPartição ≠ numProc – 1 ) então
9            contaPartição - -
10       se ( idProc = idPartição ) então
11           armazena transação
12   fim para
13   fim
```

Figura 4.19 : Algoritmo de Particionamento da Base de Dados

Assim, inicia-se o processo de armazenamento das partições nas memórias das máquinas (linhas 4-12). Para cada transação lida no arquivo verifica-se a que partição ela se destina. Primeiramente, verifica-se o contador *contaPartição*. Se este estiver vazio, todas as transações de uma partição já foram armazenadas. Então, *idPartição* é incrementada, indicando que o processo se repetirá para uma nova partição e o valor do contador *contaPartição* é restaurado (linhas 5-7) . No caso de do contador não estar

vazio, verifica-se o valor de *idPartição*. Caso este valor indique que não é a última partição, decrementa-se o contador *contaPartição* e armazena-se a transação na memória da máquina referente a partição vigente. No caso de ser a última partição, as transações restantes no arquivo de dados são armazenadas na memória da última máquina sem contagem do número de transações da partição. Isto é feito para os casos onde a divisão das transações da base pelas máquinas não é exata. Então a última partição tem uma transação a mais do que as demais, como é exemplificado na figura 4.20, para 3 máquinas e uma base de dados contendo 7 transações.

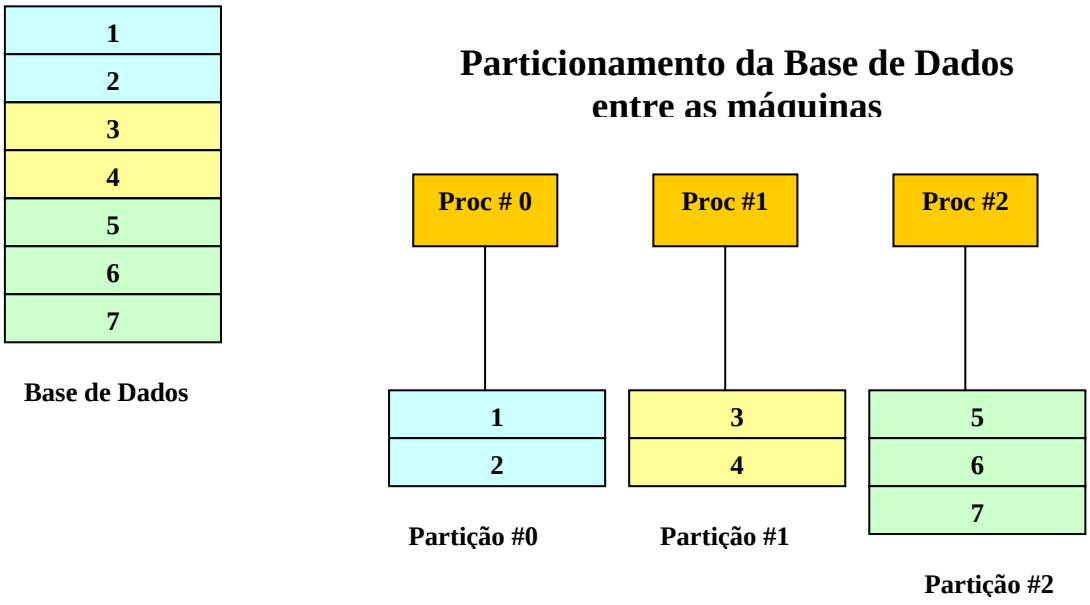


Figura 4.20 : Exemplo de particionamento de uma base dados

Na segunda etapa da implementação paralela do algoritmo APRIORI, após o particionamento da base de dados, são formados os conjuntos candidatos a partir dos *itemsets* freqüentes da iteração anterior. É realizada, então, a contagem dos suportes locais dos candidatos e, através de uma primitiva de redução de soma em uma das máquinas, são obtidos os suportes globais dos candidatos. Em seguida, por meio de uma primitiva de difusão esses valores de suporte globais são transmitidos a todas as demais máquinas que executam o algoritmo. A partir deste ponto, todos os processadores

podem extrair os *itemsets* freqüentes de suas listas de candidatos. A figura 4.21 apresenta o algoritmo utilizado para a paralelização desta tarefa.

Algoritmo Paralelo para Geração de Itens Frequentes

entrada: arquivo de Base de Dados : **Base.txt**
quantidade de máquinas : **numProc**
suporte mínimo **: X %**

```
1   ParticionaBase ( )
2   F1 = FreqF1( )
3   K = 1
4   enquanto ( F K-1 ≠ ∅ ) faça
5       K ++
6       numCandidatos = 0
7       para todos elementos f ∈ F K-1 faça
8           c a= Junta ( f i, f j )
9           se Poda (c a ) é falso então
10              numCandidatos + +
11          CK ⇐ c a
12      fim para
13      ContaSupLocal (CK )
14      Reduce ( SupLocal[0],SupGlobal[0], numCandidatos ,soma ( ), idProc )
15      Bcast (SupGlobal[0]],numCandidatos, idProc )
16      FK= FreqFK( CK )
17  fim enquanto
18  fim
```

saída : $\mathbf{Y}_1^k$ F_K

Figura 4.21 : Algoritmo Paralelo para Geração de Itens Frequentes

O algoritmo da implementação assume que os dados são representados nos arquivos das bases de dados por valores inteiros positivos. Na etapa inicial de particionamento da base de dados, a função *ContaTransação*( ), além de contar o número de transações contidas na base de dados, também acha o maior valor entre os itens que a compõem. Este valor é tomado como o número de *itemset* candidatos de

tamanho 1 e é carregado no contador de candidatos *numCandidatos*. Em seguida, cada processador aloca um registro ou *buffer* de tipos inteiros de tamanho igual ao valor armazenado em *numCandidatos*. Neste momento, assume-se que cada índice ou posição desse *buffer* representa um *itemset* candidato, mesmo que alguns destes valores não ocorram na base de dados ou em alguma de suas partições. Efetua-se, então, a contagem dos suportes locais para cada índice do *buffer*, armazenando os valores de suportes em suas respectivas posições. É alocado um novo *buffer* de tamanho igual ao anterior para o armazenamento dos suportes globais. E, por meio de uma primitiva de redução de soma, estes valores são transmitidos e somados em suas respectivas posições no *buffer* de suporte global da máquina concentradora. Esta, então, transmite a todas as demais máquinas esses valores de suportes globais. Assim, cada posição do *buffer* de suportes globais que possuir valor maior ou igual ao suporte mínimo pré-estabelecido será considerada um *itemset* freqüente de tamanho 1, formando desta maneira o conjunto F_1 (linha 2).

Processo semelhante é executado pela função *ContaSupLocal*( ) (linha 13). Porém, neste caso, os *buffers* alocados têm o tamanho determinado pelo valor do contador *numCandidatos*, atualizado durante a geração de *itemsets* candidatos (linhas 7-11). Os valores dos suportes globais dos *itemsets* candidatos de tamanho K também são obtidos por meio da primitiva de redução de soma e da primitiva de difusão (linhas 14 - 15). Como a ordem de armazenamento dos suportes no *buffer* corresponde à mesma ordem de geração e armazenamento dos *itemsets* candidatos na lista de classes, em *FreqFK*( ) é realizada uma comparação entre os valores de cada posição do *buffer* com o valor do suporte mínimo. Nos casos em que um valor de suporte for menor do que o valor do suporte mínimo, o *itemset* correspondente a esta posição no *buffer* é retirado do nó da lista de classes. Isto é feito através da eliminação do elemento de corpo que representa o *itemset* infreqüente e, se um nó perder todos os elementos de corpo durante este processo, este nó também é removido da lista de classes.

4.4.2. Resultados Parciais da Paralelização

Foram realizados testes com a implementação paralela do algoritmo, a fim de avaliar seu comportamento em diferentes bases de dados, antes de sua utilização como ferramenta de mineração de dados nas bases de dado da aplicação alvo. Nos testes, foram utilizadas as mesmas bases descritas no item 4.3.2.4 deste capítulo. E o suporte mínimo adotado foi 0,1 % . Esses experimentos foram realizados em um *cluster* de PC's com as seguintes características, 16 máquinas Pentium IV - 2.8Ghz - *Hyper Threading* com 1GB de memória RAM; 8 máquinas Pentium IV - 2.4GHz com 1GB de memória RAM e 16 máquinas AMD *Opteron* - 64bits, *Dual-Core* com 2GB de memória RAM.

A definição adotada para o *speed up*, nesta tese, é o aumento de velocidade observado quando se executa um determinado processo em *p* processadores em relação à execução deste processo em 1 processador, ou seja, *speed up* $=(T_1 / T_p)$, onde T_1 é o tempo de execução em 1 processador e *Tp* é o tempo de execução em *p* processadores. A tabela 4.1 apresenta os tempos de execução do experimento.

Bases	**Chess**	**Mushroom**	**T10I4D100K**	**Chaff-SLDM2**
# Proc	***Tempo(seg)***	***Tempo(seg)***	***Tempo(seg)***	***Tempo(seg)***
1	77,3	124,10	222,00	242,78
2	45,47	62,99	111,11	121,57
3	30,92	41,93	74,25	82,02
4	23,42	31,34	56,06	62,09
5	19,82	25,17	44,58	49,75
6	16,45	21,40	37,19	42,08
7	14,58	18,80	32,41	36,34
8	13,8	17,48	29,21	32,28
9	11,71	15,32	26,43	28,87
10	11,04	14,02	24,40	26,11
11	10,04	13,06	22,20	24,04
12	9,31	12,66	20,00	22,38
13	8,99	12,54	18,66	20,75
14	8,78	12,29	17,76	19,58
15	9,66	12,23	16,82	18,39
16	9,78	12,05	15,86	17,22
17	9,91	11,88	15,00	16,46
18	10,31	11,49	14,23	15,76

Tabela 4.1 - Tempo de execução das bases de dados

Outra medida importante é a eficiência, que trata da relação entre o *speed up* e o número de processadores. No caso ideal (*speed up* = *p*), a eficiência seria máxima e teria valor 1 (100%). As figuras 4.22 a 4.29 apresentam os gráficos das relações de *speed up* e os de eficiência nas bases de dados utilizadas no experimento.

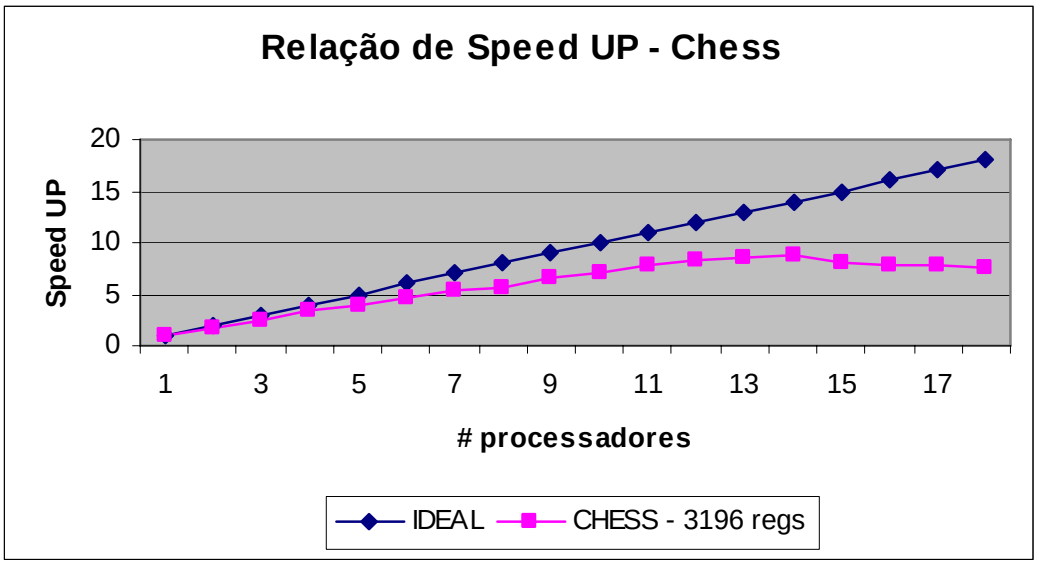


Figura 4.22 : Relação de speed up na base de testes CHESS

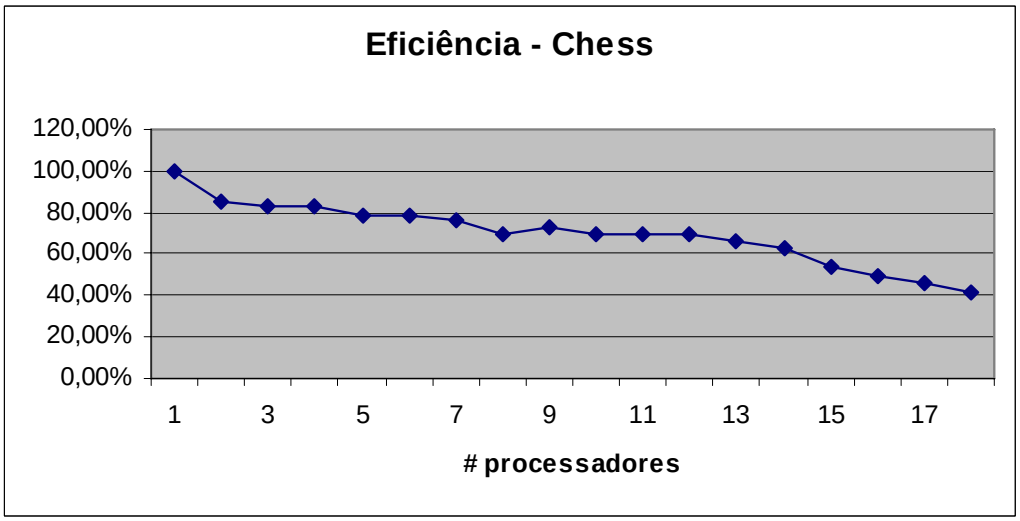


Figura 4.23 : Relação de eficiência na base de testes CHESS

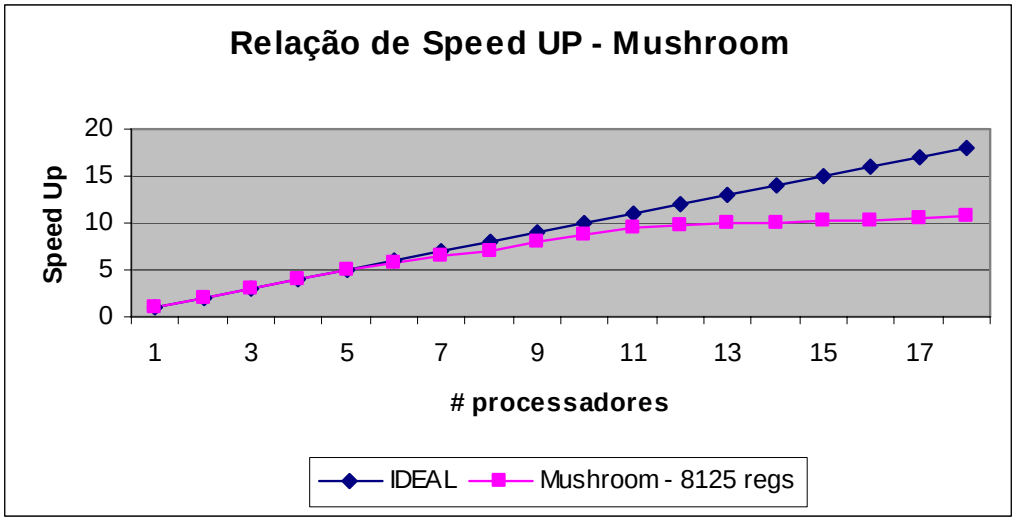


Figura 4.24 : Relação de speed up na base de testes MUSHROOM

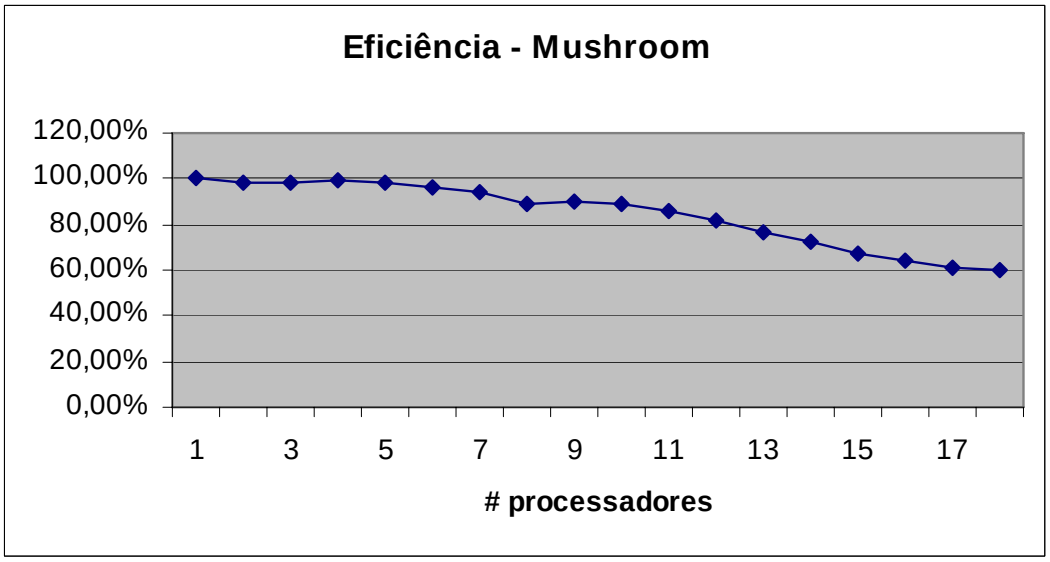


Figura 4.25 : Relação de eficiência na base de testes MUSHROOM

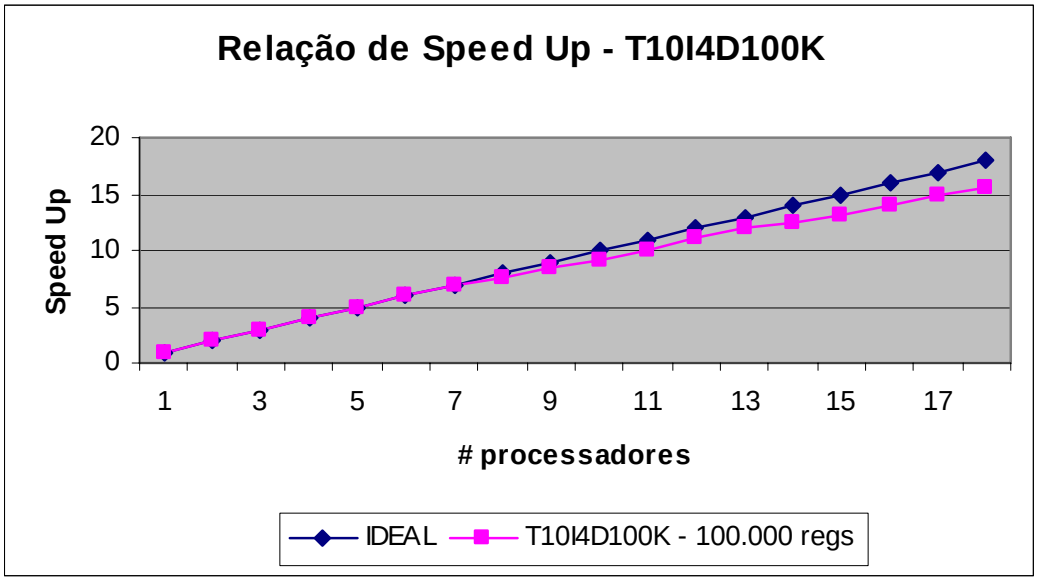


Figura 4.26 : Relação de speed up na base de testes T10I4D100K

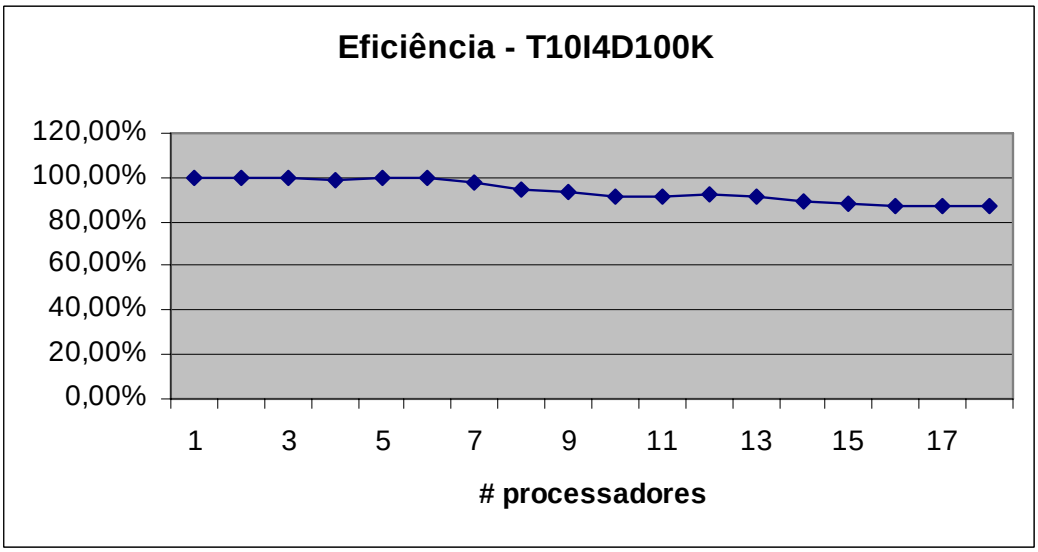


Figura 4.27 : Relação de eficiência na base de testes T10I4D100K

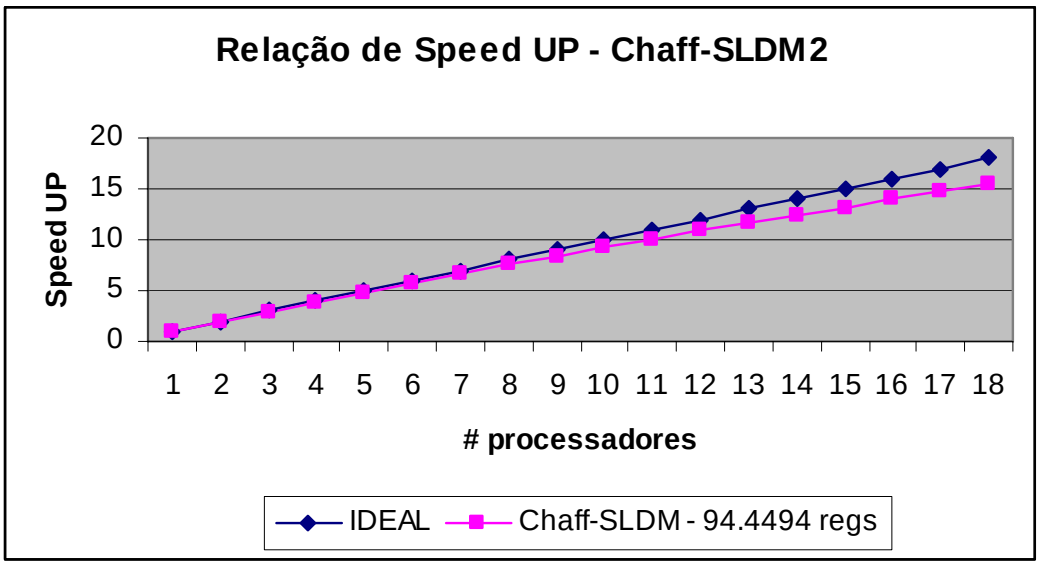


Figura 4.28 : Relação de speed up na base de testes Chaff-SLDM2

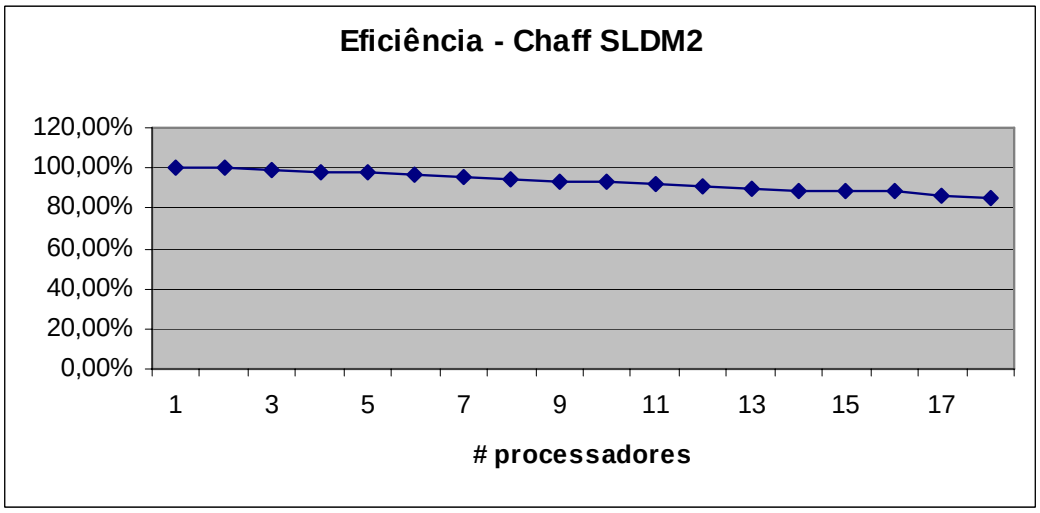


Figura 4.29 : Relação de eficiência na base de testes Chaff-SLDM2

Comparando-se o comportamento das relações de *speed up* e de eficiência nas bases utilizadas no experimento, como é lustrado pelos gráficos das figuras 4.30 e 4.31, pode-se notar que o algoritmo apresenta melhores resultados nas bases onde é maior a quantidade de transações. À medida que a quantidade de transações diminui nas partições, devido ao aumento do número de processadores, tanto o *speed up* quanto a eficiência diminuem.

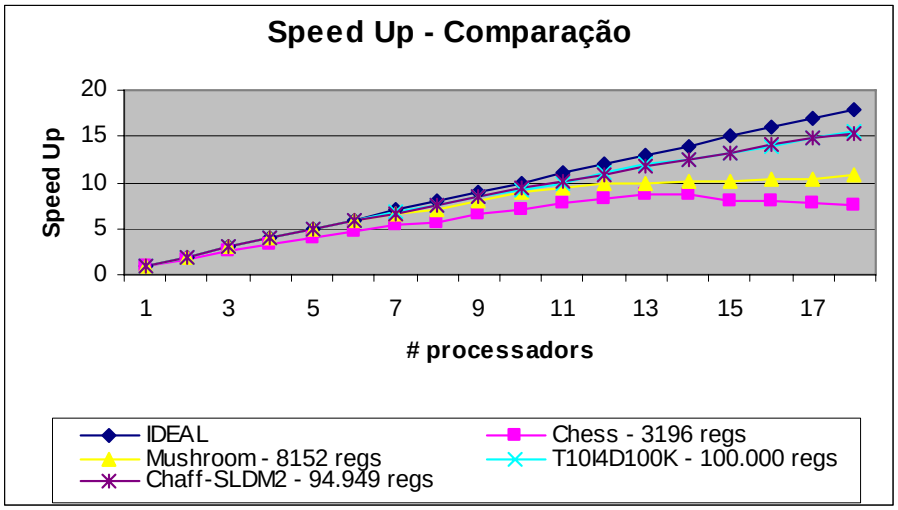


Figura 4.30 : Gráfico de Comparação de *Speed Up*

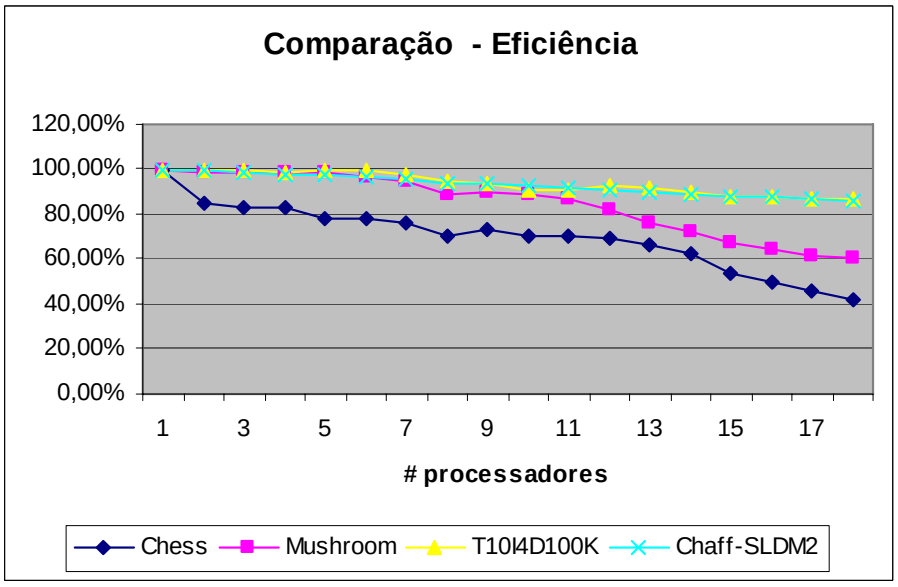


Figura 4.31 : Gráfico de Comparação de Eficiência

4.5 EXTRAINDO AS REGRAS

Nesta seção é apresentado o algoritmo utilizado na implementação para extração das regras de associação. Em seguida, são apresentadas as modificações realizadas na implementação do algoritmo de mineração de itens freqüentes para a realização da tarefa de classificação.

4.5.1. Gerando as Regras de Associação

A tarefa de geração de regras de associação a partir dos *itemsets* freqüentes é muito menos complexa do que a mineração destes conjuntos de dados. Além do mais, para a obtenção de resultados efetivos, isto é, que revelem alguma descoberta interessante é necessário um trabalho posterior envolvendo outras áreas de pesquisa, o que foge ao escopo dessa tese, onde o objetivo maior é possibilitar a extração de regras de associação e de classificação nas aplicações alvo. Portanto, na implementação apresentada nessa tese, não foi realizada nenhuma inovação na fase de geração das regras de associação, isto é, esta fase foi implementada utilizando a idéia originalmente apresentada em [3].

A figura 4.32 mostra o algoritmo utilizado para a geração das regras de associação. Neste algoritmo, primeiramente são geradas regras que possuem como conseqüente um conjunto unitário de elementos. Por exemplo: a regra R_1: 1,2 $\Rightarrow$ 3, o conjunto de antecedente é {1; 2 } e o conjunto de conseqüente é {3}. Isto é realizado pela função *GeraRegra_1*( ), para os *itemsets* freqüentes de tamanho igual ou superior a 2. A função *GeraRegra_1*( ) também gera o conjunto de todos os conseqüentes das regras geradas, armazenado em $Consq_1$(linhas 1-3). A partir de um *itemset* f_k e seu respectivo conjunto de conseqüentes $Consq_1$, a função recursiva *GeraRegras _K*( ) gera todas as demais regras de associação de diversos tamanhos.

A figura 4.33 apresenta o pseudocódigo da função *GeraRegra_1*( ). Esta função recebe como parâmetro de entrada um *itemset* "f_k" de tamanho igual ou superior a 2. Então, para cada elemento "x" desse conjunto de dados, é gerada uma regra do tipo $f_k - \{x\} \Rightarrow x$. Calcula-se a confiança da regra e, se seu valor é igual ou superior ao valor

da confiança mínima pré-estabelecida, a regra é armazenada e insere-se o conseqüente desta regra no conjunto $Consq_1$. Caso contrário, a regra é descartada.

```
Algoritmo para Geração de Regras de Associação
entrada: Itemsets freqüentes : F_K
1       para todo  f_k ∈ F_K   faça
2          se  K > 2  então
3                Consq_1 = GeraRegras_1( f_k )
4                GeraRegras_K  ( f_k ,  Consq_1 )
5          fim se
6       fim para
7    fim
```

Figura 4.32 : Algoritmo de Geração de Regras de Associação

```
Função  GeraRegras_1( f_k )
entrada: Itemsets freqüente : f_k ∈ F_K
1       para cada  { x } ∈ f_k  faça
2            conf = sup ( f_k ) / sup (  f_k  -  { x }  )
3            se  conf  ≥  conf_min  então
4                 armazena regra
5                 Consq_1 ⇐ x
6            senão  descarta regra
7       fim para
8    fim
```

Figura 4.33 : Pseudocódigo da Função *GeraRegras_1(f_k)*

A figura 4.34 apresenta o pseudocódigo da função *GeraRegra_K*( ), que também faz parte do algoritmo de geração de regra de associação. Esta função recebe como parâmetros de entrada um *itemset* freqüente "f_k" de tamanho "*k*" e um conjunto de conseqüentes $Consq_m$ de tamanho "*m*". A partir destes dois parâmetros são geradas, de

forma recursiva, todas as demais regras cujos conseqüentes possuem tamanho superior a 1. Como toda função recursiva, esta também tem um ponto de parada. Logo no início da função é realizado um teste que verifica se o conjunto de *conseqüentes* fornecido na entrada formará regras cujos antecedentes são conjuntos unitários. Se for o caso, a recursão pára, pois estas regras já foram geradas pela função *GeraRegras_1*( ). Portanto, enquanto os conjuntos antecedentes das regras a serem geradas possuírem tamanho superior a 1, novas recursões da função são executadas.

Em cada recursão da função *GeraRegra_K*( ) são gerados novos conjuntos de conseqüentes $Consq_{m+1}$ de tamanho (m+1) através da função de geração de candidatos *GeraCandidatos* ( ), utilizada na primeira fase do algoritmo APRIORI, e do parâmetro de entrada $Consq_m$. Então, todos os subconjuntos de conseqüentes "c_{m+1}" contidos em $Consq_{m+1}$ são usados juntamente com um *itemset* "f_k" para a formação de regras do tipo $f_k - \{ c_{m+1}\} \Rightarrow \{ c_{m+1}\}$. Calcula-se a confiança da regra e, caso o valor calculado seja superior ao valor pré-estabelecido de confiança mínima, a regra é armazenada. Caso contrário, a regra é descartada e o seu conseqüente é removido do conjunto de conseqüentes $Consq_{m+1}$.

```
Função GeraRegras_K( f k, Consq m )

entrada: Itemsets freqüente : f k ∈ F K
         Conjunto de Conseqüentes de tamanho m : Consq m

 1       se  k > m + 1  então
 2           Consq m + 1 = GeraCandidatos( Consq m )
 3           para todos c m+ 1 ∈ Consq m + 1 faça
 4               conf = sup( f k ) / sup(f k - { c m+ 1})
 5               se conf ≥ conf_min então
 6                    armazena regra
 7               senão
 8                    descarta a regra
 9                    retira c m+ 1 de Consq m + 1
10           fim para
11       GeraRegras_K( f k , Consq m + 1 )
12       fim
```

Figura 4.34 : Pseudocódigo Função *GeraRegras_K (f k, Consq m)*

4.5.1.1. Alguns resultados parciais

Foram realizados experimentos com as mesmas bases de dados descritas no item 4.3.2.4 deste capítulo. A plataforma de execução do experimento foi o *cluster* de PC's composto de 16 máquinas Pentium IV - 2.8Ghz - com 1GB de memória RAM; 8 máquinas Pentium IV - 2.4GHz com 1GB de memória RAM e 16 máquinas AMD *Opteron* com 2GB de memória RAM. Nesses experimentos limitou-se em cinco o número de iterações no algoritmo, enquanto variaram-se os parâmetros suporte mínimo e a confiança mínima das regras. A tabela 4.2 mostra a variação do número de regras de associação extraídas para cada base de dados, de acordo com as variações dos valores de suporte mínimo e confiança mínima.

CHESS			MUSHROOM			T10I4D100K			Chaff - SLDM2		
Suporte minimo	Conf minima	Nº Regras	Suporte minimo	Conf minim	Nº Regras	Suporte minimo	Conf minima	Nº Regras	Suporte minimo	Conf minim	Nº Regras
100,0%	100%	0	100,0%	100,0%	0	100,0%	100%	0	100,0%	100,0%	0
90,0%	100%	66	90,0%	100,0%	5	90,0%	100%	0	90,0%	100,0%	4
80,0%	100%	365	80,0%	100,0%	16	80,0%	100%	0	80,0%	100,0%	8
50,0%	100%	5202	50,0%	100,0%	132	50,0%	100%	0	50,0%	100,0%	114
30,0%	100%	19317	30,0%	100,0%	1168	30,0%	100%	0	30,0%	100,0%	511
20,0%	100%	40936	20,0%	100,0%	6181	20,0%	100%	0	20,0%	100,0%	953
10,0%	100%	88998	10,0%	100,0%	24890	10,0%	100%	0	10,0%	100,0%	3101
1,0%	100%	292173	1,0%	100,0%	282394	1,0%	100%	0	1,0%	100,0%	38205
0,1%	100%	492430	0,1%	100,0%	550308	0,1%	100%	272	0,1%	100,0%	101665
0,1%	90%	916675	0,1%	90%	587279	0,1%	90%	30490	0,1%	90%	107995
0,1%	80%	1168369	0,1%	80%	627319	0,1%	80%	38775	0,1%	80%	116995
0,1%	70%	1396789	0,1%	70%	654664	0,1%	70%	42658	0,1%	70%	129926
0,1%	60%	1648574	0,1%	60%	698282	0,1%	60%	45473	0,1%	60%	141132
0,1%	50%	1866805	0,1%	50%	961718	0,1%	50%	47801	0,1%	50%	157316

Tabela 4.2 - Número de regras de associação nas diversas bases de dados

A base de dados CHESS, por ser uma base densa, apresentou um grande número de regras de associação, mesmo para elevados valores de suporte mínimo e confiança mínima. A base de dados MUSHROOM se mostrou mais densa, isto é, com muitas regras de associação, quando o valor do suporte mínimo ou da confiança mínima diminuía. A base T10I4D100K é uma base bem esparsa. Portanto, só foi possível a obtenção de regras de associação com baixos valores de suporte mínimo. A base Chaff-SLDM2 apresentou comportamento semelhante à base MUSHROOM, isto é, as associações aumentaram à medida que o suporte mínimo era reduzido.

4.5.2. Gerando as Regras de Classificação

O classificador implementado, nesta tese, é baseado no algoritmo CBA, apresentado na seção 3.5.2.1, do capítulo 3. Esse algoritmo gera regras de classificação baseado em associações, técnica utilizada com sucesso em implementações seriais. Como contribuição desta tese é apresentada uma implementação paralela deste algoritmo. O seu desenvolvimento se deu em duas etapas, a serial e a paralela. Na implementação serial do algoritmo, foram realizadas modificações nas rotinas de mineração de itens freqüentes, para possibilitar a geração de regras de classificação e, também, a implementação do algoritmo de construção do classificador. Na segunda etapa desse desenvolvimento foi realizada a paralelização nas fases de geração de regras e de construção do classificador.

4.5.2.1. Geração de regras de classificação

A geração de regras de classificação, como descrita no algoritmo CBA, é uma adaptação das rotinas de mineração de itens freqüentes, isto é, ao invés de realizar a mineração de *itemsets* freqüentes, realiza-se a mineração de itens-regras freqüentes. Um item-regra é na verdade um *itemset* especial, composto por valores de atributos e cujo último elemento é um valor representante de uma classe. As regras de classificação distinguem-se das regras de associação por apresentarem sempre como conseqüentes um conjunto unitário representante de uma classe. Assim, por exemplo, seja o conjunto de dados I ={a_1, b_2, c_1}, onde a_1 é um valor do atributo "A", b_2 um valor do atributo "B" e c_1 uma classe. As regras de classificação possíveis de serem extraídas desse conjunto de dados são: a_1, $b_2 \Rightarrow c_1$ ou $a_1 \Rightarrow c_1$ ou ainda $b_2 \Rightarrow c_1$. Mas nunca $a_1 \Rightarrow b_2$, c_1 ou a_1, $c_1 \Rightarrow b_2$ como ocorre nas regras de associação.

A geração de regras foi implementada, nesta tese, de duas maneiras distintas. A primeira, chamada de Gerador de Regras I, é totalmente baseada no algoritmo CBA, isto é, considera a mineração de itens-regras. Nessa implementação foram utilizadas as funções desenvolvidas para implementação do algoritmo APRIORI, com algumas alterações.

A primeira alteração nesta implementação foi a introdução da função *LeAtributos*( ). Essa função lê um arquivo texto contendo as informações dos diversos atributos e das classes que compõe a base de dados de treinamento. Desta maneira é possível associar os valores da base de treinamento a seus respectivos atributos, visto que um atributo pode assumir diversos valores distintos.

A outra alteração realizada na implementação da mineração de itens freqüentes foi a substituição da função *Junta*( ), responsável pela geração dos *itemsets* candidatos no APRIORI, pela função *JuntaItemRegra*( ). Esta nova função também faz a junção de dois conjuntos de dados, porém apenas realiza junção entre itens-regras, ou seja, o último elemento de um conjunto candidato gerado por esta função é sempre uma classe. Além do mais, baseado nas informações obtidas pela execução da função *LeAtributos*( ), não há geração de itens-regras que possuam valores pertencentes a um mesmo atributo. Estas alterações reduzem bastante a geração de conjuntos candidatos.

Utilizou-se a como estrutura de dados para o armazenamento dos itens-regras a lista de classes, mesma estrutura usada na implementação da mineração dos itens freqüentes. A figura 4.35 ilustra a representação de dois nós de uma lista de classes contendo itens-regras. Note que, neste caso, os elementos do chamado corpo do nó são valores de classe e o prefixo do nó valores de atributos, que representam os antecedentes das regras.

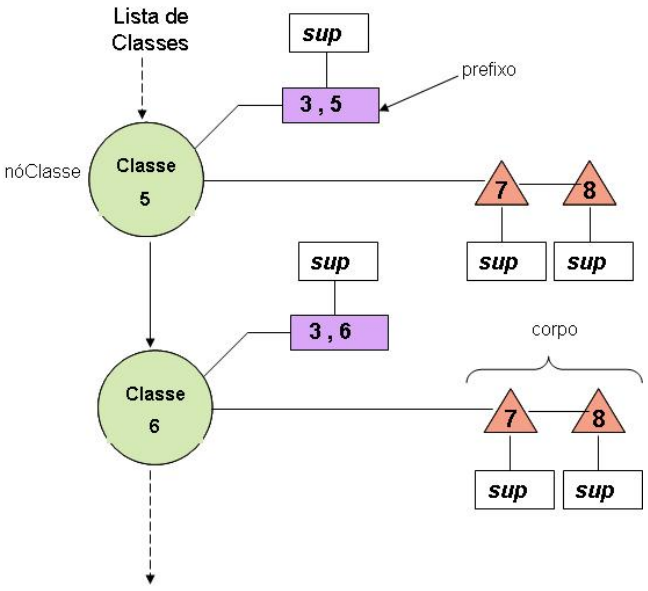


Figura 4.35 : Armazenamento de itens-regras na lista de classes no Gerador I

Esta estrutura de dados também foi alterada com a inclusão de um contador de suporte para os prefixos. Essa alteração facilita o cálculo da confiança de um item-regra, pois cada elemento do corpo do nó já possuía um contador de suporte da regra. Logo, para calcular a confiança de uma regra basta fazer a relação entre os suportes armazenados na estrutura.

A função *JuntaItemRegra*( ), responsável pela geração de itens-regras candidatos, realmente produz muito menos conjuntos de candidatos do que a função *Junta*( ) do APRIORI, não só pelo fato de promover a união apenas de valores de atributos distintos, mas também devido a restrição de formar exclusivamente candidatos que possuam no último elemento valores de classe. Porém, esta tarefa pareceu um tanto quanto onerosa em termos computacionais. Assim, foi desenvolvida uma outra implementação para a geração de regras de classificação, denominada Gerador de Regras II.

Nesta nova idéia os *itemsets* candidatos são gerados quase que da mesma forma que na geração de *itemsets* candidatos do APRIORI, excetuando pelo fato, de que se manteve a restrição de não formar *itemsets* com valores representantes de um mesmo atributo. Então, a nova função *JuntaItem*( ) gera conjuntos candidatos com o último elemento de qualquer valor, e não exclusivamente de valores de classe. Isto acarreta na geração de regras genéricas. Portanto, foi alterada a função *GeraRegras*( ), para eliminação das regras com conseqüentes que não sejam representantes de classes. A figura 4.36 ilustra a representação de um nó da lista de classes com esses *itemsets*.

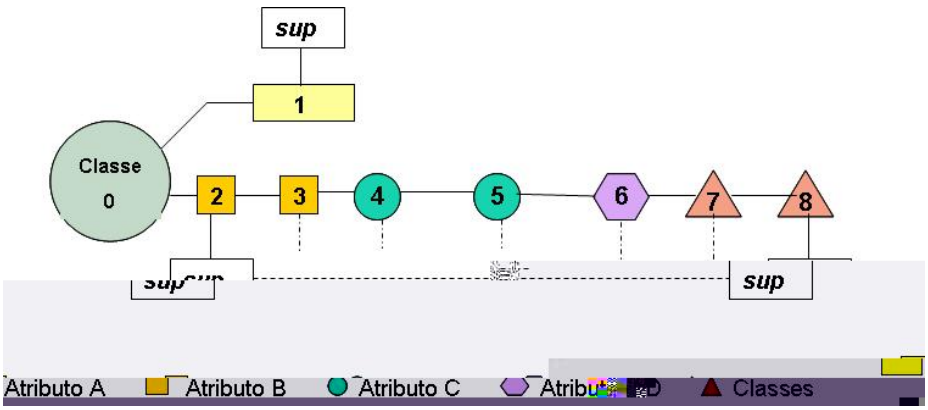


Figura 4.36 : Armazenamento de *itemsets* na lista de classes no Gerador II

Uma outra diferença apresentada na implementação do Gerador de Regras II é que nesta implementação é possível o aproveitamento da contagem de suportes de uma iteração anterior, isto é, no processo de junção entre *itemsets* freqüentes de uma iteração anterior para a formação dos *itemsets* candidatos da iteração corrente, o suporte dos prefixos do nó que contém os novos candidatos é carregado diretamente dos elementos de corpo do nó da iteração anterior, como mostra a figura 4.37.

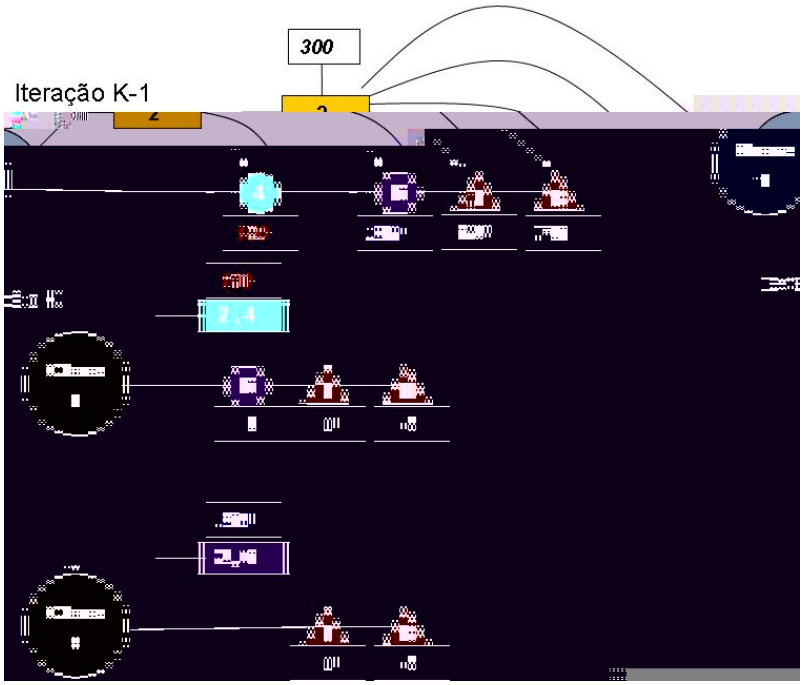


Figura 4.37 : Aproveitando os suportes da iteração anterior

No exemplo da figura 4.37 a iteração 2 apresenta um nó da lista de classes de *itemsets* freqüentes cujo prefixo é {2} e os elementos de corpo são {4;6;7;8}. Cada um destes elementos, juntamente com o prefixo do nó, representa, respectivamente, os *itemsets* {(2,4);(2,6);(2,7) e (2,8)}. No processo de junção destes *itemsets* freqüentes para a formação dos *itemsets* candidatos da iteração 3 são formados os seguintes *itemsets*: {(2,4,6); (2,4,7);(2,4,8);(2,6,7);(2,6,8)}. Repare que o *itemset* (2,7,8) não é formado, pois apresenta dois elementos representantes de classes. Após a junção dos *itemsets* candidatos, estes são armazenados na lista de classes. O primeiro nó a se formar contém os *itemsets* {(2,4,6); (2,4,7);(2,4,8)}, e portanto, o prefixo deste nó é o *itemset* {2,4}, cujo suporte se encontra armazenado no elemento de corpo{4} da lista de classes da iteração anterior. O mesmo ocorre quando da formação do próximo nó com

os *itemsets* {(2,6,7);(2,6,8)}, para o prefixo {2,6}. Neste momento, os contadores dos suportes dos elementos de corpo estão zerados, pois estes representam *itemsets* candidatos cujos suportes ainda serão computados.

4.5.2.2. A construção do classificador

Com a implementação do gerador de regras de classificação, foi possível a geração de um conjunto completo de regras de classificação a partir das bases de dados de treinamento. Entretanto, para classificar as novas instâncias de uma base de dados é necessária a seleção de um conjunto mínimo de regras que garanta uma precisão satisfatória de seus resultados. Para a construção do classificador adotou-se a idéia do algoritmo apresentado na seção 3.5.2.1, o CBA-CB. Neste algoritmo, as regras geradas na primeira etapa da implementação são ordenadas e selecionadas a partir de um critério de precedência para a formação do classificador. A figura 4.39 mostra o algoritmo desenvolvido e utilizado na implementação desse classificador.

A primeira etapa do algoritmo de construção do classificador é a ordenação das regras. A função *Ordena*( ) realiza a ordenação dessas regras em ordem decrescente de seus suportes. Em seguida, faz a segunda ordenação destas regras em ordem decrescente de confiança, já que o critério de precedência é a maior confiança, o maior suporte e o menor conjunto de antecedentes[5]. Após a ordenação das regras é criado um vetor contador, o *vetClasse*[ ], onde cada posição desse vetor corresponde a uma classe da base de treinamento (linhas 1-2). Então, cada regra "r" do conjunto ordenado de regras é testada em todas as instâncias "d" da base de dados (linhas 3-18). Primeiramente, verifica-se se o antecedente da regra "r" está contido em "d". Se for o caso, e a regra "r" classificar corretamente "d", marca-se a regra. Mas se "r" não classifica a instância "d", incrementa-se o contador de erros da regra e incrementa-se, ainda, a posição do *vetClasse*[] referente a classe da instância "d". E, no caso do antecedente de "r" não estar contido em "d", incrementa-se a posição do contador referente a classe da instância "d" (linhas 5-10). Se a regra foi marcada, retira-se "d" da base de treinamento (linha 12) e armazena-se *r.classeDefault* o valor da classe padrão associado a "r".

[5] O 3º critério da precedência, usado quando há empate nos dois primeiros critérios, para seleção de uma regra é a escolha da regra criada primeiro. Como estas são formadas em ordem crescentes de elementos dos conjuntos de seus antecedentes, aqui é sumarizado o 3º critério como menor conjunto de antecedente.

```
                    Algoritmo de Construção do Classificador
entrada: conjunto de regras: Regras

1       R_ordenado= ordena(Regras)
2       vetClasses[numClasses]
3       para cada regra r ∈ R_ordenado em seqüência faça
4           para cada caso d ∈ D faça
5               se r.prefixo ⊂ d então
6                   se r.classe ∉ d então
7                       r.erro ++
8                       vetClasse[d.classe ] ++
9                   senão marque r
10              senão vetClasse[d.classe ] ++
11              se r é marcado então
12                  retira d da base D
13                  r.classeDefault=Max(vetClasse[ ])
14                  erroDefault = AchaErro(vetClasse[ ],r.classeDefault )
15                  r.erro = r.erro + erroDefault
16              se D = ∅ então fim para 1
17          fim para 2
18      fim para1
19      regraCorte = ∞
20      para cada regra r ∈ R_ordenado em seqüência faça
21          se r. marca ≠ 0 então
22              se r.erro < regraCorte então
23                  regraCorte = r.erro
24                  ultimaRegra =r
25          senão descarta regra r
26      fim para 3
27      descarta todas as regras r ∈ R_ordenado após ultimaRegra
28      Classificador C = R_ordenado + ultimaRegra .classeDefault
29      fim
```

Figura 4.38 : Algoritmo para construção do classificador

Este valor representa a classe com a maior contagem em *vetClasse*[]. Calcula-se o erro total do classificador, isto é, a função *AchaErro*( ) acha a soma das contagens de todas as posições do vetor *vetClasse*[], exceto a contagem referente à classe padrão. Esse resultado é somado aos erros cometidos pela regra e, então, o total de erros é acumulado

em *r.erro*. O processo se repete até que não haja mais instâncias na base de dados ou termine as regras a serem testadas (linha 13 -18).

Terminada a fase de avaliação das regras, cada regra do conjunto de regras geradas tem uma marca se classificou pelo menos um caso e apresenta uma classe padrão e uma contagem de erros associados à regra. Então, todas as regras que não foram marcadas na fase de avaliação são descartadas e é buscada a primeira regra da seqüência que apresenta o menor erro. Esta regra é considerada a regra de corte, isto é, são descartadas todas as regras que a sucedem. Desta forma, o classificador é composto das regras que não foram eliminadas e da classe padrão associada à regra de corte (linhas 19-29).

4.5.2.3. Outras ferramentas de apoio

Para facilitar os testes de desenvolvimento e demais testes envolvendo as implementações descritas, foi desenvolvida uma interface para as implementações do classificador. A figura 4.39 apresenta esta interface, onde é possível a definição prévia do arquivo da base de teste, do suporte mínimo, da confiança mínima, entre outras facilidades para a realização de testes do classificador.

.

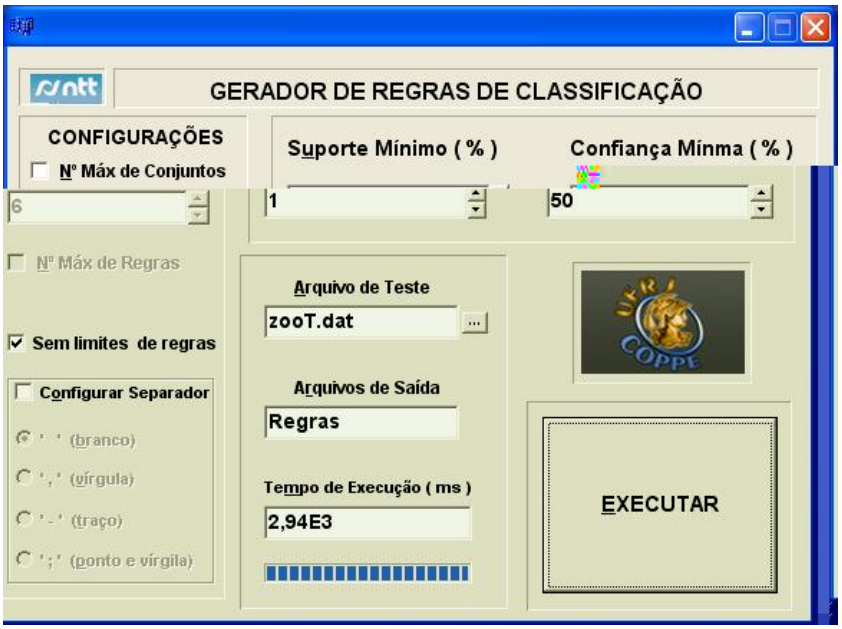


Figura 4.39 : Aproveitando os suportes da iteração anterior

4.5.2.4. Alguns experimentos parciais

Após as implementações do Gerador de Regras de Classificação I e do Gerador de Regras de Classificação II, foram realizados alguns testes para confrontar os desempenhos das duas implementações. Esses experimentos foram realizados utilizando valores de suporte mínimo igual a 1% e confiança mínima de 50%. O número máximo de regras geradas foi limitado em 80.000 regras e todos os atributos contínuos foram discretizados. Os testes foram executados numa máquina com processador Pentium IV de 2GHz com 512 Mbytes de memória RAM e sistema operacional Windows XP, em diversas bases de dados de treinamento, cujas características são apresentadas na tabela 4.3.

Bases de Dados	Nº Atributos Discretos	Nº Atributos Contínuos	Nº Classes	Total de Atributos	Nº Instâncias	Total de itens
aneal	32	6	6	39	898	131
cars	2	4	3	7	392	61
Cleve	8	6	2	15	303	31
crx	9	6	2	16	690	56
german	13	7	2	21	1000	67
heart	0	13	2	14	270	24
hepati	13	6	2	20	155	38
hypo	18	7	2	26	3163	55
labor	8	8	2	17	57	57
sonar	0	60	2	61	208	83
tic,tac	9	0	2	10	958	29
vehicle	0	18	4	19	846	75
waveform	0	21	3	22	5000	109
zoo	18	0	7	19	101	43

Tabela 4.3 - Características das bases de dados da classificação

A tabela 4.4 e o gráfico da figura 4.40 apresentam os tempos totais das duas implementações para a geração do conjunto completo de regras e para construção do classificador, nas diversas bases de treinamento.

base	GR I tempo(seg)	GR I I tempo(seg)	NºRegras
aneal	20,15	14,26	42
cars	0,26	0,24	30
Cleve	1,29	1,38	102
crx	30,97	15,99	196
german	22,06	42,81	290
heart	0,48	0,38	78
hepati	14,22	4,55	42
hypo	45,89	58,39	58
labor	0,27	0,16	15
sonar	142,45	27,75	58
tic,tac	2,3	1,67	8
vehicle	137,01	58,78	205
waveform	185,36	129,4	982
zoo	8,84	2,82	8

Tabela 4.4 - Tabela com os tempos de geração de regras de classificação

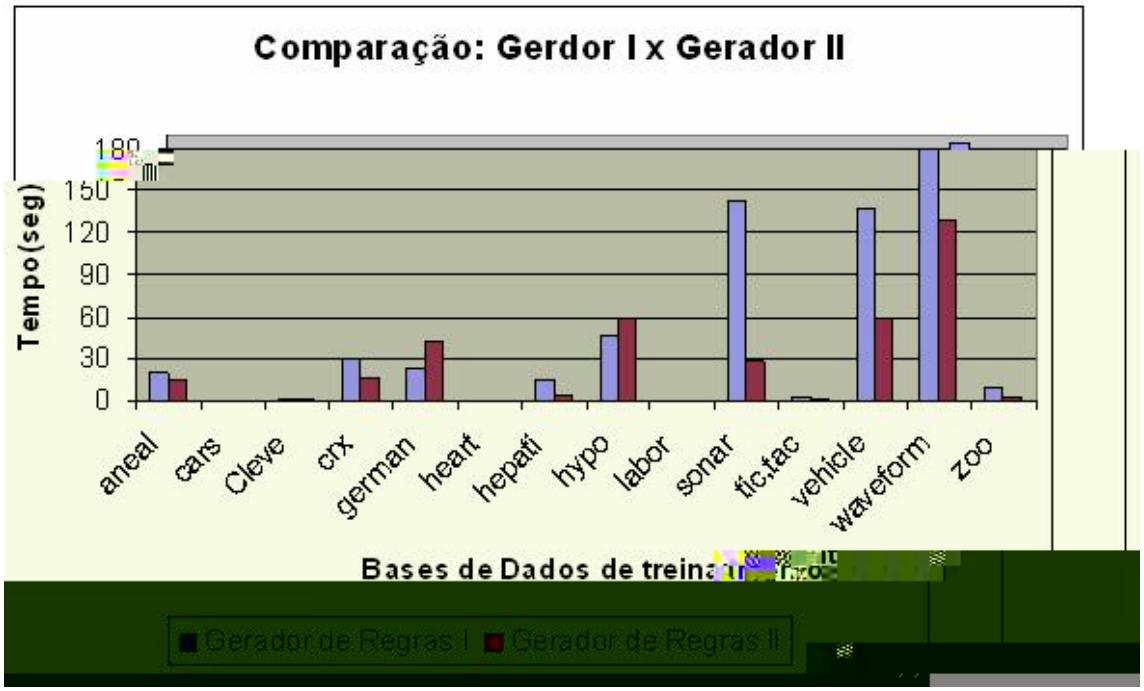


Figura 4.40 : Comparando o Gerador de Regra I e Gerador de Regra II

Os resultados apresentados na tabela 4.4 mostram que, na maioria das bases de dados experimentadas, a implementação do Gerador de Regras II apresentou melhor desempenho. Entretanto, é importante ressaltar que ambas as implementações geram o mesmo conjunto de regras, não havendo nenhum comprometimento da eficiência do classificador quanto ao processo de geração de regras utilizado.

Embora os classificadores construídos com base na técnica de classificação associativa apresentem bons resultados em termos de precisão das classificações, como demonstrado em [10]. Realizou-se testes de validação cruzada em amostragens de 10% (*10-fold-cross validation)* nas bases de dados e verificou-se que a escolha prévia dos valores dos parâmetros suporte mínimo e confiança mínima influenciam muito na qualidade dos classificadores produzidos. Esta, talvez, seja uma das maiores dificuldades de aplicação desta técnica, pois as bases e dados relacionais reais, diferentemente das bases de dados transacionais, guardam muitas correlações entre os dados, característica que para baixos valores escolhidos de suporte mínimo pode levar a uma explosão combinatória de regras, enquanto altos valores de suporte mínimo podem deixar de incluir regras importantes. A tabela 4.5 apresenta a taxa média de erros percentual encontrada para casa base de dados utilizada no experimento.

Bases de Dados	**Nº Instâncias**	**Total de itens**	**Nº de Regras**	**Taxa de erros(%)**
aneal	898	131	42	2,09
cars	392	61	30	1,03
Cleve	303	31	102	16,80
crx	690	56	196	14,62
german	1000	67	290	3,46
heart	270	24	78	18,13
hepati	155	38	42	18,18
hypo	3163	55	58	1,47
labor	57	57	15	13,67
sonar	208	83	58	21,59
tic,tac	958	29	8	1,24
vehicle	846	75	205	31,77
waveform	5000	109	982	19,52
zoo	101	43	8	2,91

Tabela 4.5 - Número de regras de associação nas diversas bases de dados

4.6 O CLASSIFICADOR PARALELO

Nesta seção são apresentados os detalhes da paralelização do classificador, isto é, são mostradas as estratégias utilizadas para geração em paralelo de todo o conjunto de regras da classificação e, também, como se deu a paralelização da fase de seleção das regras para a construção do classificador.

4.6.1. Paralelização da Etapa de Geração de Regras de Classificação

A implementação paralela do classificador visa à execução em ambiente de arquitetura distribuída e se utiliza de primitivas da biblioteca de comunicação MPI (*Message Passing Interface*) [89]. A estratégia adotada para a paralelização do algoritmo de geração de regras de classificação é semelhante à utilizada na paralelização da mineração de itens freqüentes. A base de dados é particionada e armazenada na memória principal de cada máquina. Em cada iteração do algoritmo todos os processadores executam o mesmo código.

O particionamento da base de dados e o armazenamento das partições na memória das diferentes máquinas são realizados na fase inicial do algoritmo. A distribuição dos dados é executada da mesma maneira que foi descrita para o algoritmo paralelo de mineração de itens freqüentes na seção 4.4.1 deste capítulo.

Após o particionamento da base de dados, são formados os conjuntos de itens-regras candidatos, a partir dos itens-regras freqüentes da iteração anterior. É realizada, então, a contagem dos suportes locais dos itens-regras candidatos e, através de uma primitiva de redução de soma em uma das máquinas, são obtidos os suportes globais dos itens-regras candidatos. Em seguida, por meio de uma primitiva de difusão esses valores de suporte globais, são transmitidos a todas as demais máquinas que executam o algoritmo. A partir deste ponto, todos os processadores podem extrair os itens-regras freqüentes de suas listas de candidatos. A figura 4.41 apresenta o algoritmo utilizado para a paralelização desta tarefa.

Na etapa inicial do algoritmo acha-se o maior valor entre os itens que compõem a base de dados de treinamento. Em seguida, cada processador aloca um registro ou

buffer de tipos inteiros de tamanho igual a esse maior valor encontrado. Neste momento, assume-se que cada índice ou posição desse *buffer* representa um item da base de dados, mesmo que alguns destes valores não ocorram na base de dados ou em alguma de suas partições. Efetua-se, então, a contagem dos suportes locais para cada índice do *buffer*, armazenando os valores de suportes em suas respectivas posições. É alocado um novo *buffer* de tamanho igual ao anterior para o armazenamento dos suportes globais. E, por meio de uma primitiva de redução de soma, estes valores são transmitidos e somados em suas respectivas posições no *buffer* de suporte global da máquina concentradora. Esta, então, transmite a todas as demais máquinas esses valores de suportes globais. Assim, cada posição do *buffer* de suportes globais que possuir valor maior ou igual ao suporte mínimo pré-estabelecido é considerada um item freqüente, formando, desta maneira, o conjunto F_1 (linha 2).

A partir do conjunto de itens freqüentes F_1 a função *GeraRegras1*( ) produz o conjunto de item-regras freqüentes de tamanho 1(linha 3). Então, enquanto houver a formação de um novo conjunto de itens-regras freqüentes, a função *JuntaItemRegras*( ) produz novos itens-regras candidatos, que são armazenados na estrutura de lista de classes. Para cada novo item-regra armazenado, o contador *numCandidatos* é incrementado e para cada novo nó da lista de classes gerado, o contador *numClasses* também é incrementado(linhas 8-14).

Para a contagem dos suportes locais dos itens-regras candidatos é alocado um *buffer* de tamanho igual a soma dos valores de *numCandidatos* e *numClasses*. Assim, a função *ContaSupLocal*( ) armazena na primeira posição do *buffer* o suporte do prefixo do primeiro nó da lista de classes e, nas posições *subseqüentes,* os suportes dos elementos de corpo deste nó. Repete-se o processo para os demais nós da lista de classes até que todos os suportes tenham sido armazenados neste *buffer*. Em seguida, é alocado um *buffer* de contagem global de tamanho igual ao alocado para a contagem local (linha 16). Através de uma primitiva de redução de soma, todos os suportes locais são somados no *buffer* de suporte global da máquina concentradora (linha 17). Então, estes suportes são transmitidos para as demais máquinas através de uma primitiva de difusão (linha 18).

Algoritmo Paralelo para Geração de Regras de Classificação

entrada: arquivo de Base de Dados : **Base.txt**
quantidade de máquinas : **numProc**
suporte mínimo : **X %**
confiança mínima : **Y%**

```
1    ParticionaBase ( )
2    F1 = FreqF1( )
3    R1=GeraRegras1 (F1)
4    K = 1
5    enquanto (R K-1 ≠ ∅ ) faça
6       K ++
7       numCandidatos = 0 ; numClasses=0
8       para todas as regras r∈ RK-1 faça
9          rc a= JuntaItemRegra ( r i, r j )
10         se Poda (c a ) é falso então
11            numCandidatos + +
12            ArmazenaItemRegra ( rc a)
13            se gerou novo nóClasse então numClasses ++
14      fim para
15      tamBuffer=numCandidato + numClasses
16      ContaSupLocal (CK )
17      Reduce ( SupLocal[0],SupGlobal[0], tamBuffer,soma ( ), idProc )
18      Bcast (SupGlobal[0]], tamBuffer, idProc )
19      RK= GeraRegrasK ( CK )
20   fim enquanto
21   fim
```

saída : **Regras** = $\mathbf{Y}_1^k$ R_K

Figura 4.41 : Algoritmo Paralelo para Geração de Regras de Classificação

Na função *GeraRegrasK*( ), o *buffer* global contendo a contagem dos suportes é utilizado para eliminação de todos os itens-regras que não satisfaçam aos valores de suporte mínimo ou de confiança mínima pré-estabelecidos e é formado um novo conjunto de itens-regras freqüentes iguais em todas as máquinas que executam o

algoritmo. O processo se repete até que nenhum novo conjunto de itens-regras seja gerado. Desta forma, é obtido, em todas as máquinas, um conjunto completo de regras de classificação para uma determinada base de dados de treinamento.

O processo descrito é a estratégia adotada para a paralelização do algoritmo de Geração de Regras de Classificação I, que realiza a mineração apenas de itens-regras freqüentes. A paralelização do algoritmo de Geração de Regras de Classificação II, apresentado na seção 4.5.2.1, é mais simples e é realizada de maneira semelhante à maneira descrita para a paralelização do algoritmo de mineração de itens freqüentes na seção 4.4.1. Entretanto, no algoritmo paralelo do Gerador de Regras de Classificação II, não são necessários a contagem e a transmissão dos suportes locais dos prefixos dos nós candidatos da lista de classes, pois estes valores podem ser carregados diretamente das listas de classes de *itemsets* freqüentes da iteração anterior, presente em todas as máquinas.

4.6.2. Paralelização da Etapa de Construção do Classificador

No final da etapa de geração de regras de classificação, todas as máquinas que executam o algoritmo possuem uma partição da base de dados e um conjunto completo de regras de classificação. Mas, para a construção do classificador, é necessária a seleção de um conjunto mínimo ordenado de regras. Para isto, todas as máquinas ordenam simultaneamente seus conjuntos de regras e estes são testados nas partições da base de dados das respectivas máquinas.

Durante os testes realizados com as regras sobre uma partição dos dados, para cada regra é associada uma marca, uma contagem de erro e uma contagem para as classes. É, então, alocado um *buffer* local para o armazenamento dessas informações. Após todas as máquinas realizarem seus testes nas suas partições dos dados, essas informações são transmitidas através de uma primitiva de redução de soma para um *buffer* global em uma máquina concentradora. E assim, de posse das informações integralizadas, a máquina concentradora, extrai o conjunto de regras mínimo que constitui o classificador. O algoritmo paralelo de construção do classificador é mostrado na figura 4.42.

Algoritmo Paralelo para Construção do Classificador

entrada: conjunto de regras: **Regras**

```
1   R_ordenado =Ordena(Regras)
2   passo=numClasse + 2
3   tamBuffer = passo * NumRegras
4   aloca bufferLocal [tamBuffer] ; bufferGlobal[tamBuffer]
5   indice =0
6   para cada regra r ∈ R_ordenado em seqüência faça
7       p= índice * passo
8       para cada caso d ∈ Partição_D faça
9           se r.prefixo ⊂ d então
10              se r.classe ∉ d então
11                  ErroBufferLocal( r, p) ++
12                  ClasseBuferrLocal(d.classe, p) ++
13              senão marque r
14          senão ClasseBuferrLocal(d.classe, p) ++
15          se r é marcado então
16              retira d da base de treinamento Partição_D
17              classeDefault=Max(p, bufferLocal[ ])
18              ErroTotalBufferLocal(p, classeDefault)
19              MarcaBufferLocal (r , p)
20          se Partição_D = ∅ então fim para 1
21      fim para 2
22  indice ++
23  fim para1
24  Reduce ( BufferLocal[0],BufferGlobal[0], tamBuffer,soma ( ), idProc )
25  regraCorte = ∞
26  para cada regra r ∈ R_ordenado em seqüência faça
27      se BufferGlobal . marca[r] ≠ 0 então
28          se BufferGlobal.erro[r] < regraCorte então
29              regraCorte = BufferGlobal.erro[r]
30              ultimaregra= r
31      senão descarta regra r
32  fim para 3
33  descarta todas as regras r ∈ R_ordenado após ultimaRegra
34  Classificador C = R_ordenado + ultimaRegra .classeDefault
35  fim
```

Figura 4.42 : Algoritmo Paralelo para Construção do Classificador

A primeira fase do algoritmo paralelo de construção do classificador é a ordenação das regras (linha 1). Após a ordenação das regras, são alocados *buffers* para o armazenamento de informações (linhas 2-4). Estes *buffers* contêm todas as informações pertinentes às regras, ou seja, para cada regra há um campo marca que informa a condição da regra, um campo de contagem de erros associado à regra e um campo para cada classe existente na base de dados. A figura 4.43 ilustra uma esquematização destes *buffers*.

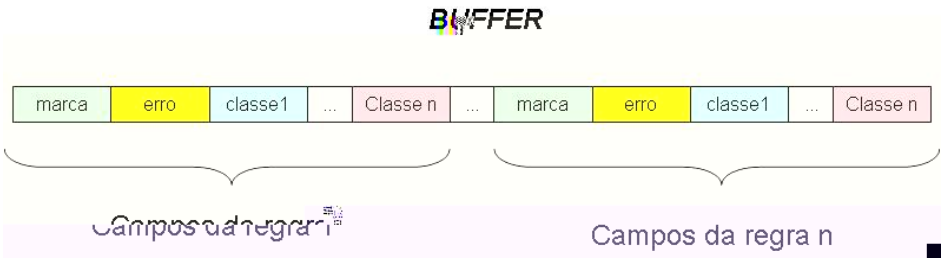


Figura 4.43 : Esquematização do *buffer*

Então, cada regra "r" do conjunto ordenado de regras é testada em todas as instâncias "d" da partição da base de dados (linhas 5-21). Primeiramente, verifica-se se o antecedente da regra "r" está contido em "d". Se for o caso e a regra "r" classificar corretamente "d", marca-se a regra. Mas se "r" não classifica a instância "d", a função *ErroBufferLocal*( ) incrementa o contador de erros referente a regra no *buffer* local e a função *ClasseBufferLocal*( ) incrementa, ainda, a posição do *buffer* local referente a classe da instância "d". Mas, no caso do antecedente de "r" não estar contido em "d", a função *ClasseBufferLocal*( ) incrementa a posição do *buffer* local referente a classe da instância "d" (linhas 10-12). Se a regra foi marcada, retira-se "d" da partição da base de treinamento (linha 16). Em seguida, a função *Max*( ) acha o valor da classe padrão associado a "r" no *buffer* local. Este valor representa a classe com a maior contagem entre os campos de classes associados a uma regra no *buffer* local (linha 12). A função *ErroTotalBufferLocal*( ) calcula o erro total do classificador, isto é, a soma das contagens de todas as posições dos campos de contagem de classe exceto a referente à classe padrão. Este valor é somado aos erros cometidos pela regra e, então, o total de erros é acumulado no campo de contagem de erro do *buffer* local. O processo se repete

até que não haja mais instâncias na partição da base de dados ou termine as regras a serem testadas (linha 15 -21).

Terminada a fase de avaliação das regras, todas as informações colhidas nos *buffers* locais de cada máquina são transmitidas para o *buffer* global de uma máquina concentradora a partir de uma primitiva de redução de soma. No *buffer* global são totalizados os erros de cada regra, bem como as contagens das classes presentes nas partições durante a avaliação das regras. As regras que não satisfizeram nenhuma instância das partições têm os campos "marca" associados a elas com valor igual a zero; as demais apresentam algum valor nestes campos diferentes de zero. Então, o *buffer* global é percorrido e todas as regras que possuem o campo marca zerado são eliminadas. Além do mais, durante este percurso, é buscada a primeira regra da seqüência que apresenta o menor valor no campo erro. Esta regra é considerada a regra de corte, isto é são descartadas todas as regras que a sucedem. Desta forma, o classificador é composto das regras que não foram eliminadas e da classe padrão associada à regra de corte. Esta classe padrão é a classe com a maior contagem dentre os campos de contagem de classes do *buffer* global associados à regra de corte (linhas 26-36).

Capítulo V

O Estudo de Caso

Neste capítulo é apresentado o estudo de caso, isto é, a aplicação das implementações desenvolvidas nesta tese em bases de dados de um sistema de contramedidas antimíssil. Primeiramente, é apresentado o sistema de contramedidas, seus dados e parâmetros. Em seguida, é apresentado o ambiente dos testes, a descrição dos experimentos e os resultados.

5.1 O PROBLEMA

Nesta seção é apresentado um sumário do problema, isto é, as vantagens e dificuldades da utilização de tarefas de mineração de dados com a finalidade específica de aprimoramento de um sistema de contramedida antimíssil para navios de superfície.

A maioria das tarefas de mineração de dados, em especial a de mineração de regras de associação e a de classificação, sofre fortes restrições em suas implementações devido a diversos fatores, sobretudo, o caráter iterativo dos algoritmos de mineração, aliado a grande quantidade de dados que limitam as plataformas de *hardware* onde estes podem ser executados.

Apesar da generalidade da maioria dos algoritmos desenvolvidos para as tarefas de mineração de regras de associação e de classificação, é evidente que as

implementações mais robustas e eficientes são aquelas onde se leva em conta, além do *hardware* disponível, o conhecimento dos dados e sua natureza [14]. Portanto, acredita-se que a maior motivação para implementações de ferramentas de mineração de dados para aplicações específicas resida no fato de se obter um maior domínio do sistema, facilitando, assim, futuras adaptações e acréscimos de funcionalidades, na medida em que estas se mostrarem necessárias.

Sistemas de mineração de dados são utilizados em aplicações militares como em [16]. Essa técnica pode ser especialmente importante na área de desenvolvimento de contramedidas, onde o conhecimento dos parâmetros de uma ameaça, para a qual será desenvolvida a contramedida, às vezes, é bastante limitado. Portanto, o uso de técnicas e sistemas de mineração de dados é fundamental na busca de informações que possam contribuir para a eficácia do desenvolvimento.

Dentre as contramedidas utilizadas por navios para se defender de ataques de mísseis está o emprego de foguetes de *Chaff* [17], isto é, foguetes com cargas de filamentos metalizados, para a formação de nuvens com o propósito de confundir ou seduzir o radar do míssil, como ilustrado na figura 5.1

Figura 5.1 : Desvio de rota do míssil por ação de nuvem de Chaff

As nuvens de *Chaff* devem ser formadas em posições chaves de modo a conseguir desviar, eficazmente, o míssil do seu alvo. Para isto, é necessário um sistema de lançamento de foguetes de *Chaff* que determina, através da análise de diversos parâmetros, a posição, o momento de abertura da carga e a quantidade de foguetes capazes de desviar a trajetória do míssil. A maior dificuldade, no desenvolvimento desse tipo de sistema de contramedida é a falta de informações precisas da ameaça, haja vista que estas, normalmente, são informações secretas. Assim, a mineração de dados de cenários táticos envolvendo diversas condições e comportamentos de ameaças, além de parâmetros meteorológicos e de navios, pode levar a descoberta de importantes informações que contribuam para o aprimoramento desses sistemas de contramedida.

A importância da análise computacional dos dados, aliada às restrições das aplicações militares, motiva o uso de ferramentas de mineração de dados, de modo a permitir a busca de novos conhecimentos para o aprimoramento de sistemas antimíssil. É evidente, contudo, que a aplicação de tarefas de mineração de dados pode ser estendida a outros sistemas militares, pois, onde haja dados coletados, certamente, existem informações úteis de forma implícita.

5.2 O SISTEMA DE CONTRAMEDIDAS

Esta seção apresenta o sistema de contramedida antimíssil, sua missão, os tipos de táticas empregadas para defesa de navios, suas principais características e configurações.

5.2.1. Missão do Sistema

A missão de um sistema de contramedida antimíssil é a defesa de um navio de superfície contra ataques de mísseis guiados por meio de radar. Utilizando para isso, lançamentos coordenados de foguetes de *Chaff*; podendo, entretanto, através da disponibilidade de outros tipos de munição, ser também utilizado na defesa de um navio de superfície contra outros tipos de ameaças.

Sistemas de contramedida para defesa de navios com a utilização de foguetes de *Chaff* são sistemas complexos, que apresentam vários componentes e diferentes configurações. Normalmente, são sistemas com processamento distribuído, onde em um dos módulos de processamento reside a parte tática, responsável pelos diferentes modos de emprego das munições. Outro módulo importante desses sistemas é o módulo lançador, responsável pelo lançamento das munições. A configuração de um sistema de contramedida deste tipo varia de acordo com a quantidade de módulos lançadores presentes no sistema. Além do processador tático e dos lançadores, existem diversos outros módulos para controle e execução das táticas de emprego da munição. A figura 5.2 ilustra um sistema de contramedidas antimíssil com foguetes de *Chaff*.

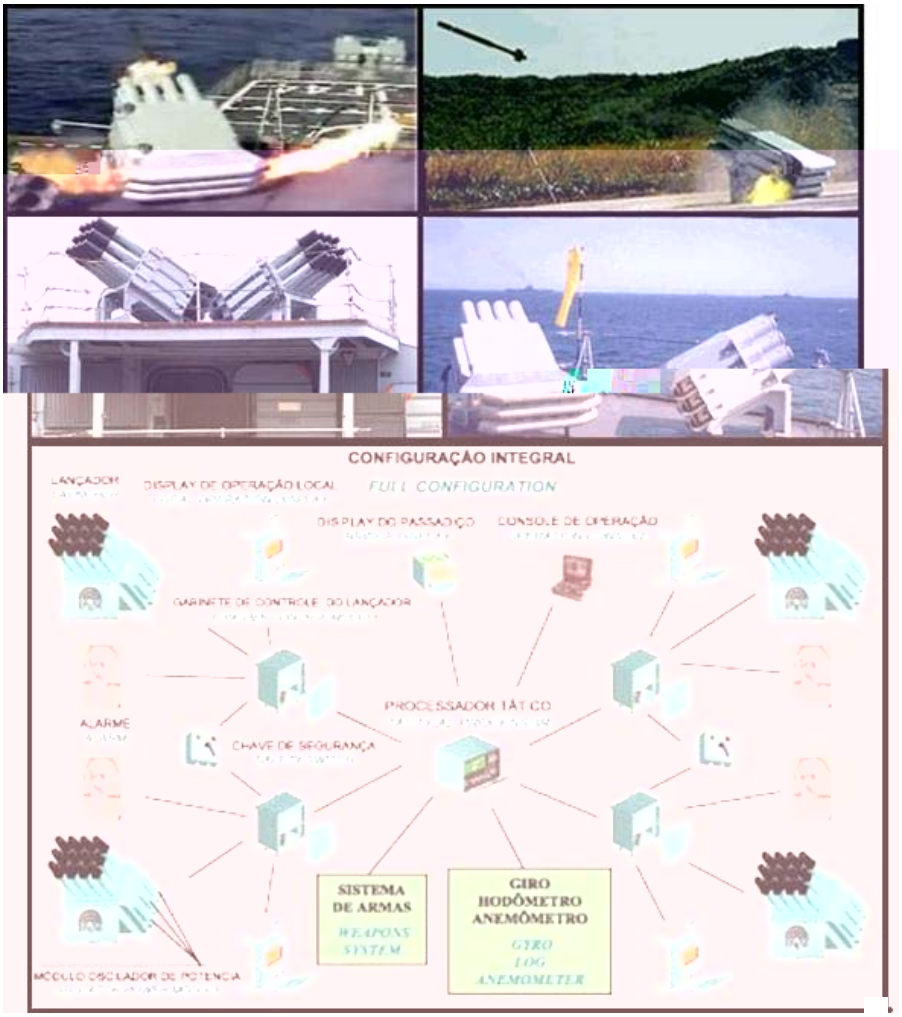


Figura 5.2 : Um sistema de contramedida antimíssil com foguetes de *Chaff*

5.2.2. Emprego Tático

Para defender um navio de superfície contra um míssil antinavio guiado por radar, o sistema de contramedida, dependendo do software tático ou tabela de tiro instalada, pode gerar soluções de tiro. Estas traduzem-se pela definição da manobra do navio, lançadores a serem designados, quantidade de foguetes a serem lançados e temporização do dispositivo de liberação da carga de *Chaff*, de tal forma que permita o engajamento do navio nos modos táticos de sedução, distração ou confusão.

5.2.2.1. Sedução

É o modo tático caracterizado por atuar basicamente a partir do momento em que o míssil adquire o alvo, ou seja, na fase de acompanhamento, também denominada fase de guiagem final. Pode ser, ainda, empregado contra radares de direção de tiro travados no alvo para quebrar o acompanhamento. Esses sistemas de contramedida normalmente permitem o uso de dois tipos de sedução: a sedução por deslocamento do centróide; e a sedução por deslocamento da janela de acompanhamento (*DUMP*).

Na Sedução por deslocamento de centróide é empregado o lançamento de foguetes de *Chaff* a curta distância e, normalmente, exige a manobra do navio para deslocar o centróide até que a nuvem de *Chaff*, que oferece um alvo radar mais atrativo e seduza definitivamente o radar de guiagem do míssil. A figura 5.3 ilustra o radar do míssil sendo seduzido pelo deslocamento do centróide causado pela formação de uma nuvem de *Chaff*.

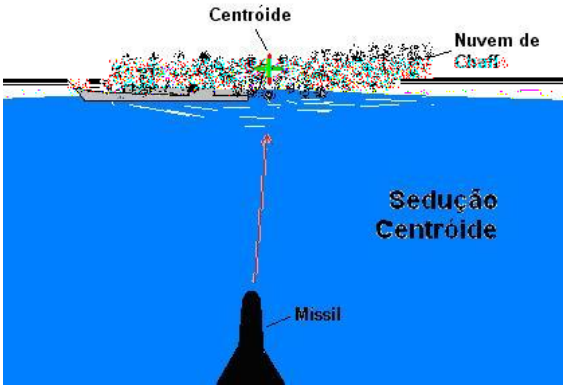


Figura 5.3 : O modo tático de sedução por deslocamento de centróide

Na Sedução por Deslocamento da Janela de Acompanhamento é realizada a ação coordenada de um equipamento bloqueador eletrônico de radar e de foguetes de *Chaff*. Inicialmente, o bloqueador aplica um truque destinado a deslocar a janela de deslocamento do radar de guiagem ou de direção de tiro. Por sua vez, o sistema de contramedida deverá lançar, com o menor tempo de reação possível, foguetes de *Chaff* nas proximidades da região para onde for deslocada a janela de travamento do radar de guiagem ou direção de tiro, como ilustrado na figura 5.4. Para que isso seja alcançado, é necessária a judiciosa escolha e programação do foguete a ser lançado, além de recomendação de manobras que viabilizem o modo de emprego e maximizem a distância do ponto de maior aproximação do míssil em relação ao navio.

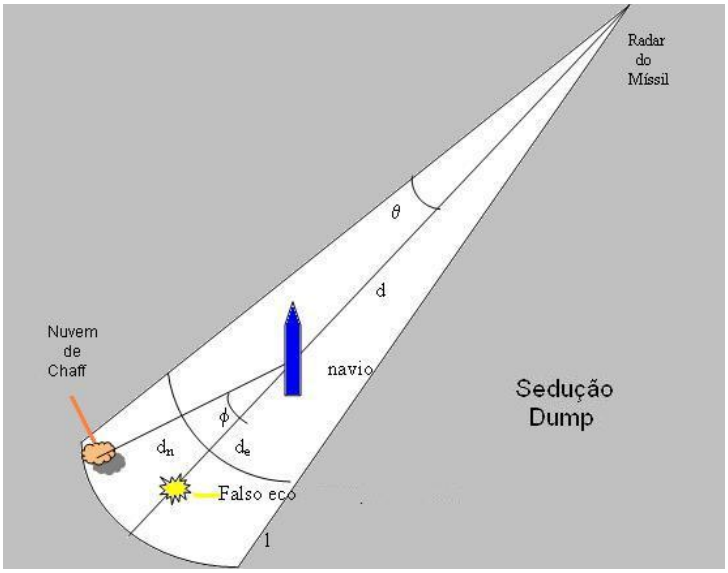


Figura 5.4 : O modo tático de sedução por deslocamento de janela de acompanhamento

5.2.2.2. Distração

Esse modo tático é eficaz apenas durante a fase de busca do radar de guiagem do míssil. Consiste na semeadura de foguetes de *Chaff*, de acordo com padrões otimizados para cada tipo de ameaça e previstos no sistema, que oferecerão ao radar de guiagem um grande número de alvos falsos durante a fase de busca. Assim, haverá uma probabilidade elevada do radar detectar um alvo falso, adquiri-lo e iniciar a sua fase

final de guiagem contra o mesmo. A figura 5.5 ilustra o radar de busca de um míssil adquirindo um alvo falso formado por uma nuvem de *Chaff*, quando do emprego da tática de distração.

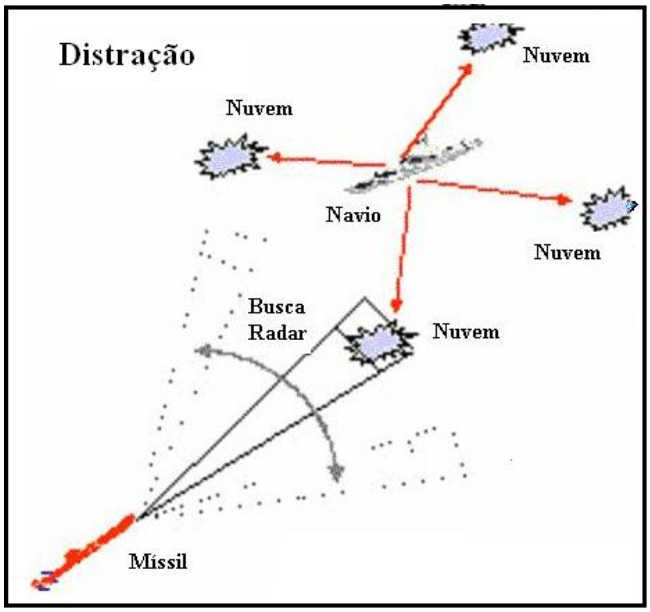


Figura 5.5 : O modo tático da Distração

5.2.2.3. Confusão

Este modo tático é caracterizado por atuar antes que o míssil seja lançado. Tem por objetivo gerar alvos falsos, visando confundir o radar inimigo dificultando a designação de alvos. A figura 5.6 ilustra a aplicação da tática de confusão e como navio se confunde com as nuvens de *Chaff* numa tela de radar.

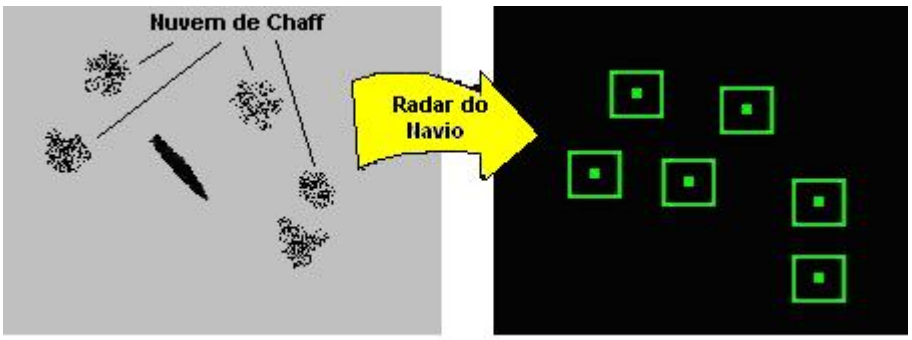


Figura 5.6 : **O modo tático de Confusão**

5.2.3. Características do Sistema

Para que seja possível a realização, pelo sistema de contramedida, de todos ou de alguns dos modos táticos, descritos na subseção 5.2.2, é necessário que o sistema e a munição apresentem requisitos que propiciem a realização da tática. Esses sistemas apresentam algumas características intrínsecas, que determinam e limitam suas aplicações.

Entre as características de um sistema de lançamento de foguetes de *Chaff* para defesa de navios podemos citar como as mais importantes: a quantidade de lançadores do sistema; a quantidade de foguetes por lançador; a inclinação dos lançadores, no caso de lançadores fixos, ou no caso de lançadores móveis, os valores limites de conteira e elevação; a quantidade de lançamentos simultâneos; o tempo de reação, entre a detecção da ameaça e o lançamento do foguete; e o intervalo de tempo entre lançamentos de um mesmo lançador. Ainda aparecem como limitações do sistema de lançamento de foguetes as características de sua munição, pois a quantidade da carga de Chaff; o tempo de abertura da nuvem; o alcance do foguete entre outros, influenciam na tática adotada.

5.3 O EMPREGO DA MINERAÇÃO DE DADOS

Esta seção trata das contribuições que a aplicação da mineração de dados traz para o aprimoramento do sistema de contramedida. Inicialmente, é sumarizada a função dos dados que estão sujeitos as técnicas de mineração de regras de associação e de classificação. É apresentado, ainda, o simulador de ambiente de *Chaff*, responsável pela geração da maioria dos dados do sistema. Em seguida, é discutido de que maneira as regras de associação e de classificação contribuem para o aprimoramento do sistema de contramedida.

5.3.1. Os Dados do Sistema

Os sistemas de contramedida possuem um *software* tático responsável pela execução dos modos táticos descritos na seção 5.2. Este *software* recebe dos demais

sistemas e sensores do navio os parâmetros necessários para criação de um cenário, ou instância de ameaça. A partir da interpretação deste cenário, do modo tático designado, da disponibilidade de municiamento e dos *status* das partes integrantes do sistema é calculada uma solução que consiste, basicamente, da direção dos lançamentos, através da escolha dos lançadores, da quantidade de foguetes de *Chaff* a ser lançado e da temporização das espoletas desses foguetes.

Muitas vezes devido aos requisitos de tempo de reação, ou de impossibilidade de lançamento devido à ausência de munição nos tubos designados, ou ainda, de falhas de ordem diversas do sistema, não é possível o cálculo ou a execução de uma solução. Para estes casos, esses sistemas são dotados de tabelas (*look-up tables*) compostas de diversos cenários táticos e soluções padrões que representam uma alternativa de defesa. Entretanto, tanto o *software* tático quanto as tabelas de soluções são baseados em dados oriundos, na sua grande maioria, de simulações.

5.3.1.1. A Simulação dos Dados

O simulador que gera os dados utilizados pelo sistema de contramedida, simula o ataque de um míssil a um navio de superfície. O míssil atacante rastreia o alvo através de seu equipamento radar, enquanto o navio busca se defender através da utilização de foguetes de *Chaff*. Para tanto, são considerados diversos fatores, como ambiente meteorológico; geometria míssil-alvo; características do míssil atacante (físicas; aerodinâmicas; de sensoriamento e do sistema de controle); características do alvo (físicas; eletromagnéticas; hidrodinâmicas; do sistema de propulsão e manobra); características do sistema de contramedida (lançadores: tipo e posicionamento; e sistema de controle de lançamento); características do foguete (físicas; aerodinâmicas e de propulsão) e características da carga útil dos foguetes (físicas; eletromagnéticas e aerodinâmicas).

A modelagem das entidades, que compõe o ambiente de simulação, é o grande desafio, pois, para se garantir bons resultados, é necessário que as entidades envolvidas, tais como navio, míssil, nuvem de *Chaff* e, etc, estejam bem representadas. Para isso são necessários alguns dados que não se encontram disponíveis, como por exemplo: a seção

reta radar do navio para todas as marcações da ameaça, as características exatas do filtro do radar de busca da ameaça, entre outras.

A simulação é um método rápido e barato de testar o comportamento de um sistema. Porém sofre de uma limitação importante: a validade dos modelos empregados. É extremamente difícil modelar precisamente um sistema qualquer e a dificuldade cresce exponencialmente com a complexidade do sistema. Por isso, é necessário um compromisso entre a precisão dos modelos e a exeqüibilidade da simulação.

5.3.2. Contribuições da Mineração dos Dados

De acordo com a descrição da simulação e a quantidade de variáveis envolvidas o

simulaç

mostrando, então, que a modelagem pode ser revista sobre algum enfoque especial que a aproxime mais da realidade. Isso melhoraria muito o desempenho do sistema de contramedida, haja vista que esses dados simulados dão origem ao *software* tático e as suas tabelas (*look-up tables*)de soluções.

Por outro lado, no caso dos modelos simulados estarem bem representados, as regras de associação podem revelar diversas características ocultas, como, por exemplo, nos padrões de busca do míssil, o que contribuiria para alterações nas táticas e doutrinas de uso de foguetes de *Chaff*.

Além do mais, baseado no conhecimento dos dados, algumas intervenções por parte do usuário podem ser aplicadas. Por exemplo, o uso de algum tipo de hierarquia entre os dados ou a separação destes em grupos de características semelhantes. Isto possibilitaria diversas outras descobertas úteis. Essa metodologia pode ser utilizada para criar parâmetros de modo a entender o comportamento de ameaças; identificar afinidades entre as soluções e cenários táticos; prever soluções; analisar comportamentos para detectarem falhas e etc. Isto demonstra que a utilização de regras de associação nessas bases de dados pode contribuir, realmente, para o aprimoramento dos sistemas de contramedida, já que as possibilidades de aplicações são inúmeras.

5.3.2.2. Minerando Regras de Classificação

A classificação, diferentemente das associações, é um método supervisionado, isto é, aprende-se com exemplos, e é usado para a predição de casos. No caso em estudo, aplicou-se a classificação de cenários táticos em classes de setores de defesa. A região em torno do navio pode ser dividida em setores de acordo com o lançador utilizado na solução de um cenário. Assim, no caso de navios que possuem quatro unidades lançadoras, existem quatro classes, se forem desprezadas as soluções combinadas, isto é aquelas onde é necessária a utilização de mais de uma unidade lançadora para a defesa do navio. O resultado da análise deste processo pode ser usado, por exemplo, no desenvolvimento de uma política de municiamento durante missões de navios em que haja escassez de munição.

Existem instâncias do problema, isto é, cenários, em que não são possíveis encontrar uma solução. Portanto, a separação dessas instâncias em duas classes, as com

de regras de associação não foi usado nenhum critério de escolha dos itens que as compõem. Como esta tarefa necessita de etapas posteriores para a geração de regras úteis, o critério usado para geração das bases foi o volume de dados, de modo a possibilitar a análise da escalabilidade da implementação.

Bases de Dados para Classificação				
bases	**Nº Atributos**	**Nº Classes**	**Nº Instâncias**	**Total de itens**
CV-Classe1	20	2	2.558	103
CV-Classe2	20	2	20.463	103
SLDM-Classe1	35	3	61.387	202
SLDM-Classe2	35	4	83.204	202
SLDM-Classe3	35	4	96.782	202

Tabela 5.1 - Características das bases de dados para Regras de Classificação

Bases de Dados para Regras de Associação		
bases	**Nº Instâncias**	**Total de itens**
Chaff-SLDM2	94.494	202
Chaff-SLDM3	238.469	202
Chaff-SLDM4	487.839	202

Tabela 5.2 - Características das bases de dados para Regras de Associação

5.4.1. Os Testes de Mineração de Regras de Classificação

Os testes da tarefa de mineração de regras de classificação foram realizados nas bases de dados da tabela 5.1. Nestes testes, foram usados valores de suporte mínimo e de confiança mínima igual a 1% e 50%, respectivamente, e limitou-se o total de regras geradas para a construção do classificador em 100.000. A tabela 5.3 e os gráficos apresentados nas figuras 5.7 e 5.8 apresentam os resultados do experimento.

base	Nº Atributos	Nº Classes	Nº Instâncias	Total de itens	Nº de Regras	Taxa de erros (%)
CV-Classe1	20	2	2558	103	497	5,10
CV-Classe2	20	2	20463	103	4.293	8,32
SLDM-Classe1	35	3	61387	202	7.618	18,80
SLDM-Classe2	35	4	83204	202	13.315	21,56
SLDM-Classe3	35	4	96782	202	17.751	24,30

Tabela 5.3 - Número de Regras de Classificação e Taxa de Erro

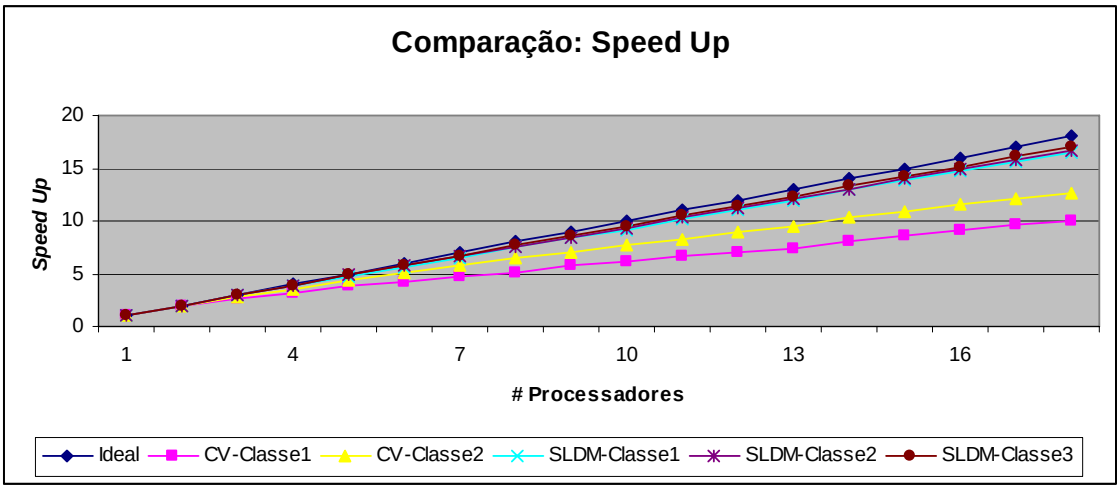


Figura 5.7 : Comparação da relação de *Seed Up* na Classificação

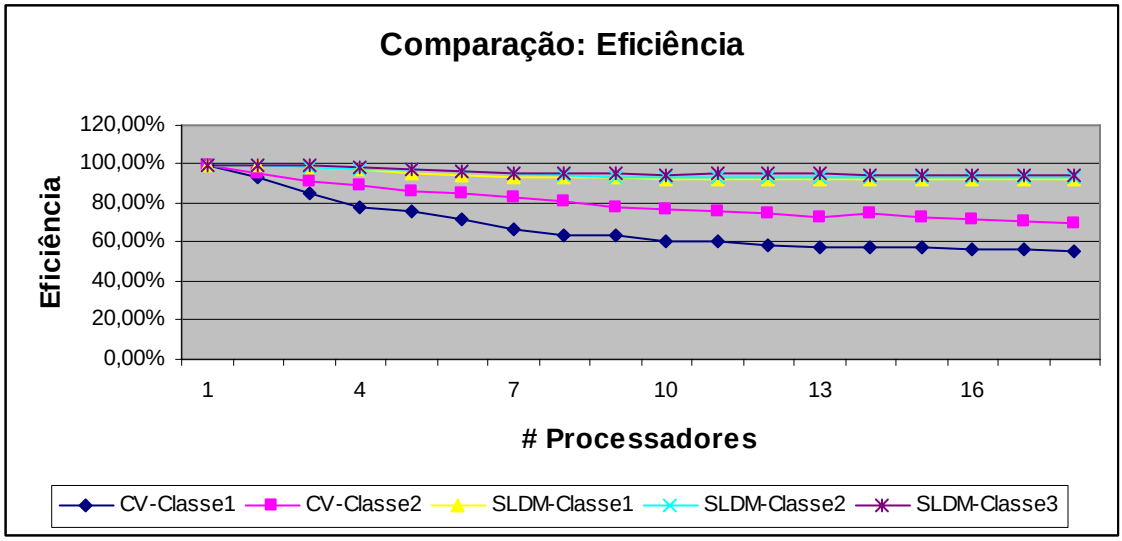


Figura 5.8 : Comparação da Eficiência na Classificação

Os resultados dos testes para a tarefa de mineração de regras de classificação mostram que a metodologia funcionou bem, pois foi possível a extração das regras das bases experimentais e a construção de classificadores, onde as taxas de erros, obtidas com o processo da validação cruzada (*10-fold Cross Validation*), são bem razoáveis. Os resultados do estudo do *speed up* mostram uma redução gradual na eficiência à medida que o tamanho das partições de dados diminui na memória das máquinas. Entretanto para as bases de dados maiores o *speed up* se mostrou razoável, garantindo a escalabilidade.

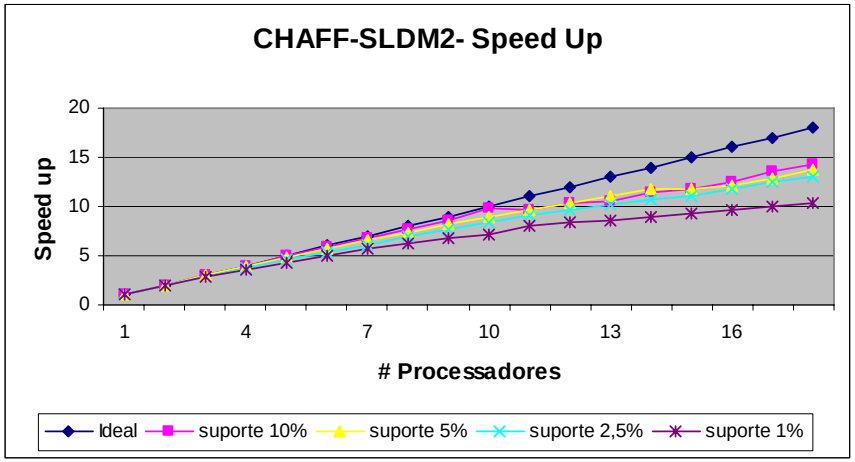
5.4.2. Os Testes de Mineração de Regras de Associação

Os testes da mineração de regras de associação foram realizados nas bases de dados da tabela 5.2. Esses testes foram executados variando-se os valores de suporte mínimo e de confiança mínima. Os resultados dos testes são apresentados nas tabelas 5.4 a 5.6 e nas figuras 5.9 a 5.14. Nessas tabelas, são mostrados os valores de *speed up*, de eficiência e de tempo de execução para os seguintes valores de suporte mínimo: 10%; 5%; 2,5% e 1%. As tabelas, também, apresentam o tamanho do maior *itemset* freqüente encontrado para cada um dos quatro valores de suporte mínimo adotados. É apresentado, ainda, a quantidade de regras de associação gerada para cada um dos quatro valores de suporte mínimo, quando se varia os valores da confiança mínima para 100%, 90%, 80% e 70%.

Os resultados dos testes para a tarefa de mineração de regras de associação mostram que a metodologia também funcionou bem, pois foi possível a extração das regras de associação das bases experimentais. As três bases de dados utilizadas apresentaram características semelhantes, isto é, poucas associações para os maiores valores de suporte mínimo, e tornando-se mais densas a medida em que os valores de suporte mínimo decresce. O *speed up* e a eficiência também apresentaram uma diminuição gradual na medida em que se diminui a partição dos dados nas memórias das máquinas. Esta diminuição dos valores do *speed up* e da eficiência foi mais acentuada para os testes onde eram menores os valores de suporte mínimo, devido ao significante aumento de *itemsets* freqüentes e, consequentemente, de consumo de memória. Porém, os valores de *speed up* obtidos são satisfatórios, o que garante a escalabilidade.

Base	CHAFF-SLDM2											
Suporte Mínimo	**10%**			**5%**			**2.5%**			**1%**		
# Proc	**Tempo(seg)**	***SpeedUp***	**Eficiência**	**Tempo(seg)**	***SpeedUp***	**Eficiência**	**Tempo(seg)**	***SpeedUp***	**Eficiência**	**Tempo(seg)**	***SpeedUp***	**Eficiência**
1	114,18	1,00	100,00%	401,71	1,00	100,00%	1414,46	1,00	100,00%	5916,82	1,00	100,00%
2	57,09	2,00	100,00%	202,47	1,98	99,20%	722,47	1,96	97,89%	3070,16	1,93	96,36%
3	38,19	2,99	99,67%	135,67	2,96	98,70%	493,39	2,87	95,56%	2107,58	2,81	93,58%
4	28,69	3,98	99,50%	102,94	3,90	97,56%	377,23	3,75	93,74%	1652,37	3,58	89,52%
5	23,46	4,97	99,40%	84,48	4,76	95,20%	307,27	4,60	92,07%	1376,43	4,30	85,97%
6	19,52	5,85	97,50%	70,61	5,69	94,82%	262,26	5,39	89,89%	1185,12	4,99	83,21%
7	16,74	6,82	97,43%	61,52	6,53	93,28%	230,72	6,13	87,58%	1040,06	5,69	81,27%
8	14,87	7,68	96,00%	55,46	7,24	90,54%	204,95	6,90	86,27%	944,09	6,27	78,34%
9	13,34	8,56	95,11%	48,94	8,21	91,20%	184,05	7,69	85,39%	880,44	6,72	74,67%
10	11,60	9,84	94,40%	44,98	8,93	89,30%	167,04	8,47	84,68%	824,07	7,18	71,80%
11	11,94	9,56	86,91%	41,70	9,63	87,58%	155,34	9,11	82,78%	743,97	7,95	72,30%
12	11,06	10,32	86,00%	38,73	10,37	86,43%	145,56	9,72	80,98%	701,58	8,43	70,28%
13	10,81	10,56	81,23%	36,07	11,14	85,67%	139,26	10,16	78,13%	690,76	8,57	65,89%
14	10,06	11,35	81,07%	34,20	11,74	83,89%	132,12	10,71	76,47%	661,70	8,94	63,87%
15	9,64	11,85	79,00%	33,96	11,83	78,86%	126,76	11,16	74,39%	637,35	9,28	61,89%
16	9,09	12,56	78,50%	32,86	12,22	76,40%	119,64	11,82	73,89%	613,27	9,65	60,30%
17	8,45	13,51	79,47%	31,05	12,94	76,10%	113,96	12,41	73,01%	590,91	10,01	58,90%
18	8,04	14,20	78,89%	29,38	13,67	75,97%	108,12	13,08	72,68%	567,43	10,43	57,93%
***itemset* Máximo**	12			13			14			15		
Confiança Mínima	**Nº de Regras de Associação**			**Nº de Regras de Associação**			**Nº de Regras de Associação**			**Nº de Regras de Associação**		
100%	36.320			164.191			605.084			3.002.057		
90%	49.481			203.222			691.881			3.211.342		
80%	57.335			231.560			784.011			3.561.038		
70%	67.843			272.838			905.001			3.977.844		

Tabela 5.4 - Resultados da base Chaff-SLDM2



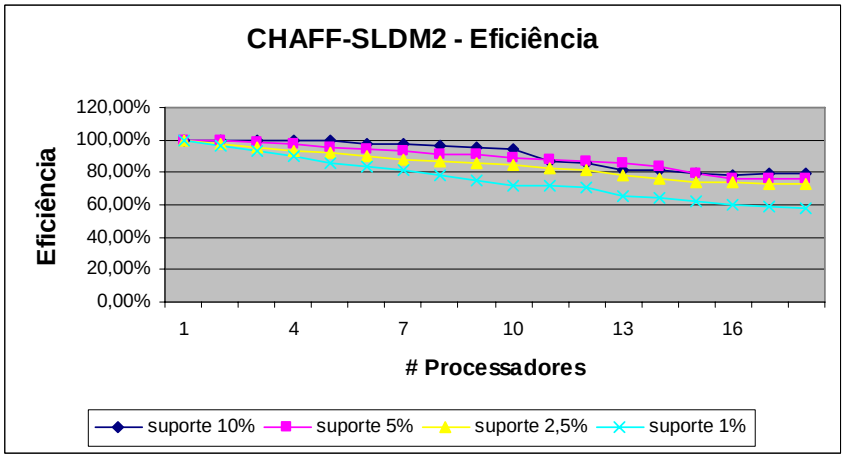


Figura 5.9 : *Speed Up* e Eficiência da base Chaff-SLDM2

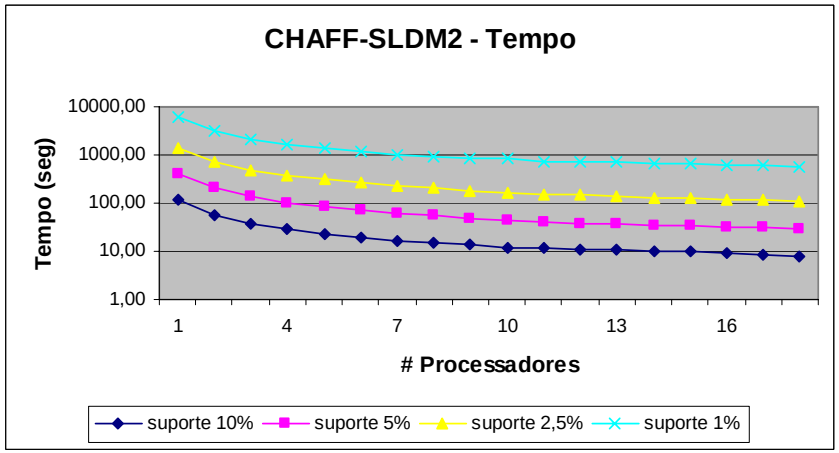
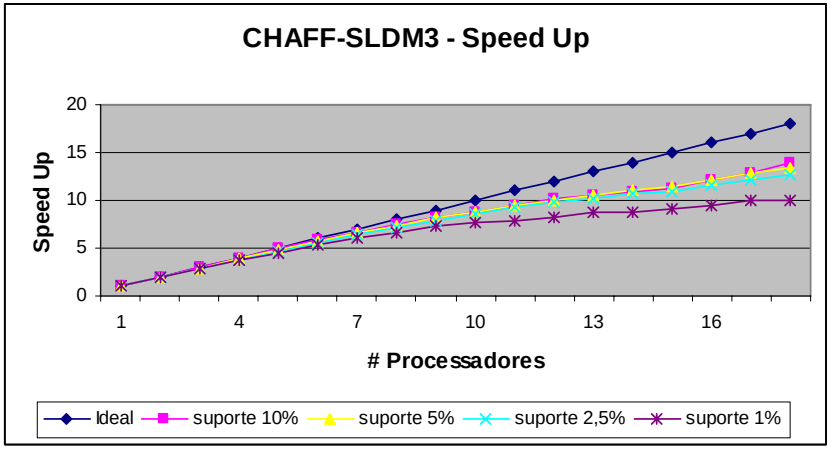


Figura 5.10 : Tempos de execução da base Chaff-SLDM2

Base	CHAFF-SLDM3											
Suporte Mínimo	10%			5%			2.5%			1%		
# Proc	Tempo(seg)	*SpeedUp*	Eficiência	Tempo(seg)	*SpeedUp*	Eficiência	Tempo(seg)	*SpeedUp*	Eficiência	Tempo(seg)	SpeedUp	Eficiência
1	95,36	1,00	100,00%	382,01	1,00	100,00%	1412,20	1,00	100,00%	7821,36	1,00	100,00%
2	47,73	2,00	99,89%	192,74	1,98	99,10%	714,1678	1,98	98,87%	3983,9868	1,96	98,16%
3	31,95	2,99	99,50%	129,78	2,94	98,12%	489,47884	2,89	96,17%	2700,5603	2,90	96,54%
4	24,11	3,95	98,87%	98,34	3,88	97,11%	375,02533	3,77	94,14%	2121,4503	3,69	92,17%
5	19,38	4,92	98,41%	79,42	4,81	96,20%	305,67	4,62	92,40%	1730,39	4,52	90,40%
6	16,39	5,82	96,97%	66,87	5,71	95,21%	258,30323	5,47	91,12%	1472,28	5,31	88,54%
7	14,36	6,64	94,89%	57,99	6,59	94,11%	222,18304	6,36	90,80%	1296,37	6,03	86,19%
8	12,79	7,46	93,23%	51,65	7,40	92,45%	199,37252	7,08	88,54%	1176,22	6,65	83,12%
9	11,64	8,19	91,02%	46,97	8,13	90,36%	179,42893	7,87	87,45%	1079,02	7,25	80,54%
10	10,89	8,76	87,56%	43,35	8,81	88,12%	165,69229	8,52	85,23%	1024,81	7,63	76,32%
11	10,17	9,38	85,23%	40,16	9,51	86,47%	153,36447	9,21	83,71%	985,36	7,94	72,16%
12	9,43	10,11	84,23%	38,06	10,04	83,65%	144,84055	9,75	81,25%	956,39	8,18	68,15%
13	9,04	10,55	81,15%	36,48	10,47	80,56%	137,59394	10,26	78,95%	896,10	8,73	67,14%
14	8,69	10,97	78,36%	34,78	10,98	78,45%	131,25712	10,76	76,85%	885,09	8,84	63,12%
15	8,46	11,27	75,11%	33,45	11,42	76,14%	128,70316	10,97	73,15%	862,57	9,07	60,45%
16	7,85	12,14	75,88%	31,61	12,08	75,52%	121,22265	11,65	72,81%	826,29	9,47	59,16%
17	7,38	12,92	76,01%	29,91	12,77	75,12%	116,05241	12,17	71,58%	788,89	9,91	58,32%
18	6,89	13,84	76,89%	28,54	13,39	74,37%	111,77561	12,63	70,19%	785,75	9,95	55,30%
***itemset* Máximo**	10			12			13			15		
Confiança Mínima	Nº de Regras de Associação			Nº de Regras de Associação			Nº de Regras de Associação			Nº de Regras de Associação		
100%	4.566			25.308			115.126			589.524		
90%	6.257			37.893			153.424			657.475		
80%	8.478			48.898			161.056			784.318		
70%	9.563			59.365			211.363			954.228		

Tabela 5.5 - Resultados da base Chaff-SLDM3



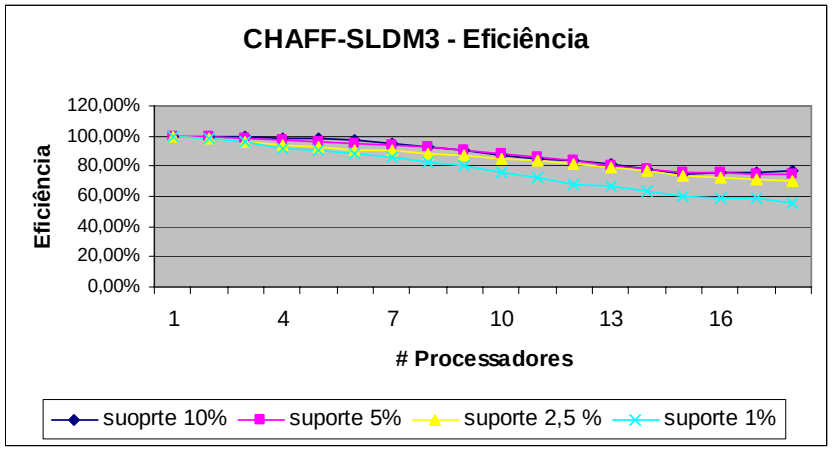


Figura 5.11 : Sp*eed Up* e Eficiência da base Chaff-SLDM3

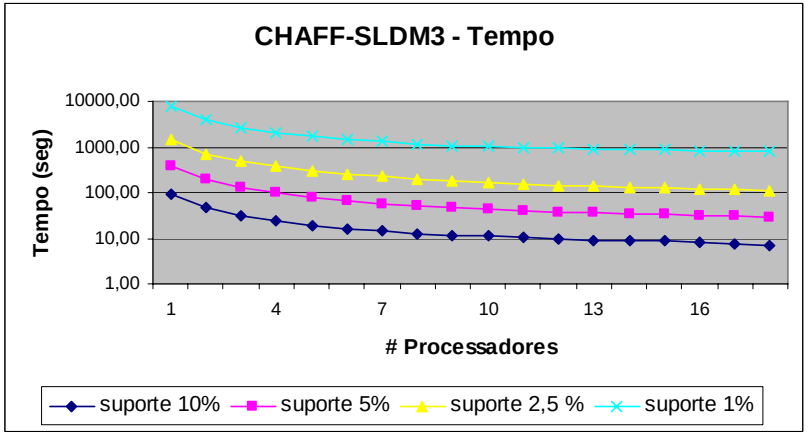
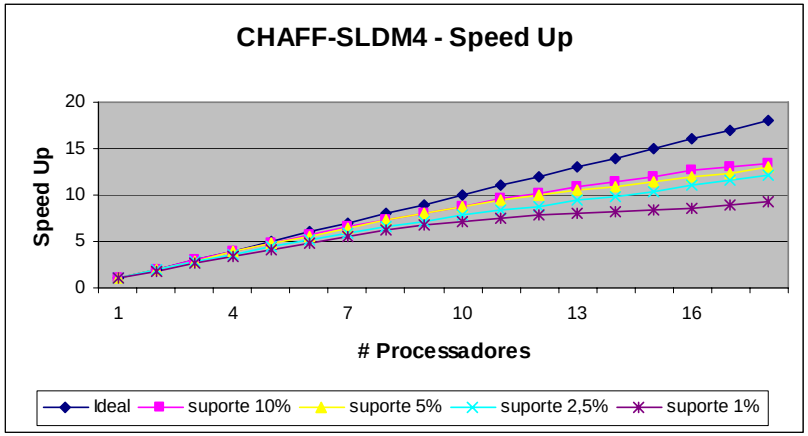


Figura 5.12 : Tempos de execução da base Chaff-SLDM3

Base	CHAFF-SLDM4											
Suporte Mínimo	10%			5%			2.5%			1%		
# Proc	Tempo(seg)	*SpeedUp*	Eficiência	Tempo(seg)	*SpeedUp*	Eficiência	Tempo(seg)	*SpeedUp*	Eficiência	Tempo(seg)	*SpeedUp*	Eficiência
1	200,40	1,00	100,00%	754,54	1,00	100,00%	2581,74	1,00	100,00%	10427,89	1,00	100,00%
2	101,32	1,98	98,89%	384,50	1,96	98,12%	1334,65	1,93	96,72%	5626,96	1,85	92,66%
3	67,78	2,96	98,56%	258,81	2,92	97,18%	920,90	2,80	93,45%	3925,87	2,66	88,54%
4	51,12	3,92	98,01%	195,58	3,86	96,45%	725,45	3,56	88,97%	3094,33	3,37	84,25%
5	40,97	4,89	97,83%	157,64	4,79	95,73%	588,40	4,39	87,75%	2543,39	4,10	80,85%
6	35,15	5,70	95,01%	134,56	5,61	93,46%	502,15	5,14	85,69%	2187,52	4,77	79,45%
7	30,49	6,57	93,89%	116,53	6,48	92,50%	441,86	5,84	83,47%	1906,45	5,47	78,14%
8	27,33	7,33	91,65%	102,64	7,35	91,89%	395,68	6,52	81,56%	1690,21	6,17	77,12%
9	24,87	8,06	89,54%	94,07	8,02	89,12%	362,43	7,12	79,15%	1555,03	6,71	74,51%
10	22,76	8,81	88,05%	86,15	8,76	87,58%	331,46	7,79	77,89%	1456,82	7,16	71,58%
11	20,77	9,65	87,73%	80,11	9,42	85,63%	309,31	8,35	75,88%	1383,12	7,54	68,54%
12	19,52	10,27	85,56%	75,60	9,98	83,17%	292,91	8,81	73,45%	1328,94	7,85	65,39%
13	18,54	10,81	83,16%	71,67	10,53	80,99%	274,87	9,39	72,25%	1286,31	8,11	62,36%
14	17,48	11,46	81,88%	68,81	10,97	78,33%	262,21	9,85	70,33%	1265,67	8,24	58,85%
15	16,67	12,02	80,12%	65,69	11,49	76,58%	248,79	10,38	69,18%	1233,49	8,45	56,36%
16	15,86	12,64	78,97%	63,37	11,91	74,42%	233,89	11,04	68,99%	1224,39	8,52	53,23%
17	15,46	12,96	76,23%	61,31	12,31	72,39%	221,35	11,66	68,61%	1173,31	8,89	52,28%
18	14,91	13,44	74,67%	55,65	13,01	72,28%	214,04	12,06	67,01%	1130,84	9,22	51,23%
***itemset* Máximo**	10			12			13			15		
Confiança Mínima	Nº de Regras de Associação			Nº de Regras de Associação			Nº de Regras de Associação			Nº de Regras de Associação		
100%	8.722			50.602			234.199			1.267.463		
90%	13.557			68.763			280.858			1.396.574		
80%	15.996			79.804			320.079			1.570.813		
70%	19.427			93.725			375.336			1.779.822		

Tabela 5.6 - Resultados da base Chaff-SLDM4



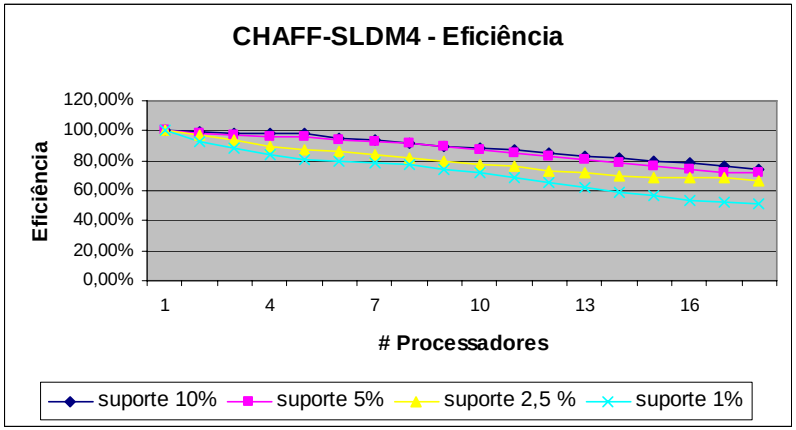


Figura 5.13 : S*peed Up* e Eficiência da base Chaff-SLDM4

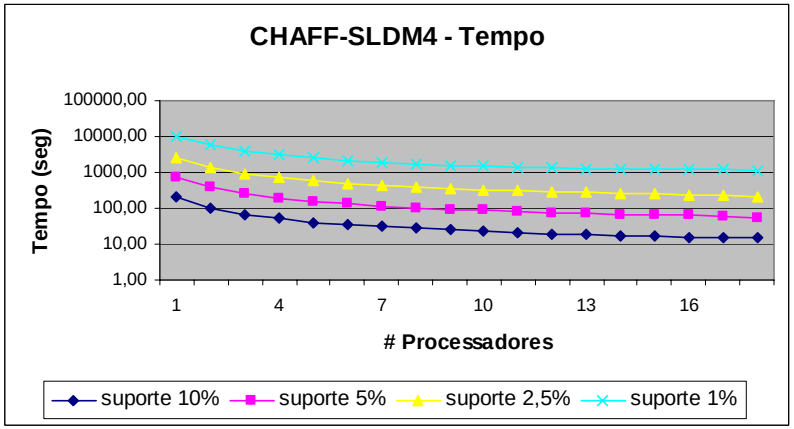


Figura 5.14 : Tempos de execução da base Chaff-SLDM4

Capítulo VI

Considerações Finais

Neste capítulo são apresentadas as conclusões desta tese e as sugestões para trabalhos futuros.

6.1 CONCLUSÕES

Esta tese teve como objetivo principal a implementação das tarefas de mineração de regras de associação e de classificação em ambiente de computação paralela, com a finalidade de contribuir para o aprimoramento de um sistema antimíssil para navios de superfície.

A implementação da tarefa de mineração de regras de associação foi baseada no algoritmo de distribuição de contagem que é uma versão paralela do algoritmo APRIORI, onde a base de dados é particionada entre as máquinas que executam o algoritmo. No desenvolvimento desta implementação foram experimentadas algumas formas de representação para os conjuntos de dados, tais como vetor de bits, produto de números primos e conjunto de inteiros. Porém, entre as formas de representação dos dados, o conjunto de inteiros foi a que apresentou melhor desempenho e, por isto, foi a forma de representação de dados utilizada nesta implementação.

A estrutura de dados utilizada, inicialmente, na implementação da mineração de itens freqüentes foi a árvore *hash*. Mas esta estrutura apresentou baixo desempenho, sobretudo, na fase de contagem dos suportes, devido às longas listas de conjuntos de dados armazenados em suas folhas. Substituiu-se, então, essas listas de dados das folhas da árvore *hash* por uma estrutura de busca denominada *Skip List*, na tentativa de melhorar o desempenho da contagem de suportes. Porém, o ganho não foi significativo devido ao excesso de consumo de memória causado pelas *Skip Lists*. A solução seria a substituição da árvore *hash* pela árvore de prefixos, usada em implementações seriais do APRIORI cujos desempenhos são muito bons. Entretanto, desenvolveu-se uma estrutura de dados mais simples chamada lista de classes, que apresentou desempenho satisfatório para os propósitos dessa implementação.

Para a implementação da tarefa de mineração de regras de classificação foi proposto um algoritmo paralelo baseado no CBA. O algoritmo proposto também utiliza particionamento da base de dados e é constituído de duas fases: a fase de geração de regras de classificação e a fase de construção do classificador. Na fase de geração de regras de classificação foram propostas duas formas distintas de implementação. A primeira, denominada de Gerador I, que gera exclusivamente regras de classificação, como no CBA. A segunda implementação, denominada de Gerador II, gera todas as regras cujos conseqüentes são conjuntos unitários, independentemente de estes serem ou não atributos de classes, para depois descartar as regras cujos conseqüentes não são uma classe. A implementação do Gerador II apresentou melhor desempenho do que a do Gerador I. Para a fase de construção do classificador aproveitou-se o particionamento dos dados realizado na fase de geração das regras. Isto trouxe um ganho no desempenho do algoritmo, já que, para a seleção das regras que compõem o classificador, são necessárias várias passagens sobre a base de dados.

Nos experimentos realizados foram utilizadas as bases de dados geradas a partir de um simulador que fornece cenários táticos e soluções de defesa para o sistema de contramedida. Os testes foram realizados em ambiente paralelo, isto é, utilizou-se um *cluster* de PC's, com a finalidade de verificar a escalabilidade das implementações. Os resultados dos testes mostram que foi possível a extração de regras de associação das bases de dados da aplicação e que com as regras de classificação extraídas foi possível a construção de um classificador com uma precisão bastante satisfatória. Além do mais, a

implementação apresentou bons resultados quanto à escalabilidade, o que permite a aplicação desta implementação em bases de dados de cenários táticos de maior escala.

A mineração de regras de associação ou a de classificação das bases de dados simuladas representa a etapa inicial para o aprimoramento dos sistemas de contramedida, pois, para a obtenção de resultados que efetivamente venham a contribuir para a melhoria desses sistemas, é necessário que haja continuidade dos trabalhos. Há necessidade, por exemplo, de se estabelecer, de forma mais criteriosa, os parâmetros suporte mínimo e confiança mínima, que influenciam diretamente a eficácia da metodologia. Valores muito elevados de suporte mínimo podem impedir a mineração de regras importantes ou reduzir muito a precisão do classificador. Assim, após a sintonia das implementações, é necessária uma fase de análise das informações reveladas pelas tarefas de mineração de dados para que se possam agregar, de forma efetiva, os novos conhecimentos aos sistemas de contramedidas. Além do mais, como há total conhecimento dos algoritmos e das estruturas de dados que compõem as implementações é possível a inclusão de melhorias nas tarefas já executadas, aproveitando-se do conhecimento prévio sobre os dados. Amplia-se, desta forma, as possibilidades de descoberta de novos conhecimentos nesta área.

Este trabalho alcançou os objetivos a que se propôs, sobretudo, quanto a aplicação de tarefas de mineração de regras com a finalidade de contribuir para o aprimoramento de sistemas antimíssil para navios. Todavia, a metodologia não se restringe ao caso em estudo, pois diversos outros sistemas militares podem se beneficiar da aplicação destas técnicas.

6.2 TRABALHOS FUTUROS

O trabalho desenvolvido e apresentado nesta tese é apenas o início de uma pesquisa de utilização da mineração de dados em aplicações militares. As implementações que foram feitas tinham o intuito de extrair regras das bases de dados que compõem o sistema antimíssil. Porém, muitas atividades ainda precisam ser desenvolvidas até que se possa agregar novos conhecimentos para o aprimoramento do sistema ou de seu uso.

Deixa-se, como sugestões para trabalhos futuros:

i. Pesquisas mais aprofundadas da influência do suporte mínimo e da confiança mínima nas bases de dados da aplicação, de modo a se obter maior eficácia tanto na mineração das regras de associação quanto na mineração das regras de classificação;

ii. Pesquisas do tamanho mínimo dos *itemsets* que devem ser utilizados para a construção do classificador, já que o tamanho dos antecedentes das regras pode alterar sua precisão.

iii. Aplicação de métodos para a redução do número regras associação, de modo a facilitar o trabalho de análise das regras na busca de informações relevantes; e

iv. Aplicação dessas e de outras tarefas de mineração de dados em outros sistemas militares.

Referências Bibliográficas

[1] FAYYAD U., PIATETSKY-SHAPIRO G. & SMYTH P. From Data Mining to Knowledge Discovery: An Overview, in Advances in Knowledge Discovery in Databases, U. Fayyad et al (Eds.) MIT Press. Cambridge, MA, USA, 1996.

[2] AGRAWAL R, T. IMIELINSKI & R. SRIKANT, Mining Association Rules between Sets of Items in Large Databases, Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pp 207–216, Washington, USA, 1993.

[3] AGRAWAL R., H. MANNILA, R. SRIKANT, H. TOIVONEN, & A. I. VERKAMO. Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, Fayyad U. and et al, editors, Advances in Knowledge, 1995.

[4] AGRAWAL R. & R. SRIKANT. Mining sequential patterns. In Proc. 11th Int. Conf. Data Engineering, ICDE, pp 3–14. IEEE Press, 1995.

[5] KEYUN H., YUCHANG L., LIZHU Z., & CHUNYI S. Integrating classification and association rule mining: A concept lattice framework. In RSFDGrC'99: International Workshop on Directions in Rough Sets, Data Mining, and Granular-Soft Computing, pp 443-447, London, UK, 1999.

[6] WALTER A. KOSTERS, ELENA MARCHIORI & ARD A. J. OERLEMANS. Mining clusters with association rules. In IDA '99: Proc. of the third international Symposium on Advances in Intelligent Data Analysis, pp 39-59, London, UK, 1999.

[7] MOBASHER B., JAIN N., HAN E. & SRIVASTAVA J. Web mining: Pattern discovery from World Wide Web transactions. Technical Report TR-96050, Department of Computer Science, university of Minnesota, USA, 1996.

[8] YEW KWONG W., WEE KEONG N. & Ee-PENG L.. On line and incremental mining of separately grouped web access logs. In WISE '02: Proc. Of the 3rd international Conference on Web information Systems Engineering, pp 53-62, Washington, DC, USA, 2002.

[9] ALVES A. S., Avaliação do problema de ordenação em diagramas de decisão binária, Tese de Mestrado, Universidade Federal do Rio de Janeiro, Fevereiro, Rio de Janeiro, Brasil, 2001.

[10] LIU B., HSU W., & MA Y. Integrating classification and association rule mining. In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98), pp 27–31, 1998.

[11] LIU B., HSU W., & MA Y. Pruning and summarizing the discovered associations. In Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining (SIGKDD 99), 1999.

[12] LIU B., MA Y., & WONG C.. Improving an association rule based classifier. In 4th European Conference on Principles and Practice of Knowledge Discovery in Databases PKDD, pp 504–509, 2000.

[13] GIRISH K. PALSHIKAR A, MANDAR S. KALE B & MANOJ M. APTE. Association rules mining using heavy itemsets, In Tata Research Development and Design Centre, Engineering and Industrial Services, Tata Consultancy Services Ltd., Pune 411001, India, 2006.

[14] JEAN-MARC ADAMO, Data Mining for Association Rules and Sequential Patterns: Sequential and Parallel Algorithms, Springer, New York, NY, USA 2000.

[15] HALSTEAD J., & SULLIVAN S. (United States Military Acad.) Improved Ballistic Test and Evaluation Methodology. In 2006 IEEE Systems and Information Engineering Design Symposium, University of Virginia, Charlottesville, VA, USA , 2006.

[16] CANTONI V., LOMBARDI L. & LOMBARDI P. Challenges for Data Mining in Distributed Sensor Networks. In The 18th International Conference on Pattern Recognition. China, Hong Kong (ICPR'06),20-24 August 2006.

[17] SERGEI A. VAKIN, LEV N. SHUSTOV & ROBERT H. DUNWELL. Fundamentals of Electronic Warfare. Artech House, INC. Norwood, MA, USA 2001.

[18] WENMIN L., JIAWEI H., & JIAN P. CMAR: Accurate and efficient classification based on multiple class-association rules. In Proc 2001 IEEE International Conference on Data Mining (ICDM 2001), pp 369–376. IEEE Computer Society Press, 2001.

[19] MERET A.S DRDI WUTHRICH B. Exte01,ng Data
[14]

[22] XIAOXIN Y. & JIAWEI H.. CPAR: Classification based on Predictive Association Rules. In SDM, 2003.

[23] BEEFERMAN D., BERGER A., & LAFFERTY J. Statistical models for text segmentation. Machine Learning, 34(1-3):177.210, 1999.

[24] QUINLAN J. R. & CAMERON-JONES R. M.. FOIL: A mid-term report. In Proc. of European Conference on Machine Learning (ECML), pp 3.20, 1993.

[25] QUINLAN J. R.. C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993.

[26] CLARK P.& NIBLETT T. The CN2 induction algorithm. Machine Learning, 2:261.283, 1989.

[27] DUDA R.& HART P. Pattern Classification and Scene Analysis. John Wiley & Sons, 1973.

[28] LIM T.-S.,. LOH W.-Y, & SHIH Y.-S. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. Machine Learning, 40(3):203. 228, September 2000.

[29] DONG G., ZHANG X., WONG L., & LI J.. Classification by aggregating emerging patterns. In Discovery Science, December 1999.

[30] MERETAKIS D. & WUTHRICH B.. Extending naive-bayes classifiers using long itemsets. In KDD, pp 165.174, 1999.

[31] MICHALSKI R. S. Pattern recognition as rule-guided inductive inference. In Proceedings of IEEE Transactions on Pattern Analysis and Machine Intelligence, pp 349–361, 1980.

[32] MICHALSKI R., MOZETIC I., HONG J., & LAVRAC N. The AQ15 inductive learning system: an overview and experiments. In Proceedings of IMAL 1986, Orsay, 1986.

[33] CLARK P. & BOSWELL R. Rule induction with CN2: Some recent improvements. In Machine Learning - EWSL-91, pp 151–163, 1991.

[34] MITCHELL T. M. Machine Learning. (eds) McGraw-Hill, IBSN-0-07-042807-7, USA 1997.

[35] F¨URNKRANZ J. Separate-and-conquer rule learning. Artificial Intelligence Review, 13(1): 3–54, January 1999.

[36] BREIMAN l., FRIEDMAN J. H., OLSHEN R. A., and c. STONE J. Classification and Regression Trees. Wadsworth International Group, 1984.

[37] RUMELHART D. E. & McCLELLAND J. L. Learning internal representations by error propagation. In Explorations in the Micro-Structure of Cognition Vol. 1: Foundations, pp 318–362. MIT Press, Cambridge, MA, 1986.

[38] RIPLEY B. D. Pattern Recognition and Neural Networks. Cambridge University Press, Cambridge, 1996.

[39] HOLLAND J. H. Escaping brittleness: the possibilities of general purpose algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, Machine Learning, an Artificial Intelligence approach, volume 2, pp 593–623. Morgan Kaufmann, San Mateo, California, 1986.

[40] FRIEDMAN N., GEIGER D., GOLDSZMIDT M., PROVAN G., LANGLEY P., & SMYTH P. Bayesian network classifiers. Machine Learning, 29:131, 1997.

[41] MINGERS J. An empirical comparison of selection measures for decision tree induction. Machine Learning, 3: 319–342, 1989.

[42] BREIMAN L. Bagging predictors. Machine Learning, 24:123–140, 1996.

[43] FREUND Y. & SCHAPIRE R. E. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1):119–139, 1997.

[44] S. CLEARWATER & F. PROVOST. RL4: A tool for knowledge-based induction. In the Second International IEEE Conference on Tools for Artificial Intelligence, pp 24–30, 1990.

[45] RYMON R. An SE-tree based characterization of the induction problem. In Proceedings of the Tenth International Conference on Machine Learning, pp 268–275, Amherst, Morgan Kaufmann, 1993.

[46] SCHLIMMER J. C. Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning. In Proceedings of the Tenth International Conference on Machine Learning, pp 284–290, Amherst, Morgan Kaufmann, 1993.

[47] SEGAL R. & ETZIONI O. Learning decision lists using homogeneous rules. In Proceedings of the 12th National Conference on Artificial Intelligence. Volume 1, pp 619–625, Menlo Park, CA, USA, AAAI Press, 1994.

[48] RYMON R. Search through systematic set enumeration. In Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning, pp 539–552, Cambridge, MA, Morgan Kaufmann, 1992.

[49] NARENDRA P. M. & FUKUNAGA K. A branch and bound algorithm for feature subset selection. IEEE Transactions on Computers, 26:917–922, 1977.

[50] WEBB G. I.. OPUS: An efficient admissible algorithm for unordered search. In Journal of Artificial Intelligence Research, volume 3, pp 431–465, 1995.

[51] HART P. E., NILSSON N. J., & RAPHAEL B.. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4(2):100–107, 1975.

[52] WEBB G. I.. Efficient search for association rules. In Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-00), pp 99–107, N. Y., ACM Press, 2000.

[53] ZHENG Z., KOHAVI R., & MASON L.. Real world performance of association rule algorithms. In Proceedings of the seventh International Conference on Knowledge Discovery and Data Mining (SIGKDD-01), 2001.

[54] JIYONG L. (MPHIL), Optimal and Robust Rule set Generation, PhD thesis, School of Computer and Information Technology, Griffith University, Brisbane, Australia, 2002.

[55] ZAKI M. J., PARTHASARATHY, S., OGIHARA, M. & LI, W., Evaluation of Sampling for Data Mining of Association Rules, KDD97, 1997.

[56] ZAKI M. J., Scalable Data Mining for Rules, PhD Thesis, Department of Computer Sciences, University of Rochester, 1998.

[57] ZAKI M. J. & OGIHARA, M., Theoretical Foundations of Association Rules, Department of Computer Sciences - University of Rochester, 1998.

[58] ZAKI M. J, C. HSIAO, "CHARM: An Efficient Algorithm for Closed Itemset Mining", In Proc. SDM'02, SIAM, pp. 457-473, 2002.

[59] ZAKI M, "Parallel and Distributed Associating Mining: A Survey", IEEE Concurrency December -1999.

[60] ZAKI M. J. et al., "Parallel Algorithms for Fast Discovery of Association Rules", Data Mining and Knowledge Discovery. An Int'l J., Vol. 1, No. 4, pp. 343-373, 1997.

[61] KITSUREGAWA M. & SHINTANI T. "Hash Based Parallel Algorithms form Mining Association Rules". Proc 4th. Int'l Conf. Parallel and Distributed Information Systems. IEEE Computer Soc. Press, Los Alamitos, Calif., pp 19-30, 1996.

[62] HOUTSMA M. & SWAMI A., Set-oriented Mining of Association Rules in Relational Databases, Technical Report, RJ 9567. IBM Almaden Research Center, 1995.

[63] MUELLER A, Fast Sequential and Parallel Algorithms for Association Rule Mining, Master Thesis, Department of Computer Science, University of Maryland-College Park, August 1995.

[64] PASQUIER N., BASTIDE Y., TAOUIL R., & LAKHAL L.. Pruning closed itemset lattices for association rules. In Actes Bases de Donncées Avancées BDA'98, Hammamet, Tunisie, Oct. 1998.

[65] PASQUIER N., BASTIDE Y., TAOUIL R., & LAKHAL L.. Efficient mining of association rules using closed itemset lattices. Information Systems, 24(1):25,46, Jan. 1999.

[66] WILLE R.. Restructuring lattice theory: An approach based on hierarchies of concepts. In I. Rival, editor, Ordered Sets, pp 445-470. Reidel, Dordrecht-Boston, 1982.

[67] VELOSO A., MEIRA W., & CARVALHO M.B. CARVALHO, Real world association mining. Advantages in Data Bases: British Nat. Conf. on Databases, BNCOD, Lecture Notes in Computer Science, 2405:73-77, 2002.

[68] GOUDA K. & ZAKI M. J.. Efficiently mining maximal frequent itemsets. In International Conference of Data Mine (ICDM'01), pp 163–170, 2001.

[69] RAMAKRISHNAN S. & AGRAWAL R., Mining Quantitative Association Rules in Large Relational Tables, Proc of the 1996 ACM SIGMOD International Conference on Management of Data, pp. 1-12, Montreal, Quebec, Canada, 4-6 June1996.

[70] PARK J.S., CHEN M.S., & YU P.S., Efficient Parallel Data Mining for Association Rules, Proc. Int'l Conf. Information and Knowledge Management, 1995.

[71] PARK J.S., CHEN M.S., & YU P.S., An Effective Hash Based Algorithm for Mining Association Rules, ACM SIGMOD Conf. Management of Data, 1995.

[72] SAVASERE A., OMIECINSKI, E. & NAVATHE, S., An Efficient Algorithm for Mining Association Rules in Large databases, Proc of the 21nd International Conference on Very Large Databases, pp. 432-444, Zurich, Swizerland,1995.

[73] BRIN S., MOTWANI R., ULLMAN J. D., & TSUR S., Dynamic Itemset Counting and Implication Rules for Market Basket Data, Proceedings of the ACM SIGMOD Conference, pp. 255-264, 1997.

[74] GUNOPULOS, D., MANNILA, H., and SALUJA, S., "Discovering All Most Specific Sentences using Randomized Algorithms", Proc. ofICDT'97, pp. 215 – 229, 1997.

[75] BAYNARDO Jr. & ROBERT J., Efficient Mining Long Patterns from Databases, ACM SIGMOD Conf. Management of Data, 1998.

[76] HAN J., PEI H., & YIN Y.. Mining Frequent Patterns without Candidate Generation. In: Proc. Conf. on the Management of Data (SIGMOD'00, Dallas, TX). ACM Press, New York, NY, USA, 2000.

[77] D. MERETAKIS & B. WUTHRICH. Extending naive-bayes classifiers using long itemsets. In KDD, pp 165-174, 1999.

[78] K.WANG, S. Q. ZHOU, & Y. HE. Growing decision trees on supportless association rules. In Proc.of 4th Intl. Conf. on Knowledge Discovery and Data Mining (KDD), August 2000.

[79] WENMIN L., JIAWEI H., & JIAN P. CMAR: Accurate and efficient classification based on multiple class-association rules. In ICDM, 2001.

[80] XIAOXIN Y. & JIAWEI H.. CPAR: Classification based on Predictive Association Rules. In SDM, 2003.

[81] MORISHA S. & SESE J., Traversing itemset lattices with statistical metric pruning. In PODS, Dallas, Texas, USA, ACM pp.226-236, 2000.

[82] A. ZIMMERMANN & L. DE RAEDT, Correlated Association Rule Mining for Classification. In DS vol. 3245/2004 – pp 60-72, Spriger-Verlag Berlin Heidelberg, 2004.

[83] SHAFER J. & AGRRAWAL R.. " Parallel Mining of Association Rules". IEEE Trans. Knowledge and Data Eng., Vol. 8. No. 6. pp 19-30, Dec. 1996.

[84] HAN E.H., KARYPIS G., & KUMAR V. "Scalable Parallel Data Mining for Association Rules" Proc. ACM Conf. Management of Data, ACM Press, pp. 277-288, New York, 1997.

[85] CHEUNG D.et al.," A Fast Distributed Algorithm for Mining Association Rules". Proc 4th. In I Conf. Parallel and Distributed Information System, IEEE Computer Soc. Press. Los Alamitos, Calif., pp 31-42, 1996.

[86] Sítio da Internet: Frequent Itemset Mining Dataset Repository http://fimi.cs.helsinki.fi/data.

[87] BODON F., A Survey on Frequent Itemset Mining, Department of Computer Science and Information Theory, Budapest University of Technology and Economics, pp 25-26, Budapest, April 28, 2006.

[88] PUGH, W., Whatever you might want to do using Balanced Trees, you can do it faster and more simply using Skip Lists, Tech Report CS–TR–2286, Dept. of Computer Science, University of Maryland, College Park, July 1989.

[89] Sítio da Internet : The Message Passing Interface (MPI) standard - http://www-unix.mcs.anl.gov/mpi/

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo