UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Um estudo de algoritmos para a extração de regras baseados em Análise Formal de Conceitos

Renato Vimieiro

Belo Horizonte, Minas Gerais

16 de fevereiro de 2007

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Um estudo de algoritmos para a extração de regras baseados em Análise Formal de Conceitos

Renato Vimieiro

Orientador: Newton José Vieira

Belo Horizonte, Minas Gerais

16 de fevereiro de 2007

DISSERTAÇÃO DE MESTRADO

Um estudo de algoritmos para a extração de regras baseados em Análise Formal de Conceitos

---

Renato Vimieiro
Aluno do Programa de Pós-Graduação em Ciência da Computação - Nível Mestrado

---

Prof. Dr. Newton José Vieira
Orientador
Universidade Federal de Minas Gerais
Instituto de Ciências Exatas - Departamento de Ciência da Computação

---

Prof. Ph.D. Rodolfo Sérgio Ferreira Resende
Membro da Banca
Universidade Federal de Minas Gerais
Instituto de Ciências Exatas - Departamento de Ciência da Computação

---

Prof. Dr. Luis Enrique Zárate
Membro Externo da Banca
Pontifícia Universidade Católica de Minas Gerais
Instituto de Informática - Departamento de Ciência da Computação

Belo Horizonte, 16 de fevereiro de 2007

Resumo

Este trabalho apresenta uma análise comparativa de técnicas para a extração de regras de bancos de dados através da Análise Formal de Conceitos (AFC). As regras consideradas aqui são conjuntos de dependências entre atributos de bancos de dados. Especificamente, as dependências são: implicações, dependências funcionais, regras de associação e regras de classificação. Essas regras são originárias, principalmente, da teoria dos bancos de dados, na qual desempenham papel fundamental para auxiliar processos de tomada de decisão – caso das implicações, regras de associação e classificação – e na normalização de modelos lógicos – caso das dependências funcionais. A AFC, por sua vez, possui uma estrutura matemática especialmente adequada para auxiliar na análise de dados. Essa análise é feita através de reticulados conceituais que representam dados de forma hierárquica. Sendo assim, o objetivo do trabalho é analisar e comparar métodos que utilizem a AFC para a descoberta de dependências entre atributos em bancos de dados.

São analisados dez algoritmos representativos para extração dos quatro tipos de regras mencionados. Desses algoritmos, quatro são usados na identificação de dependências funcionais e implicações. São eles: *Next Closure*, *Find Implications*, *Impec* e *Aprem-IR*. Os seis algoritmos restantes são úteis na identificação de regras de associação e de classificação. Foram analisados quatro algoritmos para extrair regras de associação: *AClose*, *Frequent Next Neighbours*, *Titanic* e *Galicia*. Finalmente, foram analisados dois algoritmos para identificar regras de classificação: *GRAND* e *Rulearner*.

Os algoritmos foram implementados e submetidos a bancos de dados reais e sintéticos. Os bancos de dados foram escolhidos e gerados segundo dois critérios: tamanho da base de dados (número de entradas) e densidade. Esses dois critérios tentam suprir a deficiência constatada na literatura no que diz respeito à escolha de bancos de dados para avaliação de algoritmos. Constatou-se que os algoritmos apresentam comportamentos característicos para diferentes bancos de dados. Neste trabalho, é sugerida a adequação de cada algoritmo aos bancos de dados com diferentes densidades e tamanhos.

Abstract

This work presents a comparative analysis of techniques for extracting rules from databases through Formal Concept Analysis (FCA). The rules considered here are sets of dependencies among attributes of databases. Specifically, the dependencies are: implications, functional dependencies, association rules and classification rules. Those rules are mainly sourcered in databases theory in which they have a fundamental role as a way of helping with the process of decisions' taken – case of implications, association rules and classification rules – and with normalizing logical models – case of functional dependencies. The FCA has a mathematical structure especially adequate for helping in data analysis. Such analysis is done through concept lattices that represent data in a hierachical manner. So, the objective of this work is the analysis and the comparison of methods that use FCA for discovering dependencies among attributes of databases.

It has been analyzed ten representative algorithm for extracting the four types of rules mentioned. From those algorithms, four are used in the extraction of functional dependencies and implications. They are: *Next Closure*, *Find Implications*, *Impec* and *Aprem-IR*. The last six algorithms are useful for extracting association and classification rules. Four algorithms have been analyzed for the extraction of association rules: *AClose*, *Frequent Next Neighbours*, *Titanic* and *Galicia*. Finally, two algorithms have been analyzed for the extraction of classification rules: *GRAND* and *Rulearner*.

The algorithms have been implemented and submitted to real and synthetic databases. The databases have been chosen with two criteria: database's size (number of entries) and density. Those criteria try to eliminate a deficiency detected in the literature in choosing databases for algorithms' evaluation. One noted that those algorithms have characteristic behaviors for different databases. In this work, it is suggested the adequacy of each algorithm to databases with different densities and sizes.

Sumário

1 Introdução

Neste trabalho, é feita uma análise e comparação de técnicas para a extração de regras de bancos de dados através da Análise Formal de Conceitos (AFC)[1]. Entende-se como regras conjuntos de dependências entre atributos de bancos de dados. Dessa forma, como será visto à frente, pretende-se analisar técnicas baseadas na AFC para a identificação de implicações, dependências funcionais, regras de associação e regras de classificação. Essas regras são originárias, principalmente, da teoria dos bancos de dados, na qual desempenham papel fundamental para auxiliar processos de tomada de decisão, caso das implicações, regras de associação e classificação, e na normalização de modelos lógicos, caso das dependências funcionais. A AFC, por sua vez, possui uma estrutura matemática especialmente adequada para auxiliar na análise de dados. Essa análise é feita através de reticulados conceituais que representam dados de forma hierárquica. Sendo assim, o objetivo do trabalho é analisar e comparar métodos que utilizem a AFC para a descoberta de dependências entre atributos em bancos de dados.

A AFC é um ramo da matemática aplicada que utiliza o arcabouço da teoria de reticulados[2] na concepção e análise de hierarquias de conceitos [20]. O marco inicial da AFC data do início da década de 1980. Mais precisamente, a AFC tem seu início com o trabalho de Rudolf Wille [57] publicado em 1982, que propõe um arcabouço formal para a utilização da teoria de reticulados. Nos últimos anos, a AFC tem "migrado" da Matemática para a Ciência da Computação [49]. Inicialmente, os trabalhos relacionados à AFC eram apresentados em conferências da área de Matemática, enquanto atualmente, os trabalhos têm sido, majoritariamente, publicados em conferências na área de Ciência da Computação. Essa característica mostra uma transformação na visão sobre a AFC: enquanto, no passado, a AFC era estudada apenas do ponto de vista teórico, nos últimos anos, ela tem sido explorada sob o ponto de vista prático, isto é, além do desenvolvimento de teorias relacionadas a AFC, os trabalhos atuais investigam sua utilização nas diversas áreas do conhecimento humano, principalmente na Ciência da Computação. Entre as aplicações da AFC na Ciência da Computação destacam-se aquelas para a recuperação de informação [14, 35] e para mineração e descoberta de conhecimento em bancos

[1] Em inglês *Formal Concept Analysis*.

[2] Uma visão detalhada sobre teoria de reticulados pode ser obtida em [10].

de dados [51, 59].

O livro sobre Análise Conceitual de Dados [14] confirma a aproximação da AFC à Ciência da Computação, apresentando a técnica de forma mais algorítmica que teórica em contraste com o livro de Ganter e Wille ([31]) que apresenta a AFC sob o ponto de vista teórico e matemático. Em [14], foram dedicados capítulos exclusivos para mostrar aplicações da AFC na Ciência da Computação. Os capítulos 3 e 4, por exemplo, apresentam métodos baseados na AFC para auxiliar no processo de recuperação de informação. Já o capítulo 5, é dedicado a apresentação de métodos para extração de regras de bancos de dados através da AFC.

Apesar do livro sobre análise conceitual de dados mostrar métodos para extração de regras, os algoritmos e métodos foram apresentados de forma breve. Este trabalho tenta suprir as deficiências constatadas no livro de Carpineto e Romano [14] no que diz respeito à cobertura do número de algoritmos, assim como a análise prática de suas complexidades, além de demonstrações de aplicações, uma vez que as apresentadas na referência têm como objetivo ilustrar o funcionamento dos algoritmos e não o de fornecer exemplos de aplicações.

A sociedade atual encontra-se em um período cuja maior riqueza é o conhecimento. Informações transformaram-se em *ativos* nas organizações. Assim, a capacidade de gerir informações adequadamente tornou-se um diferencial para as empresas. Remetendo-se à Ciência da Informação, informações são dados estruturados de maneira a produzir sentido. Esta mesma ciência define o seguinte ciclo: Dados $\rightarrow$ Informação $\rightarrow$ Conhecimento. Sob esta ótica, dados geram informações que, por sua vez, geram conhecimento o qual completa o ciclo gerando mais dados [19].

O avanço da tecnologia facilitou o processo de coleta e armazenagem de dados o que resultou no aumento da quantidade armazenada. Esta quantidade de dados provenientes de diversas fontes torna inexeqüível a análise manual destes para a geração de informações. Então, tornam-se necessárias técnicas para automatizar ou auxiliar o processo de obtenção de informações a partir de dados.

Neste sentido, diversas técnicas têm sido propostas na área de mineração de dados. Essas técnicas baseiam-se, em sua maioria, em conceitos estatísticos. Entretanto, há também aquelas baseadas em computação natural ou bio-inspiradas. No geral, essas correntes, apesar de distintas, possuem pontos em comum, como o fato de utilizarem a distribuição de probabilidades para analisarem os dados. Alternativamente, a AFC utiliza semelhanças estruturais dos dados, como a relação de inclusão em conjuntos, para a análise. Dessa forma, acredita-se que os métodos baseados na AFC podem propiciar melhores resultados na análise dos dados por realçar as relações estruturais. Este argumento justifica a análise e comparação de métodos baseados na AFC.

Como supracitado, uma das aplicações da AFC dá-se no campo da mineração de dados. Esta técnica basicamente tenta descobrir relacionamento entre dados. Descoberto um relacionamento, este pode ser expressado através de regras.

Os reticulados conceituais permitem a extração de regras determinísticas e probabilísticas. O conjunto de regras determinísticas é formado por regras de implicações (freqüentemente chamadas apenas de implicações) e dependências funcionais. As regras probabilísticas dividem-se em regras de associação e de classificação.

Para se ter uma idéia do que são as implicações, considere o universo dos animais. Se o animal é terrestre, então ele possui pulmões, já que todo animal terrestre possui respiração pulmonar. Este tipo de relacionamento entre atributos, aqui características dos animais, podem ser descritos por implicações. Assim, a implicação *terrestre implica pulmões* é uma forma de reescrever que todo animal possuindo a característica (atributo) *terrestre*, também possui a característica *pulmões*. Generalizando, uma implicação X *implica* Y revela que, no universo considerado, todo elemento possuindo o atributo X, também possui o atributo Y.

Os trabalhos relativos ao uso da AFC para extração de implicações sugerem diversos algoritmos para a extração de diversas bases de implicações[3]. O algoritmo *Next-Closure*, proposto por Ganter [27, 28], é utilizado para encontrar uma base de implicações mínima (*Stem-Base* ou, também, *Duquenne-Guigues base*) sugerida por Duquenne e Guigues [23]. A base de implicações mínima fornece um conjunto de implicações completo, no sentido de que qualquer implicação válida para o banco de dados pode ser obtida através da combinação das regras da base de implicações, e não-redundante, ou seja, a remoção de uma regra da base mínima faz com que a base deixe de ser completa.

Apesar de fornecer um número mínimo de regras, o problema do uso da base mínima para fins práticos está na dificuldade da derivação de todas as regras válidas para os dados através da combinação das regras da base de implicações. Carpineto e Romano [13] discutem este problema e apresentam um algoritmo para encontrar uma base de implicações mais legível (i.e. mais facilmente compreensível para usuários finais leigos). O algoritmo proposto encontra bases de implicações com um número reduzido de atributos tanto no antecedente quanto no conseqüente de uma implicação[4]. Taouil e Bastide [52] apresentam outra proposta para a extração de bases de implicações mais legíveis. O algoritmo proposto por eles encontra um conjunto de regras com um antecedente de tamanho mínimo e um conseqüente contendo apenas um elemento.

[3]Entende-se como base de implicações o conjunto de implicações extraídas de um banco de dados.

[4]O antecedente representa o lado esquerdo, enquanto o conseqüente representa o lado direito de uma implicação. Seja $A \rightarrow B$ uma implicação; A representa o antecedente e B o conseqüente.

Dependências funcionais possuem estreita relação com as implicações. Elas, como as implicações, também descrevem relacionamentos entre atributos que são válidos para todos os elementos do universo. Sendo assim, qual a diferença entre implicações e dependências funcionais? As dependências funcionais são implicações envolvendo atributos multivalorados, atributos que podem assumir diversos valores, como, por exemplo, a cor de um carro e o sexo de uma pessoa. As dependências funcionais são, então, implicações que independem dos valores dos atributos. O exemplo a seguir ilustra o que são as dependência funcionais. Sabe-se que, no Brasil, uma pessoa possui um único cadastro de pessoa física (CPF) e, também, que toda pessoa tem um nome. Uma dependência funcional entre estes atributos dos brasileiros é: *CPF determina funcionalmente NOME*, já que sabido o número do CPF de uma pessoa, sabe-se o nome dela, independentemente dos valores do CPF e do nome. Logo, para quaisquer duas pessoas, se elas possuem o mesmo número de CPF, então elas possuem o mesmo nome.

As dependências funcionais estão, fundamentalmente, relacionadas à área de modelagem de bancos de dados. Diversos trabalhos dedicam-se ao estudo do tópico desde sua origem [16]. Entre os trabalhos que se destacam no tratamento de dependência funcional sob a ótica da modelagem de bancos de dados, encontram-se os livros de Elmasri e Navathe [24], de Ullman [54], de Date [18] e de Molina, Ullman e Widom [32].

O problema da identificação de dependências funcionais através da AFC pode ser reduzido ao problema da identificação de implicações. Pode-se transformar um conjunto de atributos multivalorados em um conjunto de atributos univalorados de forma que as implicações válidas para os atributos univalorados sejam as dependências funcionais válidas para os atributos multivalorados. Esta abordagem foi inicialmente apresentada por Wille [58]. Alternativamente, pode-se utilizar o arcabouço proposto por Jaume Baixeries [5]. Este método propõe novos operadores de fecho para a construção do reticulado conceitual. Construído o reticulado, este é utilizado para a descoberta de implicações equivalentes às dependências funcionais do reticulado conceitual construído com os operadores tradicionais.

As regras probabilísticas descrevem fatos que ocorrem parcialmente, por exemplo, *se uma pessoa tem mais que dezesseis anos então ela possui título de eleitor*. Observe que existem pessoas com mais de dezesseis anos que não possuem título de eleitor; assim, a regra é válida parcialmente. Esse tipo de relacionamento pode ser descrito através de regras de associação. Elas descrevem comportamentos dos dados que são válidos para grupos específicos (no exemplo, para o grupo das pessoas com mais de dezesseis anos e título de eleitor) e não obrigatoriamente para todos os elementos como as implicações. Dessa forma, regras de associação podem ser consideradas generalizações de implicações.

Vários trabalhos [7, 45, 50, 51, 59] discutem a utilização da AFC para extração de regras

de associação. Além da demonstração da possibilidade da utilização de conjuntos fechados para encontrar atributos freqüentes em bancos de dados, Pasquier e outros [45] apresentam o algoritmo *AClose*. O algoritmo foi comparado com o tradicional e reconhecido algoritmo *Apriori* [2, 3] e, em geral, apresentou melhores resultados. De acordo com Pasquier e seus co-autores [45], o espaço de soluções é reduzido ao utilizar-se reticulados conceituais para a extração de regras de associação o que permitiu o melhor desempenho do *AClose* em relação ao *Apriori*. Além do *AClose*, pode-se citar, entre os principais algoritmos baseados em AFC, o *Pascal* [6], o *Titanic* [50], o *Frequent Next Neighbours* [14] e o Galicia [55].

As regras de classificação são úteis para descrever características de classes de objetos. Descrevendo de uma forma simplista, uma regra de classificação indica atributos que caracterizam uma classe. Considerando novamente o universo dos animais, uma regra de classificação seria: *glândulas mamárias determina a classe mamífero*. Obviamente, a presença de glândulas mamárias é uma característica comum a todos os animais pertencentes à classe dos mamíferos.

A utilização da AFC para a extração de regras de classificação ainda é pouco explorada, embora tenha sido uma das primeiras aplicações de reticulados conceituais em mineração de dados [14]. Mesmo pouco explorada, pode-se apontar alguns trabalhos com tal abordagem. Um dos primeiros trabalhos com a utilização de reticulados para classificação é o trabalho de Ganascia [26]. Além deste, entre os principais trabalhos estão as referências [12, 29, 39, 40, 43, 44, 47]. O trabalho de Sahami [47] propõe um algoritmo que utiliza o reticulado conceitual como um mapa no espaço de regras possíveis. O algoritmo apresentado em [47], denominado *Rulearner*, foi comparado com outros métodos de classificação, como árvores de decisão, e apresentou melhores resultados.

A identificação de comportamentos de dados e sua expressão através de regras tem sido um tópico de interesse em várias pesquisas ao redor do mundo. A AFC forneceu um arcabouço formal para a representação de dados. Os dados, na AFC, são representados através de diagramas o que facilita a análise de dependências entre atributos. Sendo assim, o problema considerado neste trabalho é resumido na seguinte frase: *Dado um reticulado conceitual, quais são as regras que podem ser extraídas deste reticulado?*

Os objetivos deste trabalho são *avaliar* e *comparar* os métodos propostos na literatura para a extração de regras de reticulados conceituais. O processo de avaliação consiste em verificar a aplicabilidade, as possíveis utilizações dos algoritmos e das regras geradas por eles, além de analisar o comportamento assintótico (tempo de execução e espaço necessários) e o desempenho dos principais métodos identificados na literatura. A comparação é feita somente entre os métodos baseados na AFC; não foram comparados os métodos baseados na AFC com outros métodos propostos na literatura. A comparação é realizada entre métodos de uma mesma classe

(métodos que produzem o mesmo tipo de regra). O objetivo desta comparação é verificar a adequação dos métodos sob os pontos de vista da eficiência no tempo de execução e da utilidade das regras produzidas.

Os demais capítulos estão estruturados da seguinte forma: No capítulo seguinte, uma breve introdução à AFC é realizada. São apresentados conceitos básicos como o de *contextos formais*, *conceitos formais* e *reticulados conceituais*. No Capítulo 3, os algoritmos para a identificação de regras determinísticas são apresentados. Além dos algoritmos, são apresentadas definições formais de implicações e dependências funcionais. Como as dependências funcionais estão relacionadas à área de modelagem de banco de dados, no Capítulo 3, é, também, discutida a representação de bancos de dados através de contextos formais. O Capítulo 4 apresenta as definições e algoritmos relacionados às regras de associação e de classificação. Finalmente, o Capítulo 5 apresenta a conclusão do trabalho.

2 *Análise Formal de Conceitos*

Neste capítulo, serão introduzidos conceitos básicos sobre AFC. A AFC é um método proposto para a análise de dados estruturados como conceitos formais, entidades matemáticas que formalizam, simplificadamente, a concepção abstrata de *conceitos* como manifestações do pensamento humano [30].

A AFC baseia-se na teoria de reticulados [10, 20] para estruturar conceitos formais e permitir a análise de dados. Como reticulados são tipos de ordens parciais que possuem certas propriedades, então, antes de apresentar definições relativas à AFC, serão apresentados conceitos básicos sobre ordens parciais e reticulados.

2.1 Ordens Parciais

Primeiro, segundo, terceiro ... Avô, pai, filho ... Mais alto, mais largo. Enfim, a idéia de ordem surge freqüentemente no cotidiano das pessoas. Mas qual o conceito de ordem? Ordem, de acordo com a definição do dicionário Aurélio [21], é:

> ***ordem*** *(...) 2. Disposição metódica; arranjo de coisas segundo certas relações. (...)*

Ordem, então, é uma relação estabelecida entre objetos através da qual pode-se inferir a posição de um objeto dentro de uma seqüência.

Uma relação de ordem, $\leq$, é uma relação binária definida sobre um conjunto P que respeita as três propriedades a seguir:

reflexividade $x \leq x$, para todo $x \in P$;

transitividade Se $x \leq y$ e $y \leq z$, então $x \leq z$, para todo $x, y, z \in P$;

anti-simetria Se $x \leq y$ e $y \leq x$, então $x = y$, para todo $x, y \in P$.

As relações de ordem dividem-se em dois tipos: totais e parciais. Considere o conjunto dos números reais e a relação de ordem *'maior ou igual'* $\geq$. Nesta relação, para quaisquer dois números reais x e y, ou $x \geq y$, ou $y \geq x$, ou ambas as situações são verdadeiras (quando x e y são iguais). Esta relação é chamada de **ordem total**, já que todos os elementos do conjunto (números reais) são comparáveis entre si de acordo com a relação. Entretanto, essa situação não ocorre universalmente. Se considerada a relação de descendência estabelecida entre as pessoas, obviamente, existem duas pessoas A e B em que nem A é descendente de B, nem B é descendente de A. Relações com essa característica são chamadas de **ordens parciais**, pois, nem todos elementos de um conjunto são comparáveis.

Uma relação de ordem $\leq$ estabelecida sobre um conjunto P recebe a notação $\langle P, \leq \rangle$. O conjunto P associado a uma relação de ordem $\leq$ é, em geral, chamado de conjunto ordenado.

Normalmente, quando se tem uma ordem definida sobre objetos, deseja-se saber quais objetos são os primeiros e quais são os últimos. Em uma ordem parcial, as idéias de primeiros e últimos são substituídas pelos conceitos de máximos e mínimos. Um elemento é **máximo** em uma relação de ordem, se não existem elementos maiores que ele. Analogamente, um elemento é **mínimo**, se não existem elementos menores que ele. Em geral, ordens parciais podem apresentar vários elementos máximos e mínimos ao invés de um único elemento. No entanto, existem situações em que se pode falar sobre o elemento máximo (mínimo) em uma ordem parcial. Quando um conjunto ordenado possui um único elemento máximo (mínimo), ele é chamado de **maior** (**menor**). Um elemento é o maior de um conjunto ordenado, se esse elemento for maior ou igual a todos os elementos do conjunto ordenado. Assim, o maior de um conjunto ordenado é sempre comparável a todos os outros elementos do conjunto. Similarmente, define-se o menor elemento. Essas definições são formalmente apresentadas na definição 1.

Definição 1. Seja $\langle P, \leq \rangle$ uma relação de ordem definida sobre o conjunto P. Sejam $Max \subseteq P$, $Min \subseteq P$, $x \in P$ e $y \in P$.

1. O conjunto Max é um conjunto de máximos de P se, e somente se, $Max = \{a \in P | \forall b \in P \; a \leq b \rightarrow a = b\}$.

2. O conjunto Min é um conjunto de mínimo de P se, e somente se, $Min = \{a \in P | \forall b \in P \; b \leq a \rightarrow a = b\}$.

3. O elemento x é o maior de P se, e somente se, $\forall b \in P \; b \leq x$.

4. O elemento y é o menor de P se, e somente se, $\forall b \in P \; y \leq b$.

Todo conjunto ordenado possui duas importantes famílias de conjuntos associados. Essas famílias são a **seção inferior** e a **seção superior**[1]. Seja P um conjunto ordenado. Um conjunto $X \subseteq P$ é uma seção inferior de P, se, para quaisquer elementos $a \in X$ e $b \in P$ tais que $b \leq a$, $b \in X$. Analogamente, $X \subseteq P$ é uma seção superior de P se, para quaisquer elementos $a \in X$ e $b \in P$ tais que $a \leq b$, $b \in X$. O conjunto das seções inferiores (superiores) de P é de denotado por $\mathcal{I}(P)$ ($\mathcal{S}(P)$). Os conjuntos $\mathcal{I}(P)$ e $\mathcal{S}(P)$ ordenados pela inclusão de conjuntos também formam ordens parciais.

Sejam $Q \subseteq P$ e $x \in P$. As seções inferiores (superiores) induzidas por Q e x são:

$$\downarrow Q = \{y \in P | \exists x \in Q \; y \leq x\} \quad \downarrow x = \{y \in P | y \leq x\} \tag{2.1}$$

$$\uparrow Q = \{y \in P | \exists x \in Q \; x \leq y\} \quad \uparrow x = \{y \in P | x \leq y\} \tag{2.2}$$

$\downarrow Q$, $\uparrow Q$, $\downarrow x$ e $\uparrow x$ são, respectivamente, as menores seções inferiores e superiores contendo Q e x. Esses conceitos são úteis em algumas ocasiões e, em especial, o conceito de seções inferiores será utilizado na seção 4.1.2 por algoritmos de extração de regras de associação.

Definição 2. Seja P um conjunto ordenado. Sejam $Q \subseteq P$ e $x \in P$. O elemento x é um **limite inferior** de Q, se todo elemento de Q é maior ou igual a x, ou seja, $\forall y \in Q \; x \leq y$. Da mesma forma, x é um **limite superior** de Q, se $\forall y \in Q \; y \leq x$.

Os conjuntos dos limites inferiores e superiores de Q é dado por:

$$Q^i = \{x \in P | \forall y \in Q \; x \leq y\} \tag{2.3}$$

$$Q^s = \{x \in P | \forall y \in Q \; y \leq x\} \tag{2.4}$$

Como as relações de ordem são transitivas, os conjuntos Q^i e Q^s são seções inferiores e superiores.

Se o conjunto Q^i possui um maior, então este elemento é chamado de **maior limite inferior** de Q ou, simplesmente, **ínfimo**. Como o elemento maior é único[2], se ele existir, então se Q possui um ínfimo, ele é único. Se o conjunto Q^s possui um menor, então ele é chamado de **menor limite superior** ou **supremo** de Q. Igualmente ao ínfimo, se o supremo existir, então ele é único. Em geral, adota-se a notação $\bigwedge Q$ e $\bigvee Q$ para falar do ínfimo e supremo de Q se eles existirem. Se $Q = \{x, y\}$, então adota-se a notação $x \wedge y$ e $x \vee y$ para ínfimo e supremo.

Definição 3. Seja P um conjunto ordenado.

1. Se $x \vee y$ e $x \wedge y$ existem para quaisquer $x, y \in P$, então P é chamado de **reticulado**.

[1] Respectivamente, em inglês *order ideal* e *order filter*.

[2] A prova dessa afirmação pode ser encontrada em [20].

2. Se $\bigvee S$ e $\bigwedge S$ existem para todo $S \subseteq P$, então P é chamado de **reticulado completo**.

Em um conjunto ordenado, quando existe apenas o supremo (ínfimo) para quaisquer elementos, esse conjunto ordenado é chamado de supremo(ínfimo)-semi-reticulado. Além disso, todo reticulado finito é um reticulado completo [20, corolário 2.25].

Na seção 2.3, reticulados completos de estruturas denominadas conceitos formais são utilizados para a análise de dados. Conceitos formais são pares de conjuntos fechados de objetos e atributos. Conjuntos fechados são um dos temas da próxima seção.

2.2 Sistemas de fechos

Outro importante conceito para a compreensão da AFC é o de **sistemas de fechos**. A idéia de sistemas de fechos é bastante simples apesar de não ser, em princípio, intuitiva. Isso deve-se ao fato do nível de abstração; sistemas de fechos são famílias de conjuntos com algumas propriedades.

Seja P um conjunto. Uma família $\mathcal{C} \subseteq \wp(P)$[3] é um sistema de fecho se:

$$\bigcap \mathcal{F} \in \mathcal{C} \text{ para todo } \mathcal{F} \subseteq \mathcal{C} \text{ e } P \in \mathcal{C}.$$

A família $\mathcal{C}$ é chamada de sistema de fecho já que ela é fechada sob interseções. Essas famílias são, também, conhecidas como estruturas de interseção com topo[4] [20].

Um sistema de fecho ordenado pela inclusão de conjuntos, $\langle \mathcal{C}, \subseteq \rangle$, é um reticulado completo em que o supremo e o ínfimo são determinados por:

$$\begin{aligned} \bigvee \mathcal{F} &= \bigcap \{A \in \mathcal{C} | \bigcup \mathcal{F} \subseteq A\} \\ \bigwedge \mathcal{F} &= \bigcap \mathcal{F} \end{aligned}$$

Note que o próprio conjunto potência, $\wp(P)$, é um sistema de fecho. Portanto, $\wp(P)$ ordenado pela inclusão de conjuntos é um reticulado completo.

Associado ao conceito de sistemas de fechos, encontra-se o conceito de **operadores de fecho**. Um operador de fecho é uma função que mapeia um elemento de um conjunto ao seu fecho.

Definição 4. Seja P um conjunto ordenado e $x, y \in P$, uma função $c : P \to P$ é um operador de fecho se:

[3] $\wp(P)$ é o conjunto potência de P, ou seja, $\wp(P) = \{X | X \subseteq P\}$.
[4] Em inglês *topped intersection structures*.

(i) $x \leq c(x)$;

(ii) $x \leq y \rightarrow c(x) \leq c(y)$;

(iii) $c(c(x)) = c(x)$;

As propriedades (i), (ii), (iii) da definição 4 são chamadas, respectivamente, de **monotonicidade**, **extensibilidade** e **idempotência**. Essas são propriedades necessárias e suficientes para que uma função seja um operador de fecho.

Um elemento $x \in P$ é chamado de fecho (ou fechado) se $x = c(x)$. O conjunto dos fechos de P é denotado por P_c. Esse conjunto revela a estreita relação entre operadores de fecho, sistemas de fechos e reticulados. Especificamente, o conjunto dos fechos é um sistema de fecho e, ordenado pela relação de inclusão, é um reticulado completo.

Davey e Priestley mostraram em [20] que de um operador de fecho obtém-se um reticulado e de um reticulado obtém-se um operador de fecho. Seja c um operador de fecho definido sobre um conjunto X. O conjunto $L_c = \{A \subseteq X | A = c(A)\}$ ordenado pela inclusão de conjuntos é um reticulado completo cujo ínfimo e supremo são obtidos por:

$$\begin{aligned} \bigwedge \mathcal{F} &= \bigcap \mathcal{F} \\ \bigvee \mathcal{F} &= c(\bigcup \mathcal{F}) \end{aligned}$$

Similarmente, para um reticulado completo L sobre X, a função $c_L : \wp(X) \rightarrow \wp(X)$, tal que para $A \subseteq X$,

$$c_L(A) = \bigcap \{B \in L | A \subseteq B\}$$

é um operador de fecho.

Definição 5. Sejam P e Q dois conjuntos ordenados. Sejam $^{\triangleright} : P \rightarrow Q$ e $^{\triangleleft} : Q \rightarrow P$ duas funções entre os conjuntos ordenados[5]. O par $(^{\triangleright}, ^{\triangleleft})$ é uma **conexão de Galois** se, para todo $p \in P$ e $q \in Q$,

$$p^{\triangleright} \leq q \iff p \leq q^{\triangleleft} \qquad \text{(Gal)}$$

Proposição 1 (Lema 7.26 [20])**.** *Seja* $(^{\triangleright}, ^{\triangleleft})$ *uma conexão de Galois entre conjuntos ordenados* P *e* Q*. Sejam* $p, p_1, p_2 \in P$ *e* $q, q_1, q_2 \in Q$*. Então,*

(Gal-i) $p \leq p^{\triangleright\triangleleft}$ *e* $q^{\triangleleft\triangleright} \leq q$*;*

(Gal-ii) $p_1 \leq p_2 \rightarrow p_1^{\triangleright} \leq p_2^{\triangleright}$ *e* $q_1 \leq q_2 \rightarrow q_1^{\triangleleft} \leq q_2^{\triangleleft}$*;*

[5] Leia-se direita e esquerda respectivamente para $^{\triangleright}$ e $^{\triangleleft}$.

(**Gal-iii**) $p^{\triangleright} = p^{\triangleright\triangleleft\triangleright}$ *e* $q^{\triangleleft} = q^{\triangleleft\triangleright\triangleleft}$.

Da mesma forma, um par de funções $^{\triangleright} : P \to Q$ *e* $^{\triangleleft} : Q \to P$ *que satisfizer as propriedades* (Gal-i) *e* (Gal-ii) *para todo* $p, p_1, p_2 \in P$ *e* $q, q_1, q_2 \in Q$ *forma uma conexão de Galois.*

Lema 1. *Sejam P e Q conjuntos ordenados. Seja* $(^{\triangleright}, ^{\triangleleft})$ *uma conexão de Galois sobre P e Q. Então, as funções* $\alpha : P \to P$, $\alpha = ^{\triangleright\triangleleft}$ *e* $\beta : Q \to Q$, $\beta = ^{\triangleleft\triangleright}$ *são operadores de fecho.*

Na próxima seção, serão definidas funções chamadas de operadores de derivação que formam uma conexão de Galois e, por isso, são úteis na construção de reticulados conceituais.

2.3 Introdução a análise formal de conceitos

A AFC baseia-se em três entidades: *contextos formais*, *conceitos formais* e *reticulados conceituais*.

Um conceito é determinado por sua extensão e intensão. A extensão refere-se ao conjunto de objetos que são instâncias do conceito, enquanto a intensão, ao conjunto de características comuns a todos os objetos [48]. Dada a dificuldade (ou, até mesmo, a impossibilidade) de listar-se todos os objetos e atributos de um conceito, é comum considerar a extensão e a intensão em um *contexto* específico, limitando-se o número de objetos e atributos.

Aproveitando a descrição informal de contextos, pode-se definir **contextos formais**, os quais, intuitivamente, delimitam o conjunto de objetos, o conjunto de atributos e as relações entre atributos e objetos para conceitos formais. Sendo assim, um contexto formal é matematicamente definido como uma tripla (G, M, I) na qual G é o conjunto de objetos considerados, M é o conjunto de atributos e $I \subseteq G \times M$ uma relação binária entre objetos e atributos, a relação de incidência, que determina se um objeto $g \in G$ possui um atributo $m \in M$.

Um contexto formal é, geralmente, representado por uma tabela cujas linhas representam objetos, colunas representam atributos, e interseções entre linhas e colunas representam a relação de incidência. A Tabela 1 exemplifica um contexto formal. Neste exemplo, objetos são animais, os atributos são suas características e a relação de incidência informa se um animal possui, ou não, determinada característica. Um animal possui uma característica se, e somente se, existe um "$\times$" na interseção entre a linha e a coluna respectiva.

Embora a tabela cruzada represente adequadamente a relação entre objetos e atributos, em situações reais, objetos são caracterizados por atributos que podem assumir diversos valores; por exemplo, uma pessoa pode ser caracterizada por, entre outros caracteres, sua idade e um carro por sua cor. Nestes casos, a simples presença ou ausência de atributos não caracteriza

Tabela 1: Exemplo de contexto formal

Animais	aquático	terrestre	resp. branquial	pulmões	pêlo	pena	mamífero	razão
peixe	×		×					
sapo	×	×	×	×				
homem		×		×	×		×	×
macaco		×		×	×		×	
coruja		×		×		×		
tubarão	×		×					

adequadamente os objetos, uma vez que cada um pode possuir determinado atributo com determinado valor. Estes atributos são chamados de **atributos multivalorados**. Um contexto formal cujos objetos possuem atributos multivalorados é chamado de **contexto formal multivalorado**. Formalmente, ele é definido por (G,M,V,I), em que G é um conjunto de objetos, M é um conjunto de atributos multivalorados, V é o conjunto de valores dos atributos e I é uma relação ternária, $I \subseteq G \times M \times V$, que obedece a seguinte restrição:

$$(g,m,v) \in I \ \wedge \ (g,m,w) \in I \rightarrow v = w$$

Logo, ao se escrever $(g,m,v) \in I$, diz-se que o objeto g possui o atributo m com valor v. Assim, atributos multivalorados podem ser entendidos como funções parciais de G para V, o que permite reescrever $(g,m,v) \in I$ em notação funcional: $m(g) = v$ [14].

Apesar da formalização de contextos formais multivalorados, eles não podem ser usados diretamente na AFC. Os contextos formais multivalorados devem ser transformados em contextos univalorados através do processo chamado de **escala conceitual**. A escala conceitual consiste em definir valores de atributos multivalorados a partir da combinação de valores univalorados. O exemplo 1 ilustra a definição de escala conceitual e o processo de transformação de contextos multivalorados em contextos univalorados. A escala adotada no exemplo é a escala plana. Naturalmente, ela não é a única, porém, é a mais simples delas. Ganter e Wille([31]) apresentam um estudo mais detalhado sobre os tipos de escalas e suas interpretações.

Exemplo 1. Seja a Tabela 2 um contexto multivalorado referente aos oito planetas do sistema solar mais Plutão e suas características. As características são o tamanho dos planetas, a distância do sol e a ocorrência de lua.

Como contextos multivalorados não podem ser usados diretamente na AFC, deve-se transformar o contexto dos planetas em um contexto univalorado. A transformação de contextos multivalorados em univalorados é feita através da escala conceitual. Como dito, a escala conceitual especifica a correspondência entre valores de atributos multivalorados e atributos univalorados. A Tabela 3 apresenta as escalas dos atributos. As escalas são representadas como

Tabela 2: Contexto Multivalorado de Planetas.

	tamanho	distância do sol	lua
Mercúrio	pequeno	perto	não
Vênus	pequeno	perto	não
Terra	pequeno	perto	sim
Marte	pequeno	perto	sim
Júpiter	grande	longe	sim
Saturno	grande	longe	sim
Urano	médio	longe	sim
Netuno	médio	longe	sim
Plutão	pequeno	longe	sim

Tabela 3: Escala conceitual dos atributo *tamanho*, *distância* e *lua*.

$\mathbb{S}_{tamanho}$	pequeno	médio	grande
pequeno	×		
médio		×	
grande			×

$\mathbb{S}_{distancia}$	perto	longe
perto	×	
longe		×

$\mathbb{S}_{lua}$	lua_sim	lua_não
sim	×	
não		×

contextos formais. Os objetos, nas escalas, são os valores dos atributos multivalorados e os atributos são os atributos a serem considerados no contexto univalorado. Então, por exemplo, o atributo tamanho no contexto multivalorado corresponderá aos atributos *pequeno*, *médio* e *grande* no contexto multivalorado. Ainda considerando o exemplo, quando um objeto possui o valor *pequeno* para o atributo *tamanho* no contexto multivalorado, ele possuirá o atributo *pequeno* no contexto univalorado. Aplicando-se o mesmo raciocínio para os outros atributos, obtém-se o contexto formal da Tabela 4.

Tabela 4: Contexto Formal de Planetas.

Planetas	pequeno	médio	grande	perto	longe	lua_sim	lua_não
Mercúrio	×			×			×
Vênus	×			×			×
Terra	×			×		×	
Marte	×			×		×	
Júpiter			×		×	×	
Saturno			×		×	×	
Urano		×			×	×	
Netuno		×			×	×	
Plutão	×				×	×	

Seja (G,M,I) um contexto formal, $A \subseteq G$ um subconjunto de objetos e $B \subseteq M$ um subconjunto de atributos. Pode ser necessário conhecer o conjunto de atributos comuns aos objetos de A, ou, de forma análoga, pode ser necessário conhecer o conjunto de objetos que possuem

os atributos de B em comum. Estas necessidades são atendidas através dos **operadores de derivação** definidos por:

$$A^{\uparrow} \leftarrow \{m \in M | \forall g \in A,\ (g,m) \in I\} \tag{2.6}$$

$$B^{\downarrow} \leftarrow \{g \in G | \forall m \in B,\ (g,m) \in I\} \tag{2.7}$$

Os operadores de derivação $(\cdot)^{\uparrow}$ e $(\cdot)^{\downarrow}$, em geral, recebem a mesma notação $(\cdot)'$ como forma de simplificação.

Se $A \subseteq G$, então A' é um conjunto de atributos comuns aos objetos de A. A A' pode-se aplicar, novamente, o operador de derivação e obter A'' resultando, novamente, em um conjunto de objetos. Intuitivamente, para $A \subseteq G$, A'' retorna o conjunto de todos os objetos que possuem, em comum, os atributos dos objetos de A; note que $A \subseteq A''$. O operador é definido de forma similar para o conjunto de atributos. Se $B \subseteq M$, então B' retorna o conjunto de objetos que possuem os atributos de B em comum. Assim, B'' retorna o conjunto de atributos comuns a todos objetos que possuem os atributos de B em comum; conseqüentemente, $B \subseteq B''$. Formalmente, o operador $(\cdot)''$ refere-se à composição dos operadores de derivação, ou seja, $\downarrow \circ \uparrow$ para $A \subseteq G$, ou $\uparrow \circ \downarrow$ para $B \subseteq M$[6]. Com isso, pode-se constatar as seguintes propriedades para os operadores de derivação e suas respectivas composições:

Proposição 2. *Sejam $A, A_1, A_2 \subseteq G$; os operadores de derivação satisfazem as seguintes propriedades:*

1 $A \subseteq A''$,

2 $A_1 \subseteq A_2 \rightarrow A_2' \subseteq A_1'$,

3 $A' = A'''$

De forma similar, para $B, B_1, B_2 \subseteq M$ tem-se:

1 $B \subseteq B''$,

2 $B_1 \subseteq B_2 \rightarrow B_2' \subseteq B_1'$,

3 $B' = B'''$

Observe que o par $(',')$ de operadores de derivação constitue uma conexão de Galois entre os conjuntos potência $\wp(G)$ e $\wp(M)$. No entanto, assume-se para $\wp(M)$ a ordem inversa à de inclusão, $\supseteq$. Dessa forma, a composição dos operadores de derivação são operadores de fecho e, assim, dão origem a um reticulado.

[6] Supõe-se que $f \circ g(x) = f(g(x))$.

A partir de um contexto formal, algoritmos podem ser aplicados a fim de obter-se um conjunto de **conceitos formais**, os quais são simplificações daquilo que se entende como conceitos. Com esta finalidade, um conceito formal busca identificar o conjunto de atributos que delimitam e caracterizam um objeto (intensão do conceito), assim como os objetos que compartilham tais atributos (extensão do conceito). Matematicamente, estas entidades são pares ordenados (A,B) em que A e B são respectivamente subconjuntos do conjunto de objetos e do conjunto de atributos. O par (A,B) é um conceito formal se, e somente se, $A = B'$ e $B = A'$. Assim, por exemplo, o par ({homem, macaco}, {terrestre, pulmões, pêlo, mamífero}) é um conceito formal derivado do contexto da Tabela 1. O conjunto de conceitos formais de um contexto (G,M,I) é denotado por $\mathfrak{B}(G,M,I)$.

Conceitos formais podem ser ordenados por uma hierarquia criando-se a relação *subconceito-superconceito*. Essa hierarquia determina que para dois conceitos formais, (A_1,B_1) e (A_2,B_2), $(A_1,B_1) \leq (A_2,B_2)$, ou seja, (A_1,B_1) é um subconceito de (A_2,B_2), ou (A_2,B_2) é um superconceito de (A_1,B_1) se, e somente se, $A_1 \subseteq A_2$ (ou $B_1 \supseteq B_2$) [31]. Como já dito, os operadores de derivação formam uma conexão de Galois e suas composições formam um operador de fecho. Assim, o conjunto das intensões e extensões são sistemas de fechos e, ordenados pela ordem direta e inversa da inclusão de conjuntos, formam reticulados completos. Sendo assim, o conjunto de conceitos formais associados à ordem estabelecida acima é um reticulado completo chamado de **reticulado conceitual**. Como os conceitos formais são obtidos dos operadores de derivação que formam uma conexão de Galois, os reticulados conceituais também são conhecidos como reticulados de Galois. O ínfimo e o supremo do reticulado conceitual são obtidos tal como apresentado pelo teorema 1.

Teorema 1. *Seja (G,M,I) um contexto formal. Então, $\langle \mathfrak{B}(G,M,I), \leq \rangle$ é um reticulado completo tal que, para todo $\mathcal{C} \subseteq \mathfrak{B}(G,M,I)$, supremos e ínfimos são obtidos por*

$$\bigvee \mathcal{C} = \left(\left(\bigcup \{A | (A,B) \in \mathcal{C}\} \right)'', \bigcap \{B | (A,B) \in \mathcal{C}\} \right)$$
$$\bigwedge \mathcal{C} = \left(\bigcap \{A | (A,B) \in \mathcal{C}\}, \left(\bigcup \{B | (A,B) \in \mathcal{C}\} \right)'' \right)$$

O Teorema 1 é conhecido como o teorema fundamental da AFC e sua prova pode ser obtida em [20, 31, 57].

O reticulado conceitual pode ser representado através de um *diagrama de linha*. Para entender como a hierarquia é representada pelo diagrama, é necessário compreender o conceito de relação de cobertura. Um conceito (A_1,B_1) é coberto por outro conceito (A_2,B_2), escreve-se $(A_1,B_1) \prec (A_2,B_2)$, se

$$(A_1,B_1) < (A_2,B_2) \wedge (A_1,B_1) \leq (A_3,B_3) < (A_2,B_2) \rightarrow (A_1,B_1) = (A_3,B_3),$$

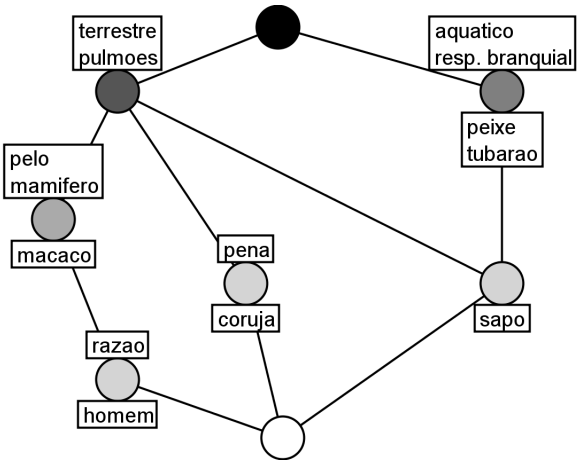

Figura 1: Exemplo de Diagrama de Linha

ou seja, (A_1, B_1) é o vizinho imediatamente inferior ao conceito (A_2, B_2) na hierarquia de conceitos. O diagrama de linha exprime a hierarquia de conceitos representando-os como círculos cuja extensão encontra-se desenhada abaixo e cuja intensão encontra-se desenhada acima, e a relação de cobertura como linhas interconectando conceitos. A extensão e a intensão dos conceitos formais são exibidas de forma *reduzida*. Neste caso, um objeto g somente aparece no rótulo da extensão do menor conceito (X_1, Y_1) tal que $g \in X_1$. O conceito formal (X_1, Y_1) é chamado de conceito-objeto do objeto g. Ele é obtido através da equação (2.8) descrita a seguir. De forma semelhante, um atributo m somente é expresso no rótulo da intensão do maior conceito (X_2, Y_2) tal que $m \in Y_2$. O conceito formal (X_2, Y_2) é chamado de conceito-atributo do atributo m e ele é obtido pela equação (2.9) descrita a seguir. A Figura 1 exibe o diagrama de linha dos conceitos obtidos do contexto da Tabela 1.

$$\gamma(g) = \bigwedge \{(A,B) \in \mathfrak{B}(G,M,I) | g \in A\} \tag{2.8}$$

$$\mu(m) = \bigvee \{(A,B) \in \mathfrak{B}(G,M,I) | m \in B\} \tag{2.9}$$

O diagrama de linha permite a visualização dos relacionamentos entre conceitos. Observando o diagrama da Figura 1, percebe-se que todo animal possuindo razão, também possui pêlo e é mamífero. Estas relações podem ser expressas através de regras do tipo *se-então*; para o exemplo anterior, a relação é expressa por: *se* razão *então* pêlo e mamífero. Na Figura 1, nota-se, ainda, que *se* terrestre *então* pulmões e *se* pulmões *então* terrestre, isto é, terrestre se, e somente se, pulmões. Observe ainda que o único animal terrestre e aquático é o sapo. Atribuindo-se o nome "anfíbio" ao terreste e aquático com rótulo sapo na Figura 1, pode-se ainda enunciar a regra: *se* terrestre *e* aquático *então* anfíbio. A extração de regras é o tema dos dois próximos capítulos.

3 *Regras Determinísticas*

Este capítulo é dedicado à apresentação e comparação de métodos para a extração de regras determinísticas de bancos de dados. O capítulo introdutório apresentou, brevemente, o conceito de regras determinísticas. Essas são regras válidas para todos elementos em um banco de dados. Elas estão divididas em dois grupos, o das dependências funcionais e o das implicações.

Os tópicos relacionados às dependências funcionais são discutidos na seção 3.1. Inicialmente, é apresentada a relação de contextos formais multivalorados e bancos de dados relacionais. Em seguida, são definidas, formalmente, dependências funcionais e, depois, apresentada sua utilidade. Os algoritmos baseados na AFC para a identificação de dependências funcionais foram propostos para a identificação de implicações. Entretanto, eles podem ser utilizados com a finalidade de identificação de dependências funcionais, transformando-se o contexto formal multivalorado em um contexto formal univalorado. Após tal transformação, as implicações identificadas do contexto transformado são, justamente, as dependências funcionais do contexto original. Esta transformação é apresentada na seção 3.1.4. Alternativamente, pode-se utilizar novos operadores de derivação, denominados operadores de particionamento, que são definidos na seção 3.1.5. As dependências funcionais, assim como as implicações, permitem inferência sobre conjuntos de regras. Dado um conjunto de regras determinísticas, pode-se utilizar certos axiomas para inferir se uma outra regra é dedutível, ou não, daquele conjunto. Estes axiomas estão apresentados na seção 3.2.

Outros tópicos abordados na seção 3.2 são a completeza, a equivalência e redundância em conjuntos de regras. Estes tópicos são importantes já que permitem que conjuntos de regras com determinadas características, denominados coberturas, possam ser encontrados. O conceito de cobertura de regras é o assunto da seção 3.3.

Na seção 3.4, as implicações são apresentadas formalmente. Naquela seção, além da definição formal de implicações, é, também, discutida a praticidade dessas regras. Apresentadas as definições acerca das regras determinísticas, os algoritmos para identificá-las são apresentados. São apresentados quatro algoritmos: *Next Closure*, *Find Implications*, *Impec*, *ApremIR*. Cada um destes algoritmos produz uma cobertura de regras diferente. Ao apresentá-los, além de

descrevê-los e mencionar o tipo de cobertura que eles geram, serão discutidas as complexidades de tempo e espaço de cada um. Finalmente, na seção 3.6, o desempenho dos algoritmos com bancos de dados reais e sintéticos é discutido.

3.1 Dependências Funcionais

Entre os objetivos principais para a utilização de bancos de dados, destacam-se dois: o fato destes evitarem redundância de dados; e o de aumentarem a confiança nos dados armazenados através da aplicação de diversos tipos de restrições. Dependências de dados desempenham um papel importante para se atingir um dos objetivos citados acima, aquele relativo à redundância de dados. Nesta seção, será explorado um tipo específico de dependência de dados: dependências funcionais. Estas são extremamente úteis para a normalização de modelos lógicos [24].

Inicialmente, será discutido o conceito de relações em bancos de dados e mostrada a equivalência destas com contextos formais multivalorados. Apresentado o conceito de relações, será definido o conceito de dependência funcional, o qual foi, informalmente, definido no capítulo introdutório.

3.1.1 Relações e Esquemas Relacionais no Modelo Relacional

O modelo de bancos de dados relacionais proposto por Codd [16] utilizava relações sobre conjuntos de atributos para descrever conexões entre dados [8]. Esta representação pode ser vista como uma tabela cujas linhas representam as instâncias e as colunas os atributos. Cada atributo possui um conjunto de possíveis valores relacionados aos quais se dá o nome de *domínio do atributo*. Assim, para cada par *(instância,atributo)*, existe um valor correspondente no domínio do atributo. Portanto, cada linha da tabela corresponde ao conjunto dos pares (instância,valor do atributo) para cada atributo da relação. Esse conjunto, em geral, é chamado de tupla.

Embora o banco de dados seja constantemente modificado com a inclusão, remoção e modificação de dados, a estrutura do mesmo é dificilmente alterada. Em outras palavras, o nome da relação assim como o conjunto de atributos quase nunca é alterado. Assim, confunde-se, muitas vezes na literatura, o conceito da relação em si (i.e. como conjunto de tuplas e atributos) com o de sua estrutura. Neste trabalho, no entanto, assume-se a notação adotada por Beeri [8], ou seja, o termo *relação* é utilizado para designar o conjunto de tuplas e *esquema relacional* é utilizado para referir-se à estrutura da relação (conjunto de atributos da relação).

Normalmente, um banco de dados é composto por diversos esquemas relacionais, cada um

com sua própria relação. Entretanto, para verificar os relacionamentos existente entre atributos do banco de dados, é conveniente imaginar o banco de dados com um único esquema relacional "universal" com uma única relação "universal" [8, 54]. Seja A o conjunto de atributos de todos os esquemas relacionais do banco de dados D e $V(a)$ o domínio do atributo $a \in A$. A relação "universal" R para o banco de dados D é definida por:

$$R \subseteq \bigtimes_{a \in A} V(a)$$

Pode-se interpretar um banco de dados como um contexto formal multivalorado. Retomando a definição de contexto formal multivalorado apresentada no capítulo 2.3, um contexto multivalorado C é uma quadrupla (G, M, V, I). Seja D um banco de dados cujo esquema relacional "universal" seja A e cuja relação "universal" seja R. Seja O um conjunto de identificadores para as instâncias de D. Seja $V = \bigcup\{V(a) | a \in A\}$ a união dos domínios dos atributos de A. Definindo-se a relação ternária $I \subseteq O \times A \times V$ tal que toda tripla (o, a, v) pertence a I se, e somente se, o valor do atributo a para a tupla o em R é v. Seja $f : O \times A \to V$ uma função que dada uma tupla da relação e um atributo retorna o valor do atributo para a tupla. A relação I é formalmente definida como $I \subseteq O \times A \times V$ tal que $(o, a, v) \in I \iff f(o, a) = v$. Desta forma, $C = (O, A, V, I)$ é um contexto formal multivalorado equivalente ao banco de dados D.

Apresentada a equivalência entre os conceitos de relações, esquemas relacionais e contextos formais multivalorados, a seção seguinte apresentará a definição de dependências funcionais. Embora elas sejam definidas sob o ponto de vista da Teoria de Bancos de Dados Relacionais, esta definição pode ser adequada à AFC.

3.1.2 Definindo dependência funcional

Uma dependência funcional entre atributos de um esquema relacional A é um par (X, Y), $X, Y \subseteq A$, em que os atributos de X determinam funcionalmente os atributos de Y, ou seja, para quaisquer duas tuplas, t_1 e t_2, de uma instância do esquema, se t_1 possui os mesmos valores que t_2 para os atributos de X, então, elas terão os mesmos valores para os atributos de Y. Usualmente, assume-se a notação $X \to Y$ para a dependência funcional (X, Y).

Seja D um banco de dados com relação universal R, esquema universal A cujo domínio dos atributos é V e sejam O um conjunto de identificadores das tuplas de R e $f : O \times \wp(A) \to \wp(V)$[1] uma função que, dados uma tupla t e um subconjunto de atributos X, retorna o conjunto de valores dos atributos de X para a tupla t. Uma dependência funcional $X \to Y$ é dita ser válida para D se, e somente se, para toda instância r de D, para toda tupla de r, t_1 e t_2, $f(t_1, X) =$

[1] Repare que f é uma generalização da função apresentada na seção 3.1.1.

$f(t_2,X) \rightarrow f(t_1,Y) = f(t_2,Y)$.

A definição de dependência funcional pode ser reformulada para o contexto da AFC bastando, para tal, considerar um contexto formal multivalorado como um banco de dados na Teoria de Bancos de Dados Relacionais tal como mostrado na seção 3.1.1. Assim, seja $C = (G,M,V,I)$ um contexto formal multivalorado. Uma dependência funcional é um par (X,Y) em que $X,Y \subseteq M$; e ela é dita ser válida para C se, e somente se, $\forall g_1,g_2 \in G\ f(g_1,X) = f(g_2,X) \rightarrow f(g_1,Y) = f(g_2,Y)$, $f : G \times \wp(M) \rightarrow \wp(V)$.

3.1.3 Qual a utilidade de uma dependência funcional?

Anteriormente, foi mencionada a utilidade das dependências funcionais sob o ponto de vista prático. A importância fundamental da descoberta de dependência de dados está na normalização de modelos lógicos [18, 24, 32, 54]. De maneira geral, a normalização consiste na adequação de modelos lógicos de forma a evitar redundância na armazenagem dos dados. Outro papel de destaque sobre a utilidade de dependências funcionas é a descoberta de atributos chave, atributos que identifiquem tuplas de uma relação.

O primeiro ponto a ser explorado será a utilização de dependências funcionais para descoberta de atributos chave. Seja $A = \{a_1, a_2, \ldots, a_n\}$ um conjunto com n atributos[2]. Um conjunto de atributos X é uma chave para A, se, o conjunto de dependências $\{X \rightarrow a_1, X \rightarrow a_2, \ldots, X \rightarrow a_n\}$[3] é válido. O leitor poderá perceber, ao ler as seções 3.2 e 3.3, que o conjunto de atributos chave pode ser obtido diretamente de uma cobertura canônica completa, utilizando-se os axiomas de Armstrong (seção 3.2).

Em relação à normalização de modelos lógicos, as dependências funcionais são utilizadas para a adequação de tais modelos às formas normais de Boyce-Codd(BCNF) e Terceira Forma Normal(3NF) [32, 54].

Um esquema encontra-se na BCNF se toda dependência válida é da forma $X \rightarrow a$, $a \notin X$ e X contém apenas atributos chave. Informalmente, as únicas dependências funcionais válidas nas instâncias de um esquema na BCNF são aquelas referentes aos atributos chave, ou seja, aquelas em que os atributos chave determinam funcionalmente os demais atributos.

A 3NF é menos restritiva que BCNF. Neste caso, uma relação R está na 3NF, se, para toda dependência $X \rightarrow Y$ válida em R, $X \cap K \neq \emptyset$ ou $Y \subseteq K$, sendo K o conjunto de atributos chave de R. Observe que a diferença entre a 3NF e a BCNF está no fato da primeira permitir dependências

[2]Neste ponto, tem maior significado explorar atributos chave sob o ponto de vista da Teoria de Bancos de Dados Relacionais, porém, como já mencionado, tal relação pode ser vista como um contexto multivalorado.

[3]A dependência $X \rightarrow a_i$ é uma abreviação para $X \rightarrow \{a_i\}$.

do tipo $X \rightarrow Y, Y \subseteq K$. Este tipo de dependência pode permitir que ocorra redundância no armazenamento. Entretanto, em certas ocasiões não é possível adequar uma relação à BCNF; neste caso, a 3NF torna-se bastante útil.

De maneira geral, essas três situações indicam que a qualidade de um projeto de banco de dados está ligada à análise de coberturas de dependências funcionais válidas. A escolha do conjunto de regras a ser identificado em uma base de dados depende do processo de análise que será aplicado. Certamente, para os casos em que a análise será feita por um projetista, a escolha da cobertura canônica (ver seção 3.3.3) pode ser mais adequada, pois a análise da mesma, para verificação das situações apontadas anteriormente, pode se dar, até mesmo, de forma direta, sem a necessidade da aplicação dos axiomas de Armstrong. Entretanto, como o número de dependências na cobertura canônica não é o menor possível, a escolha da cobertura mínima (ver seção 3.3.5) pode tornar-se mais interessante, caso a análise seja automatizada.

Como a escolha da cobertura mais adequada é dependente do uso destinado à tal, serão apresentados algoritmos para a identificação de diversos tipos de coberturas. A definição de cobertura, os principais tipos de coberturas e os algoritmos para identificá-las são temas das seções 3.3 e 3.5.

3.1.4 Transformação de contextos

Um dos caminhos para a identificação de dependências funcionais em bases de dados através da AFC é a transformação de contextos formais multivalorados em contextos univalorados. Através desta transformação, o problema de identificar dependências é mapeado para o de identificar implicações.

Seja $C = (G, M, V, I)$ um contexto multivalorado. Existe um contexto $C_T = (G_T, M, I_T)$ tal que o conjunto de implicações válidas em C_T é o conjunto de dependências funcionais válidas em C. O conjunto de objetos G_T é o conjunto de todos os pares não ordenados de objetos em G, assim, $|G_T| = \binom{|G|}{2}$; o conjunto de atributos é o mesmo de C; e $(g_1, g_2) I_T m \iff m(g_1) = m(g_2)$, para todo $(g_1, g_2) \in G_T$ e $m \in M$.

Observe que, pela definição de dependência funcional, uma dependência $X \rightarrow Y$ válida em C implica que $\forall g_1, g_2 \in G(\forall x \in X(x(g_1) = x(g_2)) \rightarrow \forall y \in Y(y(g_1) = y(g_2)))$. Portanto, uma implicação válida em C_T é equivalente a uma dependência funcional em C, pois se a implicação $w \rightarrow z$, para $w, z \in M$, é válida em C_T então $\forall (g_1, g_2) \in G_T[(g_1, g_2) I_T w \rightarrow (g_1, g_2) I_T z]$, o que é equivalente a definição de dependências funcionais em C.

Assim, foi mostrada a equivalência entre o problema da identificação de dependências funcionais em contextos multivalorados e o da identificação de implicações em contextos univalo-

rados. Entretanto, esta não é a única forma de transformar o problema de descoberta de dependências funcionais no problema da descoberta de implicações. A próxima seção apresenta uma alternativa para a transformação de contextos.

3.1.5 Operadores de particionamento

Os algoritmos baseados em AFC para a extração de regras, em geral, utilizam o operador de fecho definido sobre o conjunto de atributos M obtido através da composição dos operadores de derivação. Além disso, para a identificação de dependências funcionais, é necessária a transformação do contexto multivalorado em um contexto univalorado. Esta transformação, apesar de possível, pode ser ineficiente. Tal ineficiência encontra-se no fato que o espaço necessário para o armazenamento do contexto transformado é proporcional a $O(|G|^2)$. Assim, para situações em que o número de tuplas ($|G|$) é grande, torna-se necessário o armazenamento do contexto transformado em memória secundária para a execução dos algoritmos. Esta situação acarreta o aumento do tempo necessário para a identificação das regras. Alternativamente, pode-se definir novos operadores de derivação que possibilitem a identificação de dependências funcionais de contextos multivalorados sem a necessidade de sua transformação. A esses operadores dá-se o nome de operadores de particionamento.

Os operadores de particionamento foram apresentados por Jaume Baixeries [5]. Como supramencionado, estes operadores foram propostos como alternativa à transformação de contextos multivalorados em contextos univalorados. Ao observar a definição de dependência funcional apresentada na seção 3.1.2, constata-se que subconjuntos de atributos de M induzem partições dos objetos de G em classes de equivalência. Para $A \subseteq M$, os objetos de cada classe induzida por A possuem os mesmos valores para os atributos de A. Formalmente, seja P a partição dos objetos de G induzida pelos atributos de A, $\forall C \in P[\forall g_1, g_2 \in C(\forall a \in A[\, a(g_1) = a(g_2)\,])]$. Similarmente, dada uma partição P, existe um conjunto máximo de atributos que leva a tal partição. Assim, os operadores de particionamento são definidos da seguinte forma:

$$\varphi : \wp(M) \rightarrow \Pi_{\wp(M)} \tag{3.1}$$

$$\psi : \Pi_{\wp(M)} \rightarrow \wp(M) \tag{3.2}$$

O conjunto Π_A, $A \subseteq M$, refere-se à partição dos objetos induzida pelos atributos de A. Mais precisamente, $\Pi_A = \{C \subseteq G | \forall g_1, g_2 \in C, \forall a \in A[a(g_1) = a(g_2)]\}$. Deve-se ressaltar que $\forall C_i, C_j \in \Pi_A, C_i \cap C_j = \emptyset$ e que $\bigcup C_i = G$.

Definindo-se uma ordem parcial para os conjuntos $\wp(M)$ e $\Pi_{\wp(M)}$, o par de operadores (φ, ψ) forma uma conexão de Galois. Para $\wp(M)$, define-se $\langle \wp(M), \subseteq \rangle$. Para o conjunto das partições, define-se $\langle \Pi_{\wp(M)}, \leq \rangle$, tal que, para $\Pi_A, \Pi_B \in \Pi_{\wp(M)}$, $\Pi_A \leq \Pi_B \iff |\Pi_A| \geq |\Pi_B|$.

Então, como demonstrado por Baixeries [5], $(\varphi, \subseteq)$ e $(\psi, \leq)$ formam uma conexão de Galois. Portanto, $\gamma : \psi \circ \varphi$ e $\Gamma : \varphi \circ \psi$ definem operadores de fecho sobre M, podendo, assim, ser utilizados nos algoritmos para extração de dependências funcionais.

Note que uma dependência $A \rightarrow B$ é válida para o contexto (G,M,V,I) se, e somente se, $\Pi_A = \Pi_{A \cup B}$. Huhtala e outros [34] provaram esta afirmação. Entretanto, intuitivamente, pela definição de dependências funcionais, $A \rightarrow B$ é válida se, e somente se, $\forall g_1, g_2 \in G[\forall a \in A, \forall b \in B\ [a(g_1) = a(g_2) \rightarrow b(g_1) = b(g_2)]]$, e assim, percebe-se que $A \rightarrow B$ é válida se g_1 e g_2 pertencem à mesma classe nas partições induzidas por A e por $A \cup B$.

Embora seja possível utilizar o operador de fecho γ como alternativa para a transformação de contextos na descoberta de dependências, esta possibilidade não é tão direta. Em geral, os algoritmos para a obtenção de reticulados conceituais são baseados nos operadores de derivação definidos no capítulo 2. Assim, para a construção do reticulado com base nos operadores φ e ψ, é necessária a adaptação dos algoritmos já existentes ou a elaboração de novos.

3.2 Inferência sobre conjuntos de regras determinísticas

Outro fator que aproxima as dependências funcionais das implicações é o fato de ambas permitirem inferência sobre conjuntos de regras. Seja $C = (G,M,I)$ um contexto formal, e F um conjunto de regras[4] válidas em C, tal que $F = \{A \rightarrow B, B \rightarrow C\}$; pode-se indagar se a regra $A \rightarrow C$ também é válida em C. Como será visto, de acordo com os axiomas de Armstrong, pela regra da transitividade, se F é válido em C, então, $A \rightarrow C$ também é válida. Neste caso, se diz que F implica logicamente $A \rightarrow C$, ou $F \models A \rightarrow C$.

Dado um contexto C, torna-se impraticável, embora possível, a descoberta do conjunto de todas as regras válidas em C. Todavia, pode-se obter um conjunto menor a partir do qual toda regra válida seja implicada logicamente. Assim, um conjunto F de regras é dito ser **completo**, se toda regra válida em um contexto for implicada logicamente de F. Seja $F^+ = \{X \rightarrow Y \,|\, F \models X \rightarrow Y\}$ o conjunto das regras que sejam implicadas logicamente de F e seja T o conjunto de todas as regras válidas. F é completo se, e somente se, $F^+ = T$. De forma semelhante, pode-se afirmar que dois conjuntos de regras F e G são **equivalentes**, se, e somente se, $F^+ = G^+$. A partir da definição de equivalência de conjuntos de regras, pode-se definir o conceito de redundância de um conjunto: um conjunto F é dito ser **redundante**, se existe uma regra $X \rightarrow Y \in F$, tal que $F^+ = (F - \{X \rightarrow Y\})^+$.

O conjunto de regras F^+ pode ser inferido através da aplicação de um conjunto de axiomas

[4]Neste contexto, o termo regra está sendo usado tanto no sentido de implicação quanto de dependência funcional.

chamados axiomas de Armstrong. Este conjunto de axiomas foi inicialmente proposto por Armstrong [4] o qual, em seguida, foi aprimorado por Beeri, Fagin e Howard [9].

Proposição 3 (Axiomas de Armstrong)**.** *O conjunto de axiomas é formado por 6 regras:*

Reflexividade *Se* $Y \subseteq X \subseteq M$ *então* $X \rightarrow Y$ *é sempre válida,*

Aumento *Se* $X \rightarrow Y$ *é válida, e* $Z \subseteq M$*, então,* $XZ \rightarrow YZ$ *também é válida,*

Transitividade $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$,

União $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$,

Pseudotransitividade $\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$

Decomposição $\{X \rightarrow Y\} \models X \rightarrow Z$, $\forall Z \subseteq Y$.

□

É importante ressaltar que esse conjunto de axiomas é completo, no sentido que toda regra $X \rightarrow Y \in F^+$ pode ser obtida aplicando-se os axiomas; e correto[5], no sentido que toda regra obtida através da aplicação dos axiomas é válida. Provas e maiores detalhes desses axiomas podem ser obtidos em [37, 54].

3.3 Coberturas

O conceito de equivalência entre conjuntos de regras foi apresentado na seção anterior. A partir deste conceito, pode-se definir o conceito de cobertura. Sejam F e G dois conjuntos de regras válidas em um contexto. Se F é equivalente a G, então F é dita ser uma cobertura de G [37][Definição 5.1].

Neste ponto, não mencionou-se a relação entre o número de regras em F e em G, porém, mais à frente, serão apresentadas coberturas para as quais impõe-se restrições sobre o número de regras em F e em G. Especificamente, serão apresentadas coberturas para as quais o número de regras em F não é maior que o número de regras em G.

3.3.1 Coberturas não-redundantes

O conceito de redundância foi apresentado na seção 3.2. Relembrando a definição apresentada em tal seção, um conjunto F é dito ser redundante, se existe $X \rightarrow Y$ tal que $(F - \{X \rightarrow$

[5]Em inglês, *sound*.

$Y\})^+ = F^+$. Assim, uma cobertura F é dita ser não-redundante, se F é uma cobertura e não existe $X \rightarrow Y$ tal que $(F - \{X \rightarrow Y\})^+ = F^+$.

A definição de não-redundância sugere que não existem regras "descartáveis" em uma cobertura F. Entretanto, pode-se simplificar F, removendo-se atributos de suas regras. Este tipo de cobertura será apresentado na próxima seção.

3.3.2 Coberturas reduzidas

Antes de apresentar o conceito de coberturas reduzidas, faz-se necessária a definição de redução. Uma dependência $X \rightarrow Y \in F$ é dita ser **reduzida à esquerda**, se, para $X = aZ$ [6], tal que $a \in M$ e $Z \subset X$, $(F - \{X \rightarrow Y\}) \cup \{Z \rightarrow Y\}$ não é equivalente a F. De maneira análoga, uma regra $X \rightarrow Y \in F$ é **reduzida à direita**, se, para $Y = aW$, tal que $a \in M$ e $W \subset Y$, $(F - \{X \rightarrow Y\}) \cup \{X \rightarrow W\}$ não é equivalente a F.

Apresentado os conceitos de redução à esquerda e à direita, pode-se definir coberturas reduzidas. Uma cobertura F é uma **cobertura reduzida**, se F é uma cobertura e toda regra $X \rightarrow Y \in F$ é reduzida à esquerda, à direita e $Y \neq \emptyset$. Caso toda regra $X \rightarrow Y \in F$ seja reduzida apenas à esquerda, então F é dita ser uma **cobertura reduzida à esquerda**; o mesmo ocorre caso toda regra $X \rightarrow Y \in F$ seja reduzida apenas à direita, e $Y \neq \emptyset$, assim F é dita ser uma **cobertura reduzida à direita**.

3.3.3 Coberturas canônicas

O conceito de coberturas canônicas surge imediatamente dos conceitos de coberturas não-redundantes e coberturas reduzidas. Uma cobertura F é dita ser **canônica**, se toda regra de F é da forma $X \rightarrow a$ [7], $a \in M$, e F é não-redundante. Observe que a regra $X \rightarrow a$ é, também, reduzida à direita. Portanto, se uma cobertura é canônica, ela é reduzida e não-redundante.

O leitor mais atento pode observar que, para toda cobertura F reduzida e não-redundante, existe uma cobertura canônica G tal que $F^+ = G^+$. Como F é reduzida e não-redundante, ela pode ser transformada em uma cobertura canônica substituindo toda regra do tipo $X \rightarrow Y$, $Y = a_1 a_2 \ldots a_n$ pelo conjunto de regras $\{X \rightarrow a_1, X \rightarrow a_2 \ldots, X \rightarrow a_n\}$ utilizando-se o axioma da decomposição. Observe que F é reduzida, portanto, a substituição de $X \rightarrow Y$ por $\{X \rightarrow a_1, X \rightarrow a_2, \ldots, X \rightarrow a_n\}$ não torna a cobertura redundante. Inversamente, a partir de uma cobertura canônica G, pode-se obter uma cobertura F não-redundante e reduzida, utilizando-se o axioma da união.

[6] Utilizou-se, aqui, uma notação simplificada para expressar que $X = \{a\} \cup Z$.
[7] $X \rightarrow a$ é uma abreviação para $X \rightarrow \{a\}$.

Pode-se relaxar as restrições impostas às coberturas canônicas de forma a obter o conceito de coberturas próprias. Este assunto será tratado na próxima seção.

3.3.4 Coberturas próprias

Taouil e Bastide [52] propuseram esta cobertura e definiram-na como: uma cobertura é **própria**, se toda regra é reduzida à esquerda e o lado direito possui apenas um atributo. Mais precisamente, uma cobertura F é própria, se toda regra pertencente a F é da seguinte forma: $X \rightarrow a$ é reduzida à esquerda e $a \in M$.

Observe que esse tipo de cobertura possui uma estreita relação com as coberturas canônicas e, conseqüentemente, com coberturas reduzidas. De fato, uma cobertura canônica é uma cobertura própria não redundante. A relação com coberturas reduzidas é, ainda, de forma mais direta. Neste caso, uma cobertura própria G pode ser obtida diretamente de uma cobertura reduzida F, utilizando-se o axioma da decomposição, tal como demonstrado na seção 3.3.3. Note que não se impôs a condição que F seja não-redundante, assim, o resultado da decomposição das dependências de F é uma cobertura própria, e não necessariamente uma cobertura canônica.

3.3.5 Cobertura mínima

As coberturas não-redundantes, apesar de não apresentarem regras "descartáveis", não apresentam necessariamente o menor número de regras possíveis. Uma cobertura F é mínima, se não existe cobertura com número de regras menor. Obviamente, F é não-redundante, pois, caso contrário, existiria cobertura menor que F.

3.4 Implicações

Implicações são dependências entre elementos de um conjunto. Seja (G,M,I) um contexto formal. Uma implicação entre os atributos de M é um par (X,Y), $X,Y \subseteq M$, que, tal como as dependências funcionais, recebe a notação $X \rightarrow Y$.

Uma implicação $X \rightarrow Y$ é dita ser **válida** para o contexto (G,M,I) se, e somente se, todo objeto que possui os atributos de X também possui os atributos de Y. Formalmente, $X \rightarrow Y$é válida $\iff \forall g \in G[\forall x \in X\ gIx \rightarrow \forall y \in Y\ gIy]$, ou seja, $X' \subseteq Y'$. Caso nenhum objeto possua os atributos de X, a implicação $X \rightarrow Y$ é dita ser válida por vacuidade. Graficamente, a validade de uma implicação pode ser verificada através do diagrama de linha da seguinte forma: se o maior conceito contendo os atributos de X for menor ou igual ao maior conceito contendo os atributos de Y, então $X \rightarrow Y$ é uma implicação válida. Retomando o exemplo apresentado no

capítulo 1, a implicação *razão* $\rightarrow$ *pêlo e mamífero* é válida para o reticulado da Figura 1, pois o maior conceito contendo *razão*, $(\{homem\}, \{pêlo, mamífero, razão\})$, é menor do que o maior conceito contendo *pêlo* e *mamífero*, $(\{macaco, homem\}, \{pêlo, mamífero\})$.

Apesar de menos utilizadas para análise de dados, as implicações também são de interesse prático. Em algumas ocasiões, existe o interesse em se encontrar regras de associações entre dados cuja confiança seja alta e cujo suporte não tenha restrição. Por exemplo, considerando dados de mercado, pode-se encontrar a seguinte regra de associação *cerveja* $\rightarrow$ *batata frita* indicando relacionamento entre itens bastante vendidos, porém, regras do tipo *uísque 18 anos*$\rightarrow$ *salmão defumado* não serão abrangidas já que poucos clientes compram uísque e salmão. Em outras palavras, as regras de associação exprimem padrões freqüentes de dados, enquanto as implicações revelam dados correlacionados. Cohen [17] e Wang [56] discutem o problema da identificação de regras de associação com alta confiança, mas, sem restrições ao suporte. Eles indicam que a motivação para a identificação deste tipo de regra encontra-se no fato de que elas revelam mais novidades acerca dos dados que as regras de associação tradicionais. De acordo com Cohen [17], este tipo de regra possui aplicações em diversas áreas como na detecção de cópias, para verificar documentos ou páginas web similares ou idênticas e *clustering*. Estes argumentos demonstram a importância dos algoritmos apresentados na seção 3.5.

3.5 Algoritmos

Os algoritmos apresentados aqui foram, inicialmente, propostos para a identificação de implicações. Entretanto, como o problema da extração de dependências funcionais é equivalente ao da identificação de implicações [31], os algoritmos podem ser utilizados com o intuito de extrair dependências funcionais através da transformação de contextos formais multivalorados em contextos formais univalorados. Além destes algoritmos, foram apresentados, na seção 3.1.5, novos operadores de fecho que podem ser utilizados como alternativa à transformação de contextos.

As seções seguintes apresentam os algoritmos que são comparados neste trabalho.

3.5.1 *Next Closure*

O algoritmo *Next Closure* foi proposto por B. Ganter [27] com o objetivo inicial de gerar conjuntos fechados [20, 31]. Entretanto, como demonstrado por Ganter e Wille [31], ele pode ser usado para a identificação de implicações em contextos formais. A descrição do algoritmo é bastante simples e seu funcionamento é baseado na utilização de operadores de fecho para

encontrar conjuntos fechados na ordem lexicográfica.

Seja (G,M,I) um contexto formal. O algoritmo pressupõe a definição de uma ordem total entre os elementos de M. Assim, para um conjunto M com n elementos, pode-se, por exemplo, definir a seguinte ordem: $M = \{m_1 < m_2 < \ldots < m_n\}$.

O algoritmo é dependente da ordem lexicográfica "$\leq$" entre subconjuntos de M. Assim, para $A,B \subseteq M, A \neq B$, $A < B \Leftrightarrow \exists m_i \in B(m_i \notin A \wedge \forall m_j < m_i(m_j \in A \Leftrightarrow m_j \in B))$. Informalmente, A é lexicograficamente menor que B se o menor elemento, para o qual os dois conjuntos diferem-se, pertence a B.

O algoritmo pressupõe, também, a existência de um procedimento para encontrar o próximo conjunto fechado dados um conjunto $A \subseteq M$, um atributo $m_i \in M$ e um operador de fecho. O operador de fecho a ser considerado será a composição dos operadores de derivação $((\cdot)^{\downarrow})^{\uparrow}$, que receberá a notação $(\cdot)''$. Assim, o próximo conjunto fechado $A \oplus m_i$, para A e o atributo m_i segundo o operador $(\cdot)''$, é encontrado da seguinte forma:

1 $C = \{m_j \in A | m_j < m_i\} \cup \{m_i\}$,

2 $A \oplus m_i = C''$.

Note que para outros operadores de fecho, o algoritmo mantem-se inalterado, exceto para a linha 3, na qual deve-se utilizar o operador escolhido.

Exemplo 2. Considere o conjunto de atributos $M = \{1,2,3,4,5,6,7,8\}$ referentes aos atributos do contexto apresentado na Tabela 1, em que 1 refere-se a aquático, 2 a terrestre, e assim por diante; e a ordem linear $<$ usual imposta a M. Seja $A = \{2,7\}$ um conjunto de atributos, $m_i = 5$ um atributo, o próximo conjunto fechado $A \oplus 5$ será encontrado da seguinte forma:

1 $C = \{2\} \cup \{5\} = \{2,5\}$

2 $A \oplus 5 = C'' = \{2,5,7\}$

□

Assim, pode-se enunciar o algoritmo *Next Closure* tal como apresentado pelo Algoritmo 1.

O conjunto das implicações válidas em um contexto (G,M,I) é formado por implicações do tipo $A \rightarrow B$ em que $B \subseteq A''$. Isto denota a importância do algoritmo *Next Closure* para

Algoritmo Next Closure
Entrada: Um conjunto $A \subseteq M$ e um operador de fecho $(\cdot)''$
Saída: O próximo conjunto fechado segundo a ordem lexicográfica
início
1. $i := |M|$
2. **enquanto** $i \geq 1$ **faça**
3. $B := A \oplus m_i$
4. **se** $A < B$ **então**
5. **retorne** B
6. **fim se**
7. $i := i - 1$
8. **fim enquanto**
9. **retorne** $\emptyset$ /* *não há próximo* */
fim

Algoritmo 1: Algoritmo *Next Closure*.

a identificação de implicações. Obviamente, se $A = A''$ então $A \rightarrow A''$ é sempre válida. O leitor mais atento pode observar que computando todas implicações $A \rightarrow B \subseteq A''$, na verdade, computa-se a cobertura completa de implicações válidas. Note que para uma cobertura F, computar F^+ é caro; de fato, a computação é proporcional ao total de subconjuntos de cada conjunto B tal que $A \rightarrow B \in F$, ou seja, é proporcional a $2^{|B|}$ para cada implicação $A \rightarrow B$. Felizmente, pode-se definir o operador de fecho $\phi : \wp(M) \rightarrow \wp(M)$ de um conjunto de atributos A com relação a uma cobertura F, de forma que $A \rightarrow B \in F^+ \Leftrightarrow B \subseteq \phi(A)$.

O operador ϕ pode ser computado da seguinte forma: seja $\mathcal{L}$ a cobertura de implicações mínima, $X \subseteq M$, $\phi(X) = X \cup \bigcup\{P'' | P \rightarrow P'' \in \mathcal{L}, P \subset X\}$. Computa-se $\phi(X) = \phi(\phi(\phi(\ldots\phi(X))))$, até que $\phi(X) = \phi(\phi(X))$. Informalmente, aplica-se recursivamente o operador de fecho ϕ em X até o momento em que $\phi(X) = \phi(\phi(X))$. Como o objetivo do trabalho é o de mostrar a utilidade do operador para obtenção de regras, os detalhes foram omitidos. Entretanto, a descrição detalhada do operador e a prova da correção do algoritmo para computá-lo pode ser encontrada em [54].

A definição do operador ϕ permite a descrição do algoritmo para encontrar a cobertura de implicações mínima. Como as implicações $A \rightarrow B = A''$ são sempre válidas, o interesse está em regras para as quais $B \neq A''$.

Segue, então, a descrição do algoritmo apresentado em Algoritmo 2: inicia-se o processo considerando $A = \emptyset$ e o conjunto de implicações $F = \emptyset$. A cada passo atualiza-se o conjunto A com o próximo fecho em relação a ele utilizando o *Next Closure* com o operador de fecho ϕ; se o próximo conjunto fechado (em relação a ϕ) não é fechado em relação ao operador de fecho obtido da composição dos operadores de derivação (i.e. $A \neq A''$), então $F = F \cup \{A \rightarrow A''\}$; opcionalmente, pode-se optar por $F = F \cup \{A \rightarrow (A'' - A)\}$ uma vez que $A \subset A''$ e $A \rightarrow A$ é sempre válida. Repete-se o processo até que $A = M$. Mais uma vez a prova de correção do

algoritmo foi omitida, porém ela pode ser obtida em [31].

Algoritmo Cobertura Mínima
Entrada: Um contexto formal (G,M,I)
Saída: A cobertura mínima F
início
1. $F := \emptyset$
2. $A := \emptyset$
3. **enquanto** $A \neq M$ **faça**
4. **se** $A \neq A''$ **então**
5. $F := F \cup \{A \rightarrow A''\}$
6. **fim se**
7. $A := NextClosure(A, \phi)$
8. **fim enquanto**
9. **retorne** F

fim

Algoritmo 2: Algoritmo Cobertura Mínima.

Apresentado o algoritmo, pode-se analisar o comportamento assintótico de tempo e espaço em relação ao contexto (G,M,I) recebido como parâmetro. A análise do comportamento assintótico do tempo de execução envolve a análise do custo de cada operador de fecho e do número de conjunto fechados de M. O custo do operador de fecho relativo ao operador de derivação é $O(|G||M|)$. O custo do operador ϕ é definido em função do número de implicações no conjunto mínimo de implicações, assim, pode-se enunciá-lo como $O(|F|)$. São gerados C conjuntos fechados. A cada conjunto fechado encontrado, são aplicados os dois operadores de fecho ($(\cdot)''$ e ϕ). Portanto, o custo geral do algoritmo é $O(C(|G||M| + |F|))$. Entretanto, deve-se constatar que a análise foi feita para identificação de implicações. Então, para a análise de dependências, deve-se considerar o contexto indicado como parâmetro um contexto transformado. Assim, para o contexto transformado tem-se que o comportamento assintótico em relação ao tempo é $O(C(\binom{|G|}{2}|M| + |F|))$, F é a cobertura mínima das dependências funcionais válidas. É importante ressaltar que o pior caso do algoritmo ocorre com contextos do tipo $(G,G,\neq)$, os quais representam situações extremas que dificilmente acontecem em bases de dados reais. Em relação ao comportamento do espaço utilizado, este é constante, pois a cada passo é necessário armazenar apenas o conjunto fechado corrente.

3.5.2 *Find Implications*

O algoritmo *Find Implications* é apresentado nesta seção. Este algoritmo tem como objetivo encontrar uma cobertura de implicações de um contexto formal (G,M,I) através de seu reticulado conceitual. O algoritmo foi proposto por C. Carpineto e seus co-autores [15]. Carpineto e

Romano apresentam, em seu livro sobre Análise Conceitual de Dados [14], o mesmo algoritmo de forma mais clara. A cobertura encontrada por este algoritmo, apesar de reduzida, pode ser redundante. Entretanto, pode-se utilizar algoritmos, tal como o apresentado por Maier [37], para remover a redundância de forma a obter uma cobertura equivalente à cobertura canônica apresentada na seção 3.3.3.

O reticulado conceitual é útil para a identificação de implicações porque uma implicação $A \rightarrow B$ é válida para um contexto (G,M,I), se o maior conceito de cuja intensão A é subconjunto, é menor que o maior conceito de cuja intensão B é subconjunto. Formalmente, $A \rightarrow B$ é válida se $(A',A'') \leq (B',B'')$.

Exemplo 3. Retomando o exemplo dado no capítulo 1 para o reticulado da Figura 1, a implicação *razão* $\rightarrow$ *pêlo,mamífero* é válida uma vez que ({*macaco, homem*}, {*terrestre, pulmonar, pêlo, mamífero*}) $\geq$ ({*homem*}, {*terrestre, pulmonar, pêlo, mamífero, razão*}). □

Considerando-se cada conceito (X,Y), o interesse nas implicações obteníveis de (X,Y), são nas implicações $A \rightarrow B$, $A \subseteq Y$, $B = Y - A$, tal que $A \rightarrow B$ não possa ser obtida a partir de um conceito (W,Z) superconceito de (X,Y), ou seja, a partir de um conceito $(W,Z) \geq (X,Y)$.

O conjunto de todas as implicações referentes a um conceito é redundante, mesmo respeitando as restrições impostas acima. Assim, acrescenta-se mais uma restrição fazendo com que o interesse seja em implicações reduzidas. Para obedecer tal restrição, considere o conjunto parcialmente ordenado $\langle \wp(Y), \subseteq \rangle$. Sejam dois conjuntos de atributos A e B ($A,B \in \wp(Y)$), A é denominado mais geral que B (mais específico que A), se $A \subset B$. Então, para encontrar implicações reduzidas à esquerda, pode-se iniciar a busca utilizando-se conjuntos de atributos mais gerais, especializando-os até que as restrições mencionadas no parágrafo anterior sejam satisfeitas. Este é o processo de funcionamento do algoritmo *Find Implications*. O algoritmo repete tal processo para cada um dos conceitos do reticulado, adicionando as regras obtidas a cada iteração à cobertura que será retornada. O Algoritmo 3 apresenta o *Find Implications*; o Algoritmo 4 refere-se ao comportamento do algoritmo descrito acima, e tem como objetivo encontrar as implicações de cada conceito; enquanto o Algoritmo 5 apresenta a parte referente à conferência de consistência da implicação em relação aos superconceitos de (X,Y).

Em relação à análise de complexidade do método, verifica-se que o ciclo do algoritmo principal (Algoritmo 3) é executado para todos os conceitos do contexto. Logo, é executado $O(|L|)$, sendo L o reticulado conceitual. Repare que a cada conceito, a função *Concept Implications* (Algoritmo 4) é acionada. A função atualiza o conjunto de antecedentes baseado em cada superconceito do conceito recebido como parâmetro. A atualização dos antecedentes refere-se ao ciclo das linhas 4–18 do Algoritmo 4. Observe que este é executado $|lhsSet|$ vezes, e que os ciclos das linhas 9–11 e 12–16 são executados em tempo proporcional a $O(|lhsSet||M|)$. Como

Algoritmo Find Implications
Entrada: Um reticulado $L = \langle \mathfrak{B}(G,M,I), \leq \rangle$
Saída: Uma cobertura F
início
1. $F := \emptyset$
2. **para cada** $(X,Y) \in L$ **faça**
3. $F := F \cup ConceptImplications((X,Y),L)$
4. **fim para**
5. **retorna** F

fim

Algoritmo 3: Algoritmo *Find Implications*.

Algoritmo Concept Implications
Entrada: Um conceito (X,Y) e um reticulado conceitual L
Saída: O conjunto C de implicações do conceito (X,Y)
início
1. $lhsSet := Y$
2. **para cada** (W,Z) tal que $(X,Y) \prec (W,Z)$ **faça**
3. $updLhsSet := \emptyset$ //*Contém especializações de atributos*
4. **para cada** $lhs \in lhsSet$ **faça**
5. **se** $lhs \not\subseteq Z$ **então**
6. $updLhsSet := updLhsSet \cup \{lhs\}$
7. **senão**
8. $candSpecSet := \emptyset$
9. **para cada** $m \in Y - \{lhs\}$ **faça**
10. $candSpecSet := candSpecSet \cup \{lhs \cup \{m\}\}$
11. **fim para**
12. **para cada** $candSpec \in candSpecSet$ **faça**
13. **se** $ConfirmSpec(candSpec, lhs, Z, lhsSet) = verdadeiro$ **então**
14. $updLhsSet := updLhsSet \cup \{candSpec\}$
15. **fim se**
16. **fim para**
17. **fim se**
18. **fim para**
19. $lhsSet := updLhsSet$**fim se**
20. **fim para**
21. $C := \{lhs \rightarrow Y - lhs |\ lhs \in lhsSet, Y - lhs \neq \emptyset\}$
22. **retorne** C

fim

Algoritmo 4: Algoritmo para encontrar implicações de um conceito.

Algoritmo Confirm Specialization
Entrada: O candidato a especialização $candSpec$, a intensão Z do superconceito, uma premissa lhs, e o conjunto de premissas $lhsSet$
Saída: Um valor lógico confirmando, ou não, a especialização
início
1. $S := \{s \in lhsSet - \{lhs\} |\ s \subseteq candSpec\}$
2. **se** $S = \emptyset \wedge candSpec \not\subseteq Z$ **então**
3. **retorna** $verdadeiro$
4. **senão**
5. **retorna** $falso$
6. **fim se**

fim

Algoritmo 5: Verifica a consistência da implicação em relação ao pai do conceito.

o ciclo das linhas 4–18 é executado q vezes, em que q é a maior quantidade de pais de um conceito, o tempo de execução da função é proporcional a $O(|L||max_i\{lhsSet_i\}|^2|M|q)$, em que $max_i\{lhsSet_i\}$ indica o maior conjunto de antecedentes.

Em relação à complexidade de espaço, é necessário $O(|\wp(M)|)$ para armazenar o conjunto dos antecedentes. A explicação é encontrada no fato que os antecedentes são subconjuntos das intensões dos atributos; a maior intensão refere-se à intensão do conceito ínfimo do reticulado que é M. Vale lembrar que esta é uma situação avaliada para casos extremos; na prática, isto pode não ocorrer, uma vez que as implicações referentes ao ínfimo são válidas por vacuidade na maioria dos casos. Portanto, pode-se desconsiderar o conceito ínfimo durante a extração de regras. Entretanto, deve-se considerar o espaço necessário para a armazenagem do reticulado. Então, o espaço necessário pelo algoritmo é $O(|L||\wp(M)|)$.

3.5.3 Impec

O terceiro algoritmo a ser abordado neste trabalho é o Impec. Este algoritmo foi apresentado por R. Taouil e Y. Bastide [52] e tem como objetivo encontrar a base de implicações própria (seção 3.3.4). O objetivo é encontrar implicações cujo lado esquerdo é mínimo e o lado direito possui apenas um atributo. O algoritmo baseia-se em conjuntos e operadores de fecho definidos sobre estes conjuntos. Assim, para a descoberta das implicações (ou dependências funcionais) são necessários o conjunto de atributos M e um operador de fecho definido sobre M.

Enunciando o problema em outras palavras, o objetivo do algoritmo é encontrar implicações do tipo $A \rightarrow b$, em que $A \subseteq M, b \in M$ e não existe implicação válida da forma $D \rightarrow b$ em que $D \subset A$. O algoritmo encontra o lado direito de uma implicação através da proposição 4.

Proposição 4. *Seja $A \subseteq M$ e $(\cdot)''$ um operador de fecho definido sobre M. O conjunto dos atributos $ld(A) = \{b \in M | A \rightarrow b\}$ é obtido da seguinte forma:*

$$ld(A) = A'' - (A \cup \bigcup \{ld(B) |\ B \subset A\})$$

Demonstração. Esta proposição foi retirada de [31, Proposição 22]. Assim, detalhes e a prova da proposição podem ser obtidas na referência citada. □

As implicações a serem descobertas pelo algoritmo devem ser reduzidas à esquerda. Seja F a cobertura descoberta pelo algoritmo, se $A \rightarrow ld(A)$ pertence a F, então, $B \rightarrow ld(A) \notin F$ para $B \supset A$. Logo, para garantir que as implicações sejam reduzidas à esquerda, utiliza-se a proposição 5 apresentada em [52, Proposição 1].

Proposição 5 (Proposição 1, [52]). *Seja $A,B \subseteq M$, $B \supset A$ e $(\cdot)''$ um operador de fecho definido sobre M, tem-se:*

$$B'' = A'' \iff B - A \subseteq A'' - A$$

Apresentadas as proposições que servem de base para o algoritmo, pode-se descrevê-lo. O Algoritmo 6 mostra, em pseudo-código, o Impec. O algoritmo verifica as implicações envolvendo cada atributo de M utilizando a própria cobertura computada. O algoritmo inicia a cobertura com a implicação $\emptyset \rightarrow \emptyset''$. Em seguida, entra em um ciclo para verificar as implicações de cada atributo de M (linhas 2 – 15). A obtenção de novas implicações é feita especializando-se as implicações já pertencentes à cobertura, linhas 4–14. Para cada especialização a ser considerada, aplica-se as proposições 5 (linhas 7 e 8), e 4 (linhas 9–11) respectivamente. Por fim, adiciona-se a nova implicação encontrada (linha 13).

Algoritmo Impec
Entrada: Um conjunto de atributos M, e um operador de fecho $(\cdot)''$ definido sobre M
Saída: Uma cobertura própria F
início

```
 1.  F := {∅ → ∅''}
 2.  para cada m ∈ M faça
 3.      F' := F
 4.      para cada A → B ∈ F' faça
 5.          Z := (A ∪ {m})'' − (A ∪ {m})
 6.          se Z ≠ ∅ então
 7.              R := {X → Y ∈ F' | B ⊃ A, B − A ⊆ Z}
 8.              F' := F' − R
 9.              para cada X → Y ∈ F', X ⊂ A ∪ {m} faça
10.                 Z := Z − Y
11.             fim para
12.         fim se
13.         F := F ∪ {A ∪ {m} → Z}
14.     fim para
15. fim para
16. retorne {A → b | b ∈ B, A → B ∈ F, B ≠ ∅, A ≠ ∅}
```

fim

Algoritmo 6: Encontra cobertura própria de um conjunto M.

Analisa-se o custo computacional envolvido com o algoritmo. A iniciar a análise pela complexidade de tempo, considera-se como operador de fecho parâmetro do algoritmo a composição dos operadores de derivação. Dessa forma, o custo relacionado à linha 5 é proporcional a $|G||M|$; por se tratar da extração de regras através da AFC, considera-se que M seja o conjunto de atributos de um contexto (G,M,I). O custo relacionado a aplicação das proposições 4 e 5 é, no pior caso, $O(|F'||M|)$. O ciclo das linhas 4–14 é executado $O(|F|)$ vezes. Logo, o tempo necessário para a computação das implicações de cada atributo é $O(|F|(|G||M| + |F||M|))$. Assim, tem-se que o tempo para todos atributos é $O(|M||F|(|G||M| + |F||M|))$.

O algoritmo utiliza a cobertura sendo computada para encontrar a cobertura a ser dada como resposta. Logo, deve-se levar em conta o espaço necessário para o armazenamento das regras

já computadas. Deve-se, também, levar em conta os conjuntos necessários para a computação das novas regras. Tais conjuntos são Z na linha 5, R na linha 7 e F' na linha 3. O conjunto Z é limitado pelo tamanho de M, ou seja, é $O(|M|)$. Os conjuntos F' e R são limitados pelo tamanho de F. F, por sua vez, é limitado por $2^{|M|}$. Portanto, pode-se definir um limite fraco para o espaço necessário como $O(2^{|M|})$.

3.5.4 Aprem-IR

O último algoritmo a ser apresentado é o *Aprem-IR*. Este algoritmo foi apresentado por Taouil e outros [53], também, por R. Taouil e seus co-autores. O algoritmo recebe, como parâmetro de entrada, uma lista de conceitos e retorna uma cobertura própria de implicações.

Seja (G,M,I) um contexto formal e $\mathfrak{B}(G,M,I)$ a lista de conceitos associados ao contexto. O conjunto dos antecedentes, para um atributo $m \in M$, é o conjunto

$$lhs(m) = \{X \in \mathfrak{B}(G,M,I) | m \notin X \wedge \forall Y \subset X \; Y \not\rightarrow m\}.$$

De acordo com Taouil e seus co-autores [53], este conjunto pode ser encontrado através de conjuntos ínfimo-irredutíveis. Esta afirmação é baseada em um teorema retirado do livro de Mannila e Raiha [38] o qual demonstra que o conjunto dos antecedentes de um atributo m é o conjunto de *minimal transversals* do hipergrafo formado com o complemento dos conjuntos máximos ínfimo-irredutíveis não contendo m ($CIRR(\bar{m})$). Como o algoritmo *Aprem-IR* baseia-se nestes conceitos, é importante descrevê-los com um pouco mais de detalhes.

Um conjunto C é ínfimo-irredutível, se ele não pode ser descrito como o ínfimo de elementos estritamente maiores que ele. Estes conjuntos podem ser, facilmente, detectados em um diagrama de linha observando-se os elementos com apenas um vizinho inferior. Para computar o conjunto $CIRR(\bar{m})$, é necessária a computação do conjunto $IRR(\bar{m})$ dos conjuntos máximos ínfimo-irredutíveis não contendo m. Como demonstrado por Mannila e Raiha [38], o conjunto $IRR(\bar{m})$ é idêntico ao conjunto das máximas intensões não contendo m. Assim, a família dos conjuntos máximos ínfimo-irredutíveis não contendo m é

$$IRR(\bar{m}) = \{Y | (X,Y) \in \mathfrak{B}(G,M,I),\ m \notin Y,\ \forall (W,Z) \in \mathfrak{B}(G,M,I)[Y \subset Z \rightarrow m \in Z]\}.$$

Obtido o conjunto $IRR(\bar{m})$, pode-se obter o conjunto $CIRR(\bar{m})$ já que

$$CIRR(\bar{m}) = \{M - I | I \in IRR(\bar{m})\}.$$

O conjunto $CIRR(\bar{m})$ forma um hipergrafo cujos vértices são os atributos de M e as arestas os elementos de $CIRR(\bar{m})$. Um *transversal* em um hipergrafo é um subconjunto de vértices

que possui interseção com todas as arestas do hipergrafo. Seja $\mathcal{H} = (M, \mathcal{E})$ um hipergrafo cujo conjunto de arestas seja $\mathcal{E}$ e cujo conjunto de vértices seja M e seja $T \subseteq M$. T é um *transversal* se, para toda aresta $E \in \mathcal{E}$, $T \cap E \neq \emptyset$. T é um *minimal transversal* se não existe subconjunto próprio de T que seja também um *transversal*. Assim, o conjunto dos *minimal transversals* associados a $CIRR(\bar{m})$ é

$$\begin{aligned} MTR(CIRR(\bar{m})) \quad = \quad & \{T \subseteq M | T \neq \{m\}, \forall C \in CIRR(\bar{m})[C \cap T \neq \emptyset], \\ & \forall X \subseteq M \text{ se X é um } \textit{transversal} \text{ e } X \subseteq T, \text{ então } X = T\}. \end{aligned}$$

Como o algoritmo *Aprem-IR* baseia-se no fato de que o conjunto dos antecedentes de um atributo m, $lhs(m)$, é igual ao conjunto $MTR(CIRR(\bar{m}))$, ele possui basicamente duas etapas. Primeiro, ele determina, para cada atributo m do contexto, uma lista com o complemento dos conjuntos ínfimo-irredutíveis máximos não contendo m. Depois, determina o conjunto dos *minimal transversals* e, com isso, o conjunto de implicações para cada atributo $m \in M$. O pseudo-código do algoritmo é apresentado pelo Algoritmo 7. As funções *obterCIRR* e *obterRegras* referem-se às duas etapas mencionadas anteriormente as quais são apresentadas em Algoritmo 8 e 9.

Algoritmo Aprem-IR
Entrada: Uma lista de conceitos formais $\mathfrak{B}(G,M,I)$
Saída: Uma cobertura canônica F
início
1. $CIRR := obterCIRR(\mathfrak{B}(G,M,I))$
2. $F := obterRegras(CIRR)$
3. **retorne** F
fim

Algoritmo 7: Encontra cobertura canônica dada uma lista de conceitos $\mathfrak{B}(G,M,I)$.

O Algoritmo 8, para obter os conjuntos $CIRR(\bar{m})$ de cada atributo m, inicialmente, insere o conjunto vazio na lista dos conjuntos ínfimo-irredutíveis não contendo m (linhas 1 a 3). Em seguida, utilizando-se do fato de que $IRR(\bar{m})$ é igual ao conjunto das máximas intensões não contento m, o algoritmo verifica cada conceito (linhas 4 a 15). Para cada atributo m não pertencente à intensão do conceito, remove-se de $IRR(\bar{m})$ todo subconjunto da intensão do conceito (linhas 6 a 10). Nas linhas 11 a 13, o algoritmo verifica se a intensão do conceito C não é subconjunto de nenhum conjunto pertencente a $IRR(\bar{m})$ (i.e. verifica se a intensão do conceito é máxima) antes de inserí-la no conjunto. Finalmente, nas linhas 16 a 21, o algoritmo obtém, para cada atributo m, o complemento dos conjuntos máximos ínfimo-irredutíveis não contendo m.

A segunda etapa do algoritmo, a obtenção dos *minimal transversals* e das implicações, é apresentada em pseudo-código pelo Algoritmo 9. O algoritmo inicia com a base de implicações

Algoritmo obterCIRR
Entrada: Uma lista de conceitos formais $\mathfrak{B}(G,M,I)$
Saída: O complemento dos conjuntos máximos ínfimo-irredutíveis associados a cada atributo *CIRR*
início

```
1.  para cada m ∈ M faça
2.      IRR(m̄) := ∅
3.  fim para
4.  para cada C ∈ 𝔅(G,M,I) faça
5.      para cada m ∈ M − int(C) faça
6.          para cada conjunto ínfimo-irredutível I ∈ IRR(m̄) faça
7.              se I ⊂ int(C) então
8.                  IRR(m̄) := IRR(m̄) − I
9.              fim se
10.         fim para
11.         se ∀I ∈ IRR(m̄)[int(C) ⊄ I] então
12.             IRR(m̄) := IRR(m̄) ∪ {int(C)}
13.         fim se
14.     fim para
15. fim para
16. para cada m ∈ M faça
17.     CIRR(m̄) := ∅
18.     para cada conjunto ínfimo-irredutível I ∈ IRR(m̄) faça
19.         CIRR(m̄) := CIRR(m̄) ∪ {M − I}
20.     fim para
21. fim para
22. retorne CIRR
fim
```

Algoritmo 8: Encontra o complemento dos conjuntos máximos ínfimo-irredutíveis.

vazia (linha 1). Em seguida, entra em uma repetição para encontrar os antecedentes de cada atributo de M (linhas 2 a 28). Para determinar o conjunto $MTR(\bar{m})$, o algoritmo inicia com o conjunto vazio (linha 3) e obtém os *minimal transversals* gradualmente. Isto é, na i-ésima iteração, o algoritmo obtém os *minimal transversals* de tamanho i e determina os potenciais *minimal transversals* de tamanho $i+1$. Os candidatos iniciais a *minimal transversals* são obtidos na linha 4. Seguindo, nas linhas 5 a 24, todos os candidatos a *minimal transversal* são considerados. O ciclo das linhas 6 a 11 verifica quais candidatos do conjunto *GEN* são *minimal transversals* (linha 7), insere aqueles que, realmente, são (linha 8), removendo-os do conjunto de candidatos (linha 9). Nas linhas 12 a 23, o algoritmo determina os novos candidatos a *minimal transversals*. Inicialmente, ao conjunto dos novos candidatos, é atribuído o conjunto vazio (linha 12). Depois, cada candidato G remanescente em *GEN* (linhas 13 a 22) é combinado com os demais candidatos G' tais que $|G| < |G'|$ e $|G \cap G'| = |G| - 1$ (linhas 14 a 21). Assim, o pontencial novo candidato $G'' = G \cup G'$ é gerado (linha 16). Se todos os subconjuntos de G'' com tamanho $|G''| - 1$ elementos já foram avaliados como candidatos a *minimal transversals* (linha 17), então, G'' é inserido no conjunto dos novos candidatos (linha 18). Após todos *minimal transversals* terem sido encontrados, as implicações do tipo $T \rightarrow m$ são encontradas para cada antecedente (*minimal transversal*) $T \in MTR(\bar{m}) - \{m\}$ no ciclo das linhas 25 a 27. Enfim, após todos os atributos terem sido considerados, a base de implicações F é retornada.

Algoritmo obterRegras
Entrada: Um conjunto com os complementos dos conjuntos máximos ínfimo-irredutíveis $CIRR$
Saída: Uma cobertura canônica F
início

1. $F := \emptyset$
2. **para cada** $m \in M$ **faça**
3. $MTR(\bar{m}) := \emptyset$
4. $GEN := \{\{a\} | a \in \bigcup CIRR(\bar{m})\}$
5. **enquanto** $GEN \neq \emptyset$ **faça**
   ** Verificando quais $G \in GEN$ são* minimal transversals ***
6. **para cada** $G \in GEN$ **faça**
7. **se** $\forall I \in CIRR(\bar{m}) I \cap G \neq \emptyset$ **então**
8. $MTR(\bar{m}) := MTR(\bar{m}) \cup \{G\}$
9. $GEN := GEN - \{G\}$
10. **fim se**
11. **fim para**
    ** Encontrando novos potenciais* minimal transversals *em GEN **
12. $GEN' := \emptyset$
13. **para cada** $G \in GEN$ **faça**
14. **para cada** $G' \in GEN$ tal que $|G| < |G'|$ **faça**
15. **se** $|G \cap G'| = |G| - 1$ **então**
16. $G'' := G \cup G'$
17. **se** $\{S \subset G'' | \, |S| = |G''| - 1\} \subseteq GEN$ **então**
18. $GEN' := GEN' \cup \{G''\}$ **então**
19. **fim se**
20. **fim se**
21. **fim para**
22. **fim para**
23. $GEN := GEN'$
24. **fim enquanto**
    ** Determinando as regras cujo lado direito é m a partir de $MTR(\bar{m})$ **
25. **para cada** *transversal* $T \neq \{m\} \in MTR(\bar{m})$ **faça**
26. $F := F \cup \{T \rightarrow m\}$
27. **fim para**
28. **fim para**
29. **retorne** F

fim

Algoritmo 9: Encontra o conjunto de implicações próprias a partir dos $CIRR$.

3.6 Comparação dos métodos

Os algoritmos são comparados sob o ponto de vista prático nesta seção. Eles foram implementados em *Java*. Os testes foram realizados em um computador Pentium 3, 850 MHz com 440 MB de memória principal em sistema operacional Windows XP. O leitor pode questionar a escolha da linguagem de programação, uma vez que ela não oferece a melhor utilização dos recursos computacionais. Entretanto, a implementação dos algoritmos tem como objetivo compará-los, e não o objetivo de fornecer implementações eficazes dos mesmos. Vale ressaltar que, pela implementação ser em Java, também, não é garantido que recursos da linguagem como o coletor de lixo não tem impacto nos testes.

A intensão dos testes é verificar o comportamento dos algoritmos diante de situações reais. Gerou-se bases de dados sintéticas para a realização dos testes, expondo os algoritmos a situações extremas. Além de bases sintéticas, foram utilizadas algumas bases reais retiradas do repositório de bases de dados da Universidade da Califórnia em Irvine [22].

As bases sintéticas utilizadas para avaliar o comportamento dos algoritmos para extrair implicações foram geradas através da metodologia proposta por Agrawal [3] com o auxílio do programa implementado por ele e adaptado por M. Zaki que o dispôs em sua página da web[8].

As bases foram geradas para verificar o desempenho dos algoritmos diante de contextos formais com diferentes densidades[9] e com diferentes quantidades de objetos. A variação da densidade foi feita mantendo-se as dimensões dos contextos (número de atributos e objetos) e variando-se a média de atributos por objeto. Variou-se a média de atributos por objeto de forma a obter densidades dos contextos entre 20 e 70%. O número de objetos foi variado entre 100 e 10000 objetos.

O objetivo de avaliar os algoritmos sob os pontos de vista da densidade dos contextos e do número de objetos encontra-se no fato de que a construção do reticulado conceitual é sensível à relação I [14] e a variação dos objetos foi feita para avaliar o desempenho dos algoritmos com bases de dados de tamanhos diferentes. Esses testes tentam cobrir uma deficiência constatada na literatura, pois os testes já apresentados não comparam os algoritmos sob os dois pontos de vista mencionados e, em geral, utilizam bases de dados reais porém sem a definição de critérios bem estabelecidos.

As bases de dados retiradas do repositório foram tratadas e transformadas em contextos formais antes de serem utilizadas para avaliar os algoritmos. O tratamento inclui a remoção de

[8] http://www.cs.rpi.edu/∼zaki/software/.

[9] A densidade de um contexto formal (G,M,I) é $|I|/(|M| * |G|)$.

tuplas com valores desconhecidos[10], a discretização de atributos contínuos através da adoção de intervalos e a transformação de atributos multivalorados em atributos univalorados.

A avaliação do desempenho dos algoritmos para a extração de dependências funcionais também foi feita utilizando-se bases sintéticas. A geração desses contextos multivalorados deu-se através da escolha aleatória dos valores de cada atributo de cada objeto. Exemplificando, escolhendo-se $|W| = 10$, um valor no intervalo de 1 a 10 é atribuído a cada atributo de cada objeto. Gerou-se contextos formais com 50, 100, 150 e 200 objetos ($|G| = 50$, $|G| = 100$ ...) e com 7 e 10 atributos com a mesma quantidade de valores por atributo, ou seja, para $|M| = 7$ o número de valores por atributo também é 7. A variação do número de atributos e do número de objetos foi relativamente pequena pois, ao se considerar o problema da descoberta de dependências funcionais, deve-se considerar a transformação de contextos descrita na seção 3.1.4 que transforma um problema de ordem n em um de ordem n^2. Assim, contextos formais relativamente pequenos quando multivalorados, ao serem transformados em univalorados, tornam-se contextos com tamanho significativo.

A Tabela 5 apresenta as bases de dados utilizadas para a avaliação dos algoritmos. A tabela apresenta informações sobre as bases já transformadas em contextos formais tais como o número de atributos, o número de objetos, o tamanho da relação I, a média de atributos por objetos e a densidade do contexto formal. Os contextos formais de 2 a 14 na Tabela 5 foram utilizados para avaliar os algoritmos na extração de implicações, enquanto os contextos de 15 a 22 foram utilizados para a extração de dependências funcionais. O contexto formal 1 foi utilizado tanto para extração de implicações quanto para dependências funcionais. Outra ressalva em relação à Tabela 5 é que os contextos de 15 a 22 são contextos formais multivalorados, porém, a densidade deles foi calculada baseando-se no contexto formal univalorado obtido através da escala plana.

Os algoritmos foram inicialmente comparados para a extração de implicações. A Tabela 6 apresenta o desempenho dos algoritmos. Nela são apresentados o tempo de execução e o número de regras extraídas com os algoritmos. Os contextos 1 e 2 na tabela, respectivamente a base de dados referente a recomendações sobre o uso de lentes de contato e a base de dados sobre doenças hepáticas, representam duas situações distintas. O contexto 1 é bastante pequeno, contendo apenas 24 objetos e 11 atributos. Ele foi escolhido com o intuito de avaliar o comportamento dos algoritmos diante de problemas simples. Já o contexto 2, apesar de ser também um contexto pequeno, foi escolhido porque os dados são bastante correlacionados e o contexto apresenta alta densidade. Logo, esse contexto foi escolhido para avaliar o comportamento dos algoritmos frente a bases de dados complexas com dados com alto índice de correlação. O

[10]Em geral, as bases do repositório apresentam tuplas cujos valores para alguns atributos são, às vezes, desconhecidos. Nesses casos, a tupla foi removida da base.

ID	Contexto	# Objetos	# Atributos	|I|	média de atributos/objeto	Densidade(%)
1	lentes	24	11	108	4	40,91
2	hepatite	80	24	1231	15	64,11
3	T5I4D0.1K	100	10	545	5	54,50
4	T5I4D0.5K	500	10	2676	5	53,52
5	T5I4D1K	1000	10	5286	5	52,86
6	T5I4D2K	2000	10	9511	5	47,56
7	T5I4D10K	10000	10	52899	5	52,90
8	M10T2I2D0.5K	500	10	1188	2	23,76
9	M10T3I2D0.5K	500	10	1581	3	31,62
10	M10T4I2D0.5K	500	10	1927	4	38,54
11	M10T5I2D0.5K	500	10	2345	5	46,90
12	M10T6I2D0.5K	500	10	2696	6	53,92
13	M10T7I2D0.5K	500	10	3126	7	62,52
14	M10T8I2D0.5K	500	10	3444	8	68,88
15	Aleatorio50_10_10	50	10	500	10	10,00
16	Aleatorio50_7_7	50	7	350	7	14,29
17	Aleatorio100_10_10	100	10	1000	10	10,00
18	Aleatorio100_7_7	100	7	700	7	14,29
19	Aleatorio150_10_10	150	10	1500	10	10,00
20	Aleatorio150_7_7	150	7	1050	7	14,29
21	Aleatorio200_10_10	200	10	2000	10	10,00
22	Aleatorio200_7_7	200	7	1400	7	14,29

Tabela 5: Contextos formais para avaliação dos algoritmos

desempenho dos algoritmos para esses contextos foi satisfatório. Tanto o *Next Closure* quanto o *Find Implications* gastaram cerca de 1 segundo para extrair implicações do contexto 1 e gastaram, respectivamente, cerca de 4 minutos e pouco mais de 1 minuto para o contexto 2. Na Tabela 6, os tempos de execução do Impec e do Aprem-IR não foram apresentados para o contexto 2 pois os testes foram abortados devido a falta de memória.

O desempenho dos algoritmos para a variação da densidade e da quantidade de objetos são apresentados graficamente nas Figuras 2 e 3.

Analisando-se a Figura 2, observa-se que o desempenho dos algoritmos *Find Implications* e Aprem-IR está relacionado à densidade dos contextos. À medida que a densidade dos contextos aumenta, observa-se o aumento no tempo de execução. Entretanto, esse comportamento não foi observado para o Impec e o *Next Closure*. O Impec teve o tempo de execução reduzido ao aumentar a densidade do contexto. Já o *Next Closure* não sofreu aumento no tempo de execução com o aumento da densidade dos contextos, mantendo tempo de execução praticamente constante. Os algoritmos Aprem-IR e *Find Implications* sofreram maior influência no tempo de execução com o aumento da densidade porque utilizam diretamente reticulados conceituais para a extração de implicações. Como mostrado por Carpineto e Romano [14], o desempenho dos algoritmos para a construção de reticulados conceituais depende diretamente da relação I. O pior caso para esses algoritmos ocorre quando os objetos do contexto formal possuem todos

ID Contexto	Tempo de execução (segundos)				# regras			
	NC	FI	Impec	Aprem-IR	NC	FI	Impec	Aprem-IR
1	0,96	0,81	600,39	5,23	25	67	134	134
2	225,58	77,22	–	–	147	3229	–	–
3	2,06	2,62	581,20	28,34	40	165	90	90
4	2,45	12,80	573,51	48,14	24	117	34	34
5	1,98	31,96	569,03	38,30	12	49	21	21
6	2,99	36,67	39,33	49,46	27	94	46	46
7	6,39	1768,87	582,66	51,96	5	6	14	14
8	5,48	2,00	245,25	28,43	70	53	375	375
9	4,32	3,92	272,83	75,03	72	120	199	199
10	7,90	5,26	78,54	169,94	41	118	80	80
11	3,40	8,27	116,40	98,44	27	100	34	34
12	5,68	12,21	99,51	80,33	16	100	21	21
13	3,37	16,93	105,14	80,85	22	146	26	26
14	3,34	24,56	82,57	102,96	20	186	23	23

Tabela 6: Desempenho dos algoritmos para implicações. NC = *Next Closure* e FI = *Find Implications*.

os atributos exceto aqueles na diagonal principal. Como, ao aumentar a densidade dos contextos, eles aproximam-se do pior caso, os desempenhos do Aprem-IR e do *Find Implications* são degradados ao aumentar a densidade. No entanto, entre o Aprem-IR e o *Find Implications*, observa-se que o desempenho do *Find Implications* é superior. O motivo dessa superioridade está na escolha do algoritmo para a construção do reticulado conceitual. A proposta original do Aprem-IR apresentada por Taouil e co-autores [53] supõe a utilização do algoritmo Aprem para a construção do reticulado, já o *Find Implications* utiliza o algoritmo *Concept Covers* de L. Nourine e O. Raynaud [42] para a construção do reticulado. Esse algoritmo é apontado, na literatura, como o mais eficiente para a construção do reticulado.

Em relação ao aumento na quantidade de objetos, a Figura 3 revela que, em geral, os algoritmos mostraram-se pouco sensíveis. O tempo de execução dos algoritmos permaneceu praticamente constante ao se variar a quantidade de objetos. Uma situação inesperada é observada quando o número de objetos é 2000. Nesse ponto, o algoritmo Impec obteve o menor tempo de execução. Contudo, observando-se a descrição dos contextos apresentada na Tabela 5, constata-se que a densidade do contexto 6 referente aos 2000 objetos é inferior em relação aos contextos 3, 4, 5 e 7 referentes ao número de objetos 100, 500, 1000 e 10000 (cerca de 47% contra 53% dos demais) e, observando-se novamente a Figura 2, constata-se que, por volta de 47%, o tempo de execução esperado é, realmente, inferior em relação ao de 53%.

Os algoritmos *Next Closure* e Aprem-IR apresentaram os melhores resultados com relação ao aumento da quantidade de objetos no contexto. Eles sofreram pequeno acréscimo no tempo de execução com o aumento do número de objetos. O algoritmo que mostrou-se mais sensível em relação ao aumento do número de objetos foi o *Find Implications* que apresentou

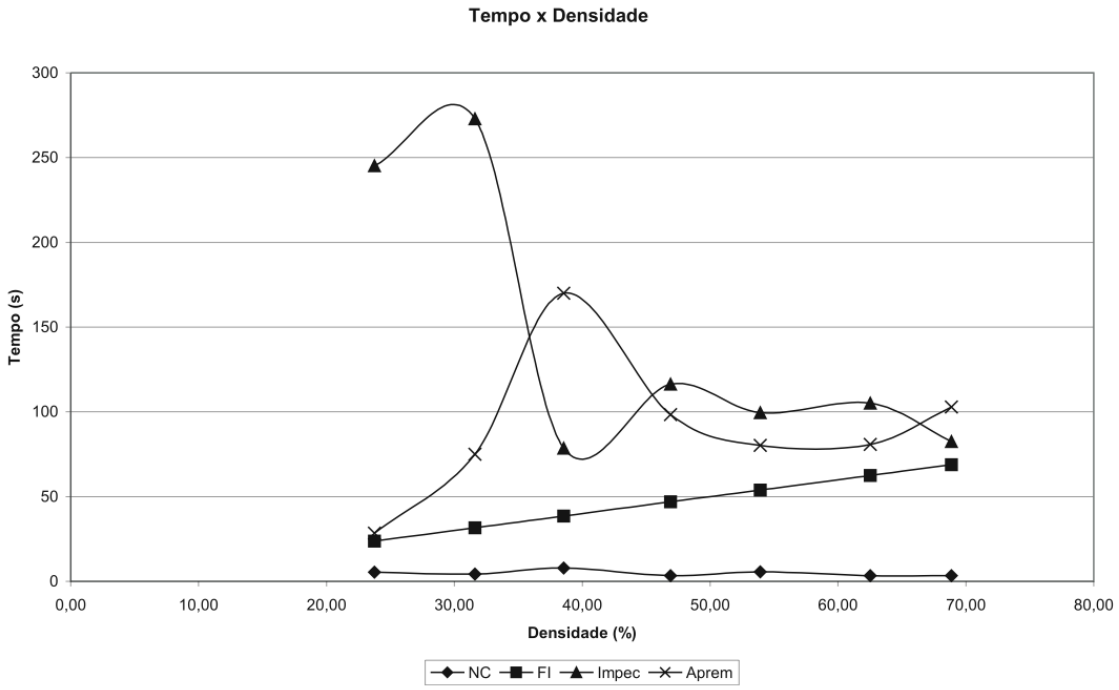

Figura 2: Desempenho dos algoritmos para extração de implicações variando-se a densidade.

crescimento exponencial do tempo de execução. Mais uma vez o fato relaciona-se à construção do reticulado. Parte do algoritmo *Concept Covers*, utilizado na construção do reticulado pelo *Find Implications*, computa diversas vezes fechos de atributos, aplica os operadores de derivação com freqüência e realiza interseções entre as extensões de conceitos e conjuntos de objetos resultados da aplicação do operador de derivação de atributos para a descoberta da relação de cobertura entre conceitos. O tempo para a computação dos operadores de fecho e de derivação de atributos e a computação das interseções é desprezível quando o número de objetos é pequeno, porém, quando o número de objetos cresce (como ocorreu no experimento) o tempo torna-se significativo e, com isso, o desempenho do algoritmo cai.

Analisando-se os algoritmos sob o ponto de vista do número de regras, observa-se que o aumento da densidade provoca uma queda no número de regras produzidas pelo *Next Closure*, Impec e Aprem-IR. Já para o *Find Implications* ocorre um leve aumento no número de regras. A queda do número de regras com o aumento da densidade é esperada uma vez que, com o aumento da densidade, os objetos do contexto tornam-se mais correlacionados possuindo mais atributos em comum e, assim, são necessárias menos regras para descrevê-los. O aumento do número de regras geradas pelo *Find Implications* também é esperado. O aumento da densidade do contexto acarreta o aumento no número de conceitos formais no reticulado e, assim, mais regras são geradas já que o algoritmo percorre o reticulado para gerá-las. Contudo, a maioria dessas regras são válidas por vacuidade e são obtidas pelo ínfimo do reticulado. Logo, pode-se

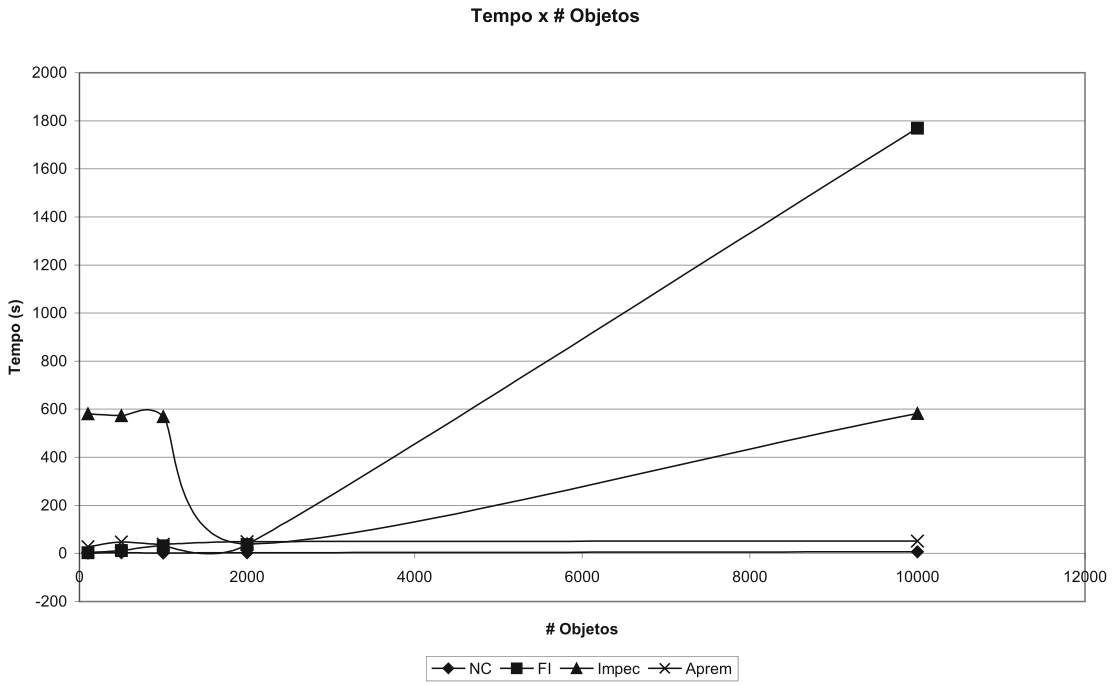

Figura 3: Desempenho dos algoritmos para extração de implicações variando-se o número de objetos.

modificar o algoritmo para desconsiderar essas regras. O mesmo comportamento é constatado com o aumento do número de objetos. No entanto, houve também queda no número de regras para o *Find Implications*. A queda do número de regras com relação ao aumento do número de objetos é normal já que, com o aumento do número de objetos, eles tornam-se mais diferentes, com menos atributos em comum e menos regras podem ser extraídas.

A comparação dos métodos para a extração de dependências funcionais foi realizada com os contextos 15 a 22 da Tabela 5. Os algoritmos foram comparados tanto utilizando a transformação de contextos quanto utilizando os operadores de particionamento. No entanto, o *Find Implications* não pôde ser comparado com os demais utilizando-se os operadores de particionamento, pois a construção do reticulado através do *Concepts Cover* baseia-se nos operadores de derivação. Esse fato não ocorre com o Aprem-IR que também utiliza reticulados diretamente, pois o Aprem-IR utiliza o algoritmo Aprem para construir o reticulado e ele baseia-se em operadores de fecho gerais[11] tal como o *Next Closure*. A Tabela 7 apresenta o tempo de execução e o número de regras para os quatro algoritmos através da transformação de contextos. A Tabela 8 apresenta o tempo de execução e o número de regras para os algoritmos utilizando os operadores de particionamento.

Os tempos de execução apresentados nas Tabelas 7 e 8 são apresentados graficamente na

[11] Quaisquer operadores que satisfaçam as propriedades dos operadores de fecho.

ID Contexto	Tempo de execução (segundos)				# regras			
	NC	FI	Impec	Aprem-IR	NC	FI	Impec	Aprem-IR
1	0,932	1,152	0,691	0,711	1	1	1	1
15	168,392	5,949	232,694	32,967	66	111	416	416
16	23,363	1,593	1,382	1,843	19	22	54	54
17	4960,073	20,99	212,645	136,286	142	161	734	734
18	124,178	11,056	5,067	4,987	14	15	33	33
19	7989,939	152,909	155,234	63,712	117	143	576	576
20	424,591	68,198	29,262	29,353	11	11	22	22
21	6939,409	400,976	2341,467	392,684	98	153	529	529
22	857,362	368,168	114,665	102,768	8	8	16	16

Tabela 7: Desempenho dos algoritmos para extrair dependências funcionais transformando contextos.

ID Contexto	Tempo de execução (segundos)			# regras		
	NC	Impec	Aprem-IR	NC	Impec	Aprem-IR
1	0,831	0,871	0,841	1	1	1
15	403,54	2309,381	147,252	66	416	416
16	21,27	47,769	16,103	19	54	54
17	2106,389	19051,676	1353,416	142	734	734
18	120,794	205,226	87,516	14	33	33
19	3288,699	12339,082	2159,405	117	576	576
20	390,482	532,145	305,198	11	22	22
21	18283,35	51086,416	22444,614	98	529	529
22	780,963	1034,157	576,858	8	16	16

Tabela 8: Desempenho dos algoritmos para extrair dependências funcionais com operadores de particionamento

Figuras 4, 5, 6 e 7. As duas primeiras comparam o desempenho dos algoritmos para contextos com 10 atributos e 10 valores por atributo com variações no número de objetos. A Figura 4 utilizando a transformação de contextos e a Figura 5 utilizando os operadores de particionamento. As duas últimas apresentam as mesmas comparações porém para contextos com 7 atributos e 7 valores por atributo.

Analisando as figuras de uma forma geral, percebe-se que os operadores de particionamento mostraram-se menos eficazes que a transformação de contextos. O tempo gasto para a computação do fecho utilizando os operadores de particionamento é maior que o tempo para a computação do fecho com os operadores de derivação. Isso faz com que o desempenho de todos os algoritmos seja comprometido; principalmente daqueles que foram desenvolvidos com base no operador de derivação como o Impec.

Obviamente, todos os algoritmos mostraram-se sensíveis ao aumento do número de objetos tanto utilizando a transformação de contexto quanto utilizando os operadores de particionamento. Todavia, se comparados os tempos de cada algoritmo entre si, observa-se que a diferença entre o tempo utilizando operadores de particionamento e utilizando a transformação de contexto é menor quando o número de atributos é menor. Exemplificando, tomando-se os contextos 15 e 16 (ambos com 50 objetos, porém o primeiro com 10 atributos e o segundo com 7) nas Tabelas 7 e 8 para o *Next Closure*, a diferença entre o tempo de execução com o uso de operadores de particionamento e transformação de contexto para o contexto 15 é de 235,15 segundos. Já a diferença entre o tempo com o uso dos operadores de particionamento e transformação de contexto para o contexto 16 é de 2,1 segundos. Além disso, em geral, o desempenho do *Next Closure* utilizando operadores de particionamento é melhor que utilizando a transformação de contexto quando o número de atributos é menor. Esse mesmo fato não ocorre para o Aprem-IR e para o Impec. Isso já era esperado para o Impec pois, como dito anteriormente, o Impec foi criado com base nos operadores de derivação. Contudo, o mesmo não se esperava para o Aprem-IR pois ele foi criado para ser utilizado com quaisquer operadores de fecho. Assim, esperava-se que o desempenho do Aprem-IR também fosse melhor com operadores de particionamento quando o número de atributos fosse menor. Isso revela um forte indício que o Aprem-IR foi criado com base nos operadores de derivação, mesmo sendo afirmado por seus autores que ele fora criado com base em operadores de fecho gerais.

Analisando-se as Figuras 4 e 6 observa-se que o desempenho dos algoritmos baseados em reticulado é melhor que os demais para a extração de dependências funcionais, ou seja, o desempenho do *Find Implications* e do Aprem-IR é melhor que o desempenho do *Next Closure* e do Impec. Observa-se também que o bom desempenho do *Next Closure* para a extração de implicações não foi mantido para a extração de dependências funcionais. Enquanto o *Next*

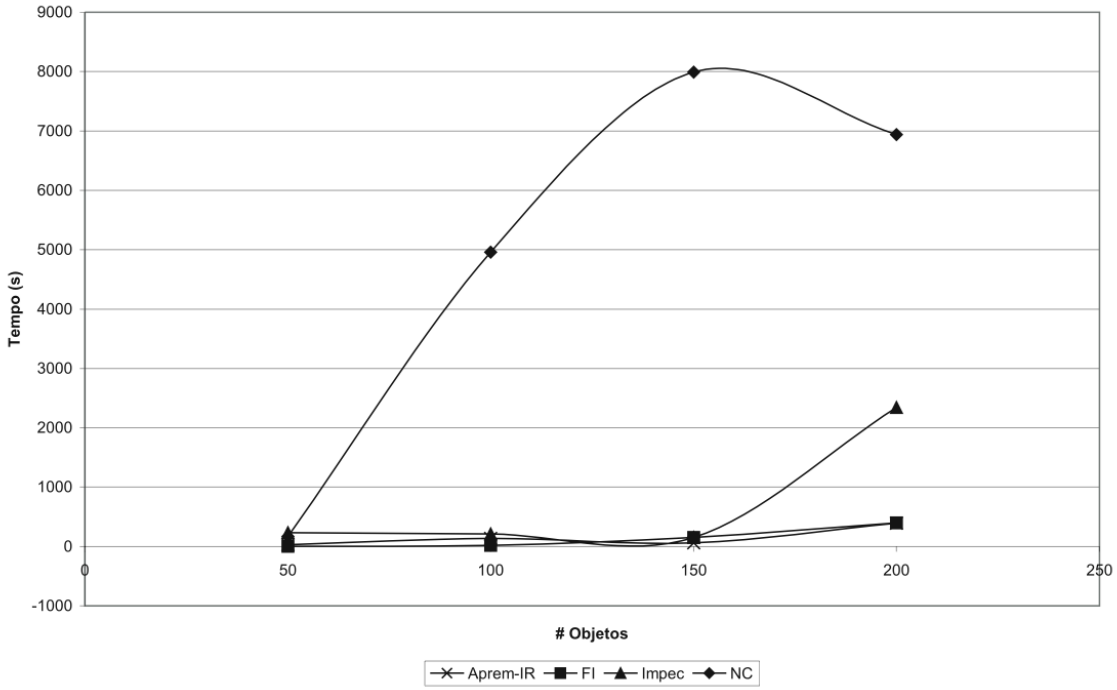

Figura 4: Desempenho dos algoritmos para extrair dependências funcionais variando número de objetos e transformando contextos. Número de atributos 10 e 10 valores por atributo.

Closure obteve o melhor resultado com relação ao tempo de execução para a extração de implicações, para a extração de dependências funcionais, ele obteve o pior resultado entre os quatro algoritmos.

Pode-se ainda observar nas figuras que a variação do número de atributos no contexto influencia menos o desempenho dos algoritmos que utilizam diretamente o reticulado conceitual para a extração de dependências funcionais, ou seja, tanto o Aprem-IR quanto o *Find Implications* obtiveram tempos de execução semelhantes para 7 e 10 atributos.

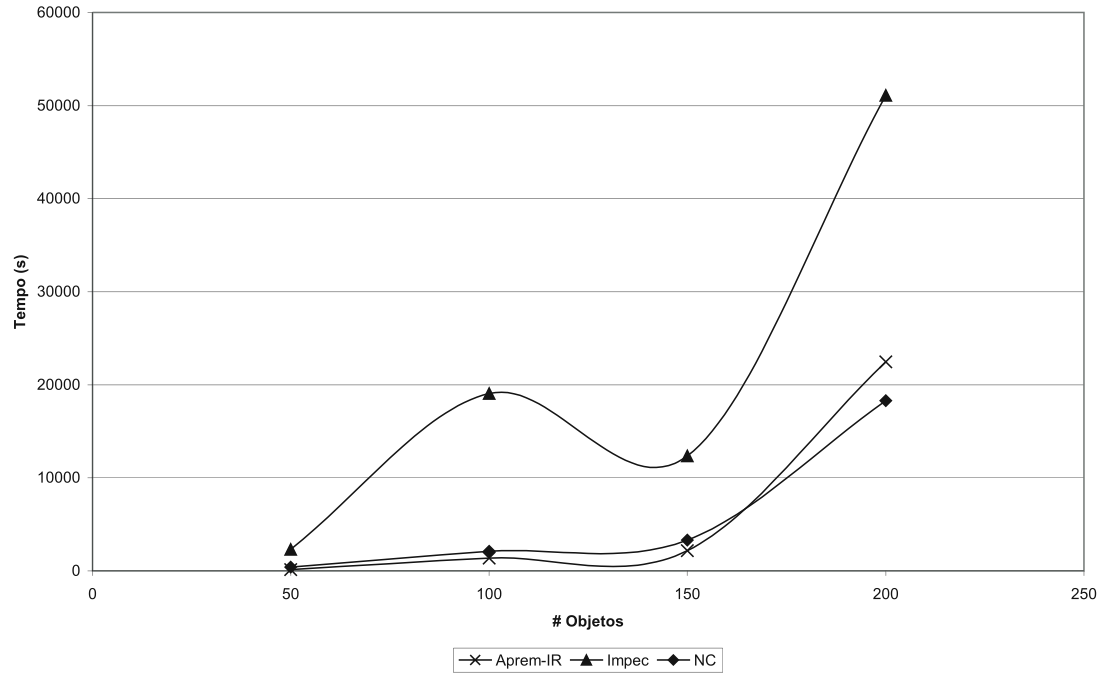

Figura 5: Desempenho dos algoritmos para extrair dependências funcionais variando número de objetos e utilizando op. de particionamento. Número de atributos 10 e 10 valores por atributo.

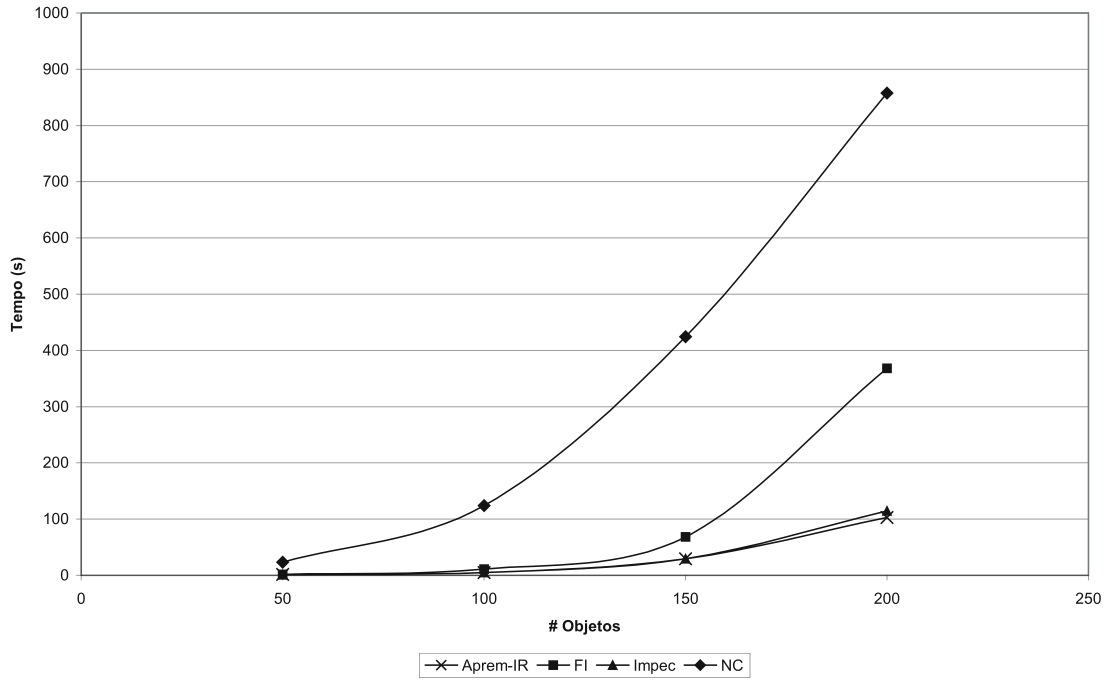

Figura 6: Desempenho dos algoritmos para extrair dependências funcionais variando número de objetos e transformando contextos. Número de atributos 7 e 7 valores por atributo.

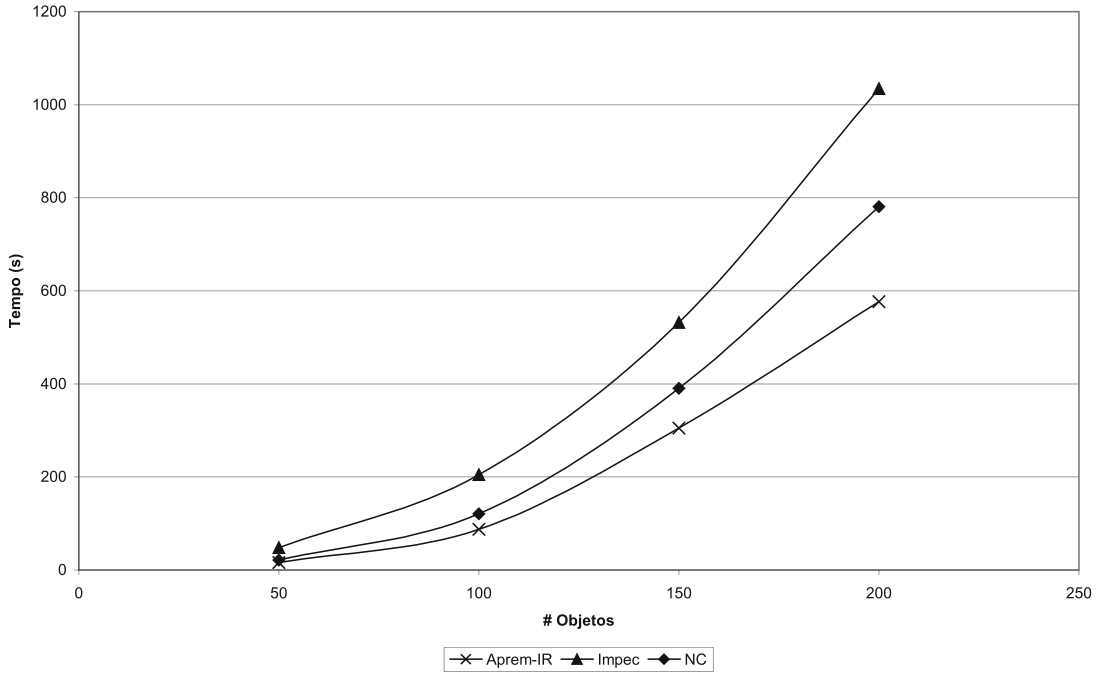

Figura 7: Desempenho dos algoritmos para extrair dependências funcionais variando número de objetos e utilizando op. de particionamento. Número de atributos 7 e 7 valores por atributo.

4 *Regras Probabilísticas*

4.1 Regras de Associação

O avanço da tecnologia facilitou o processo de coleta e armazenagem de dados provenientes de diversas fontes. Esta facilidade no tratamento dos dados acarretou o aumento da quantidade armazenada, tornando inexeqüível a análise manual destes para a geração de informações que auxiliem o processo de tomada de decisões dentro das organizações. Então, tornam-se necessários mecanismos automáticos que auxiliem o processo de tomada de decisões dentro das organizações.

A mineração de dados possui papel importante para auxiliar naquele processo já que fornece mecanismos para analisar os dados dentro de uma empresa. Ela fornece técnicas que, por exemplo, permitem a uma loja analisar seu histórico de vendas, verificando produtos que, normalmente, são vendidos juntos e, assim, criar promoções para o aumento de vendas futuras. Este foi o problema apontado por Agrawal e seus co-autores [1] ao introduzirem a idéia de regras de associação.

As regras de associação são implicações entre atributos de um banco de dados que são válidas apenas para grupos de elementos. Uma regra de associação descreve, por exemplo, o relacionamento: 80% dos clientes que compraram um televisor e um aparelho de DVD, também compraram um *home theater*. Neste caso, a regra de associação é *televisor e aparelho de DVD implica* home theater *com 80% de confiança*. *Televisor* e *aparelho de DVD* são o **antecedente** da regra, enquanto, *home theater* é o **conseqüente**. A **confiança** indica o quanto a regra é válida para o banco de dados do qual ela foi obtida, ou seja, no histórico de vendas, 80% dos clientes que compraram televisores e aparelhos de DVD, também compraram *home theaters*. Observe que as regras de associação assemelham-se às implicações, pois revelam relacionamentos entre atributos do banco de dados. Porém, estes dois tipos de regras diferem na questão da confiança. As implicações são válidas, sempre, para 100% dos dados. Já as regras de associação admitem exceções para a regra, o que levou-se a introduzir o parâmetro confiança à regra. Ao se obter as regras de associação de um banco de dados, pode-se desejar apenas aquelas com uma predeterminada confiança mínima. Assim, em geral, os algoritmos para extração de regras de

associação admitem que o usuário informe qual a confiança mínima das regras que ele deseja obter.

De acordo com a definição de confiança no exemplo anterior, ela indica que 80% dos clientes que compraram televisores e aparelhos de DVD, também compraram *home theaters*. Ela relaciona-se diretamente com os clientes que compraram televisores, aparelhos de DVD e *home theaters*. Porém, o quanto esta regra é representativa para o banco de dados? O quanto a venda de televisores, aparelhos de DVD e *home theaters* é representativa em todo o histórico de vendas? A esta questão, respondeu-se introduzindo mais um parâmetro às regras de associação. Foi introduzido o denominado **suporte** que revela, para o exemplo, quantos clientes compraram televisores, aparelhos de DVD ou *home theaters* do total de clientes. Generalizando, o suporte revela quantos elementos do banco de dados atendem àquela regra, ou seja, quantos possuem os atributos relacionadas à regra. Da mesma forma que com a confiança, pode-se informar o suporte mínimo para a extração das regras de associação.

Formalmente, as regras de associação são quádruplas ordenadas $(X, Y, \mathit{suporte}, \mathit{confiança})$ em que X e Y são subconjuntos disjuntos de atributos e o suporte e a confiança são números reais entre 0 e 1 que indicam, respectivamente, a probabilidade dos elementos de um banco de dados possuírem os atributos de X e de Y e a probabilidade condicional dos elementos do banco de dados possuírem os atributos de Y, dado que eles possuem os atributos de X. Normalmente, utiliza-se a notação $X \rightarrow Y_{(\mathit{suporte,confiança})}$ para representar a regra de associação $(X, Y, \mathit{suporte}, \mathit{confiança})$. Seja (G, M, I) um contexto formal. No contexto da AFC, a definição de regra de associação descrita acima pode ser reescrita da seguinte forma: (X, Y, *suporte*, *confiança*) é uma regra de associação tal que $X, Y \subset M$, $X \cap Y = \emptyset$, o suporte e a confiança são obtidos como abaixo.

$$\mathit{suporte}(X,Y) = \frac{|(X \cup Y)'|}{|G|},$$

$$\mathit{confiança}(X,Y) = \frac{|(X \cup Y)'|}{|X'|}$$

O problema da extração de regras de associação de bancos de dados é, em geral, dividido em dois subproblemas:

1 Encontrar combinações (subconjuntos) de atributos com suporte mínimo, chamados de conjuntos de itens freqüentes[1].

2 O segundo subproblema consiste em extrair, dos conjuntos de itens freqüentes, as regras de associação.

[1] Em inglês, *frequent itemsets*.

Tabela 9: Exemplo de contexto para regras de associação

Exemplo	A	B	C	D	E
1	×		×	×	
2		×	×		×
3	×	×	×		×
4		×			×
5	×	×	×		×

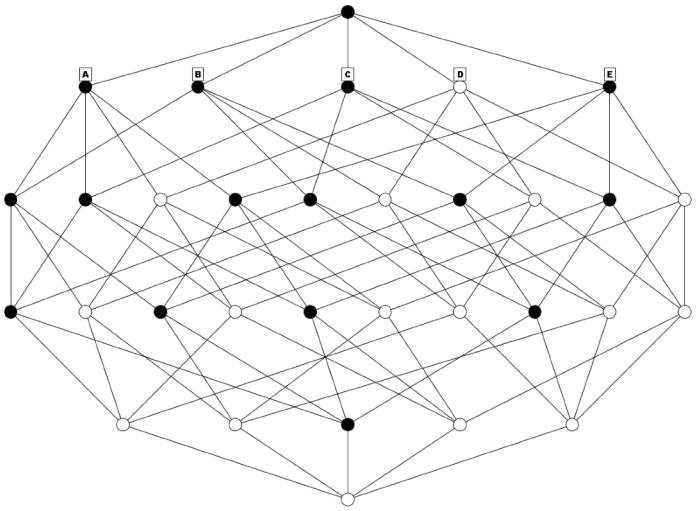

Figura 8: Reticulado $\wp(M)$

O primeiro problema consiste em encontrar o conjunto $CIF = \{X \subseteq M | sup(X) \geq min\_sup\}$, sendo que M é o conjunto de atributos do contexto (G,M,I), $\sup(X) = |X'|/|G|$ é o suporte de um conjunto de atributos e *min_sup* é o suporte mínimo definido pelo usuário. Este problema é computacionalmente caro, pois, o espaço de soluções é equivalente ao conjunto potência dos atributos, considerando-se que toda combinação de atributos é, potencialmente, um conjunto de itens freqüentes. A solução consiste em percorrer o reticulado de subconjuntos de M verificando os conjuntos com suporte acima do requerido. Exemplificando, seja (G,M,I) o contexto da Tabela 9. O reticulado apresentado na Figura 8 ilustra o espaço de soluções do problema de busca por conjuntos de itens freqüentes no contexto da Tabela 9. Os conjuntos freqüentes (com suporte mínimo igual a 40%) estão destacados. Nesse reticulado apenas os conjuntos contendo apenas um atributo foram rotulados, assim, para saber o conjunto de atributos representado em um nó, basta unir os atributos dos nós superiores.

Observe que para confirmar se um subconjunto de atributos é freqüente, deve-se verificar todos os objetos que possuam os atributos do conjunto, o que torna o processo de descoberta de conjuntos de itens freqüentes ainda mais caro. Felizmente, os algoritmos podem, com certa eficiência, computar os conjuntos de itens freqüentes utilizando-se as proposições 6 e 7 apre-

sentadas a seguir:

Proposição 6 (Propriedade 1 em [46])**.** *Todo subconjunto de um conjunto de itens freqüentes é freqüente.*

Demonstração. Seja $X \in CIF$. Seja $Y \subseteq X$. Pela propriedade 1 dos operadores de derivação, $Y' \supseteq X'$. Como $Y' \supseteq X'$, $|Y'| \geq |X'|$. Logo $sup(Y) \geq sup(X)$. Como $sup(X) \geq min\_sup$ e $sup(Y) \geq sup(X)$, $sup(Y) \geq min\_sup$. Portanto, Y é freqüente. □

Proposição 7 (Propriedade 2 em [46])**.** *Todo superconjunto de um conjunto de itens não-freqüentes é não-freqüente.*

Demonstração. Seja $X \notin CIF$. Seja $Y \supseteq X$. Pela propriedade 1 dos operadores de derivação, $Y' \subseteq X'$. Como $Y' \subseteq X'$, $|Y'| \leq |X'|$. Logo $sup(Y) \leq sup(X)$. Como $sup(X) \leq min\_sup$ e $sup(Y) \leq sup(X)$, $sup(Y) \leq min\_sup$. Portanto, Y é não-freqüente. □

Estas proposições podem ser visualmente constatadas observando-se o reticulado da Figura 8. Nele todo superconjunto do conjunto de itens não-freqüentes $\{D\}$ também é não-freqüente. Já os subconjuntos do conjunto de itens freqüentes $\{A,B,C,E\}$(rótulo do nó marcado mais inferior na Figura 8) são todos freqüentes.

Proposição 8. *Seja* $CIFM = \{X \in CIF | \forall Y \in CIF[Y \supseteq X \rightarrow Y = X]\}$ *a família dos conjuntos de itens freqüentes máximos. A família dos conjuntos de itens freqüentes é igual à família de subconjuntos dos elementos de* $CIFM$ *(*$\{\wp(X)|X \in CIFM\} = CIF$*).*

Demonstração.

($\subseteq$) Seja $X \in \{\wp(Y)|Y \in CIFM\}$. Logo, existe $Z \in CIFM$ tal que $X \subseteq Z$. Como Z é freqüente, pela proposição 6, X é freqüente. Portanto, $X \in CIF$.

($\supseteq$) Seja $X \in CIF$. Assim, X pode, ou não, pertencer a $CIFM$. Caso X pertença a $CIFM$ não resta nada a demonstrar. Então, considere o caso em que X não pertença a $CIFM$. Neste caso, existe $Y \in CIFM$ tal que $X \subseteq Y$. Logo, $X \in \{\wp(Z)|Z \in CIFM\}$. Portanto, $CIF \subseteq \{\wp(Z)|Z \in CIFM\}$.

□

Os algoritmos podem, ainda, utilizar a proposição 8 para reduzir o espaço de busca de soluções, uma vez que encontrado o conjunto dos itens freqüentes máximos pode-se derivar o conjunto dos itens freqüentes.

Até agora, foram apresentadas propriedades que possibilitam aos algoritmos encontrar com maior eficiência conjuntos de itens freqüentes. No entanto, não se mostrou como o uso de reticulados conceituais pode contribuir para a solução do problema. À frente, será demonstrado que o conjunto dos itens freqüentes máximos é igual ao conjunto das máximas intensões dos conceitos freqüentes. Porém, anteriormente, algumas definições serão feitas.

Um conceito formal (X,Y) é freqüente, se o número de objetos em sua extensão é maior ou igual ao mínimo requerido. Assim, (X,Y) é freqüente, se $|X|/|G| \geq min\_sup$[2]. A família dos conceitos formais freqüentes é definida como a seguir:

$$CF = \{(X,Y) \in \mathfrak{B}(G,M,I) | sup(Y) \geq min\_sup\}$$

Sendo $\mathfrak{B}(G,M,I)$ o reticulado conceitual do contexto formal (G,M,I).

Pode-se definir o conjunto dos conceitos freqüentes mínimos. Note que o conjunto do conceitos freqüentes mínimos dá origem ao conjunto das máximas intensões devido à relação de ordem definida sobre os conceitos.

$$CFM = \{(X,Y) \in CF | \forall (X_1,Y_1) \in CF[(X_1,Y_1) \leq (X,Y) \rightarrow (X_1,Y_1) = (X,Y)]\}$$

Antes de demonstrar que conjunto dos itens freqüentes máximos é igual ao conjunto das máximas intensões dos conceitos freqüentes, é necessária a apresentação do lema 2.

Lema 2 (Propriedade 5 em [46])**.** *O suporte de um conjunto de atributos C é igual ao suporte do maior conceito* (X,Y) *tal que* $C \subseteq Y$.

Demonstração. Seja $C \subseteq M$. O suporte de C em (G,M,I) é: $sup(C) = |C'|/|G|$. O maior conceito com C em sua intensão é: (C',C''). O suporte de (C',C'') é $|C'''|/|G|$. Como, pela propriedade 3 dos operadores de derivação, $C''' = C'$, o suporte de (C',C'') é igual a $|C'''|/|G| = |C'|/|G|$. Portanto, $sup(C) = sup(C'')$. □

Teorema 2 (Propriedade 6 em [46])**.** *Seja* $ICFM = \{Y | (X,Y) \in CFM\}$ *o conjunto das intensões dos conceitos freqüentes mínimos. O conjunto ICFM é igual ao conjunto dos itens freqüentes máximos.*

Demonstração. Para demonstrar que $ICFM = CIFM$, basta mostrar que os elementos de $CIFM$ são intensões de conceitos no reticulado conceitual. Seja $Y \in CIFM$. Pela propriedade 2 dos operadores de derivação, $Y \subseteq Y''$. Pelo lema 2, $sup(Y'') = sup(Y) \geq min\_sup$. Como Y é máximo, conclui-se que $Y = Y''$. Portanto, $Y \in ICFM$. □

[2] O suporte de (X,Y) é igual a $|X|/|G|$ pois, pela definição de conceito formal, $X = Y'$.

O teorema 2 demonstra que o conjunto das intensões dos conceitos freqüentes mínimos é igual ao conjunto dos itens freqüentes máximos. Dessa forma, o reticulado conceitual pode ser usado para encontrar o conjunto de itens freqüentes com maior eficiência, já que o conjunto das intensões dos conceitos é subconjunto do conjunto de itens ($\wp(M)$).

O segundo problema consiste em: a partir de cada conjunto de itens freqüentes W, encontrar regras do tipo $X \rightarrow W - X$ em que $X \subset W$. Obviamente, a regra possui suporte acima do requerido já que W é freqüente. Para verificar a validade da regra, basta verificar se a confiança está acima do mínimo requerido. A confiança da regra pode ser obtida através da razão entre o suporte de W e o suporte de X. Este também é um problema computacionalmente custoso pois, para cada conjunto de itens freqüentes, existem $2^k - 2$ potenciais regras válidas (k é o número de atributos no conjunto) uma vez que devem ser verificadas todas as combinações de atributos no antecedente, exceto os conjuntos vazio e o próprio conjunto de itens freqüentes.

Mostrado como reticulados conceituais podem ser úteis para a identificação de regras de associação, nas seções subseqüentes serão apresentados os algoritmos baseados em AFC para tal propósito.

4.1.1 *AClose*

O algoritmo *AClose* foi proposto por Nicolas Pasquier e seus co-autores [45]. Os autores propuseram um algoritmo para a identificação dos conjuntos de itens freqüentes através do uso de conceitos freqüentes. Para a geração efetiva das regras através dos itens freqüentes, o algoritmo utiliza uma versão modificada do algoritmo utilizado no *Apriori* [3].

O pseudo-código do algoritmo para a identificação dos conceitos freqüentes é apresentado em Algoritmo 10. O algoritmo encontra as intensões dos conceitos freqüentes gradualmente, utilizando a idéia de geradores. Os geradores são os menores conjuntos de atributos que dão origem a uma intensão através da aplicação do operador de fecho $(\cdot)''$. Considere o reticulado do contexto da Tabela 9, o conjunto $\{B\}$ é um gerador para a intensão do conceito $(\{2,3,4,5\},\{B,E\})$ pois $B'' = BE$. Na i-ésima etapa são avaliados os geradores de tamanho i. O algoritmo inicia com o conjunto dos elementos de M (linha 1). Em seguida, enquanto existirem geradores a serem avaliados o algoritmo mantém-se no ciclo das linhas 2 à 13. O algoritmo atribui, inicialmente, o conjunto vazio a cada conjunto fechado referente a um gerador, assim como atribui zero ao suporte de cada conjunto fechado (linhas 3 e 4). A seguir, o algoritmo determina o conjunto fechado e seu respectivo suporte para cada gerador com o auxílio da função *determinarConjuntosFechados* (linha 5). Depois, o algoritmo avalia cada candidato a intensão freqüente armazenando aqueles com suporte acima do suporte mínimo (linhas 6 à 10). O próximo passo do algoritmo é encontrar os geradores de tamanho i+1. Na linha 11, com o

auxílio da função *encontrarGeradores*, o algoritmo determina os novos geradores. Repare que são utilizados apenas os conjuntos freqüentes para determinar os novos candidatos, pois, como demonstrado na proposição 7, os conjuntos originários de conjuntos não-freqüentes também são não-freqüentes. Por fim, o algoritmo retorna as intensões dos conceitos freqüentes encontradas.

Algoritmo AClose
Entrada: Um contexto formal (G,M,I)
Saída: O conjunto ICF das intensões dos conceitos freqüentes
início
1. $i := 1$
2. $FCC_i.geradores := \{\{m\} | m \in M\}$
3. **enquanto** $FCC_i.geradores \neq \emptyset$ **faça**
4. atribuir conjunto vazio aos fechos de FCC_i
5. atribuir 0 aos suportes FCC_i
6. $FCC_i := determinarConjuntosFechados(FCC_i)$
7. **para cada** $c \in FCC_i$ **faça**
8. **se** $c.suporte \geq min\_sup$ **então**
9. $ICF_i := ICF_i \cup \{c\}$
10. **fim se**
11. **fim para**
12. $FCC_{i+1} := encontrarGeradores(ICF_i)$
13. $i := i+1$
14. **fim enquanto**
15. **retorne** $\bigcup_{j=1}^{i-1} ICF_j$

fim

Algoritmo 10: Algoritmo *AClose*

Para calcular com eficiência os conjuntos fechados dos geradores de FCC_i, a função *determinarConjuntosFechados* utiliza a seguinte propriedade do operador de fecho $(\cdot)''$. Seja $Y \subseteq M$. Então, $Y'' = \bigcap \{g' | g \in G \wedge Y \subseteq g'\}$. Dessa forma, percorrendo o contexto formal uma única vez é possível calcular não só o conjunto fechado de cada gerador mas também seu suporte. O Algoritmo 11 apresenta o pseudo-código da função para calcular os conjuntos fechados.

O Algoritmo 11 funciona da seguinte forma: para cada objeto g do contexto formal (linhas 1 à 11), apenas os geradores subconjuntos do conjunto de atributos do objeto g são atualizados (linha 2). Caso os fechos desses geradores sejam o conjunto vazio, então a eles é atribuído o conjunto de atributos de g (linha 5). Caso contrário, o fecho é atualizado fazendo a interseção do seu estado atual com o conjunto de atributos do objeto (linha 7). Em seguida, o suporte do gerador é incrementado. Finalmente, apenas os geradores com fecho não-vazio são retornados. Geradores com fecho vazio podem ser descartados pois não serão utilizados na geração de regras.

Algoritmo determinarConjuntosFechados
Entrada: Um conjunto de geradores FCC_i
Saída: O conjunto FCC_i com os conjuntos fechados e suporte calculados
início
1. **para cada** $g \in G$ **faça**
2. $Subsets := \{ger \in FCC_i.geradores | ger \subseteq g'\}$
3. **para cada** $s \in Subsets$ **faça**
4. **se** $s.fecho = \emptyset$ **então**
5. $s.fecho := g'$
6. **senão**
7. $s.fecho := s.fecho \cap g'$
8. **fim se**
9. $s.suporte := s.suporte + 1$
10. **fim para**
11. **fim para**
12. **retorne** $\{s \in FCC_i | s.fecho \neq \emptyset\}$

fim

Algoritmo 11: Função para determinar conjuntos fechados.

A função *encontrarGeradores* é responsável por determinar os geradores de tamanho i+1. Esta função recebe como parâmetro o conjunto dos geradores freqüentes de tamanho i. Para determinar os geradores de tamanho i+1, a função *encontrarGeradores* utiliza as proposições 9 e 10 retiradas de [45]. Além disso, considera-se uma ordem total sobre os atributos de M. O Algoritmo 12 apresenta o pseudo-código da função *encontrarGeradores*. O algoritmo inicia determinando os novos geradores (linha 1). Os geradores de tamanho i+1 são encontrados unindo-se dois geradores freqüentes p e q de tamanho i com os mesmos i-1 elementos. Em seguida, remove-se os geradores originários de conjuntos não-freqüentes (linhas 2 à 8). Geradores originários de conjuntos não-freqüentes são geradores que possuem subconjunto não pertencente a ICF_i. A segunda poda no conjunto FCC_{i+1} é feita aplicando-se a proposição 10 (linhas 9 à 16). Cada novo gerador é verificado e aqueles a que a proposição 10 aplica-se, isto é, aqueles que existe algum subconjunto cujo fecho é superconjunto do gerador, são removidos do conjunto de novos geradores. Após as duas podas, o conjunto de novos geradores é retornado.

Proposição 9 ([45] lema 2)**.** *Seja* $Y_1 \subseteq M$ *e* $Y_2 \subseteq Y_1$ *tal que* $Y_1 \subseteq Y_2''$*. Então,* $Y_1'' = Y_2''$ *e, para todo* $Y_3 \subseteq M$*,* $(Y_1 \cup Y_3)'' = (Y_2 \cup Y_3)''$*.*

Proposição 10 ([45] corolário 2)**.** *Seja ger um gerador de tamanho i e* $S = \{s \subseteq ger | |s| = (i-1)\}$ *tal que* $ger = \bigcup S$*. Se existe* $s \in S$ *tal que* $ger \subseteq s''$*, então* $ger'' = s''$*.*

Encontrada as intensões dos conceitos freqüentes, elas podem ser utilizadas para encontrar o conjunto dos itens freqüentes e, em seguida, as regras de associação válidas para o contexto formal.

Algoritmo encontrarGeradores
Entrada: Um conjunto de geradores freqüentes de tamanho i ICF
Saída: O conjunto dos geradores de tamanho i+1 FCC
início

```
                    /* Determina novos geradores */
 1.   FCC := {p∪q | p,q ∈ ICF ∧ ∀j < i[p_j = q_j] ∧ p_i < q_i}
                    /* Remove geradores originários de conjuntos não freqüentes */
 2.   para cada ger ∈ FCC faça
 3.     para cada s ∈ {s ⊆ p || s| = i} faça
 4.       se s ∉ ICF.geradores então
 5.         FCC := FCC − {s}
 6.       fim se
 7.     fim para
 8.   fim para
                    /* Aplica proposição 10 e remove geradores com fechos já calculados */
 9.   para cada ger ∈ FCC faça
10.     Subsets := {s ∈ ICF.geradores | s ⊆ ger}
11.     para cada s ∈ Subsets faça
12.       se ger ⊆ s.fecho então
13.         FCC := FCC − {s}
14.       fim se
15.     fim para
16.   fim para
17.   retorne FCC
fim
```

Algoritmo 12: Função para determinar geradores de tamanho i+1

Para encontrar o conjunto dos itens freqüentes a partir das intensões freqüentes, basta, para cada intensão freqüente, encontrar todos os seus subconjuntos. Esse é, basicamente, o funcionamento do Algoritmo 13. O algoritmo inicialmente particiona o conjunto das intensões freqüentes em função do tamanho e descobre qual o tamanho da maior intensão freqüente (linhas 1 a 5). Em seguida, cada conjunto de itens freqüentes de tamanho i-1 é completado com os subconjuntos dos itens freqüentes de tamanho i; iniciando-se o processo pelos conjuntos de tamanho k (linhas 6 à 15). Sobre cada conjunto de itens freqüentes CIF_i, mantém-se uma ordem sobre seus elementos: as intensões freqüentes vêm sempre à frente dos conjuntos derivados. Essa ordem é mantida para que o suporte de um conjunto de itens seja sempre atribuído de forma correta, ou seja, o suporte de um conjunto de itens seja igual ao da menor intensão de que ele é subconjunto. Assim, avalia-se cada conjunto de itens freqüentes (linhas 7 à 14). Para cada conjunto freqüente I, verificam-se todos os seus subconjuntos de tamanho i-1, adicionando-os ao CIF_{i-1}. O suporte de cada conjunto adicionado é igual ao suporte do conjunto que lhe originou que, por sua vez, é igual ao suporte da menor intensão que ele é subconjunto. Finalmente, o algoritmo retorna todos os conjuntos de itens freqüentes derivados.

Algoritmo derivarItensFreqüentes
Entrada: Um conjunto de intensões freqüentes ICF
Saída: O conjunto dos itens freqüentes
início
1. $k := 0$
2. **para cada** $I \in ICF$ **faça**
3. $CIF_{|I|} := CIF_{|I|} \cup \{I\}$
4. $k := max(k, |I|)$
5. **fim para**
6. **para** i de k até 1 **faça**
7. **para cada** $I \in CIF_i$ **faça**
8. **para cada** $s \in \{X \subseteq I || X| = i-1\}$ **faça**
9. **se** $s \notin CIF_{i-1}$ **então**
10. $s.suporte := I.suporte$
11. $CIF_{i-1} := CIF_{i-1} \cup \{s\}$
12. **fim se**
13. **fim para**
14. **fim para**
15. **fim para**
16. **retorne** $\bigcup_{i=1}^{k} CIF_i$

fim

Algoritmo 13: Deriva os conjuntos de itens freqüentes.

Obtido o conjunto dos itens freqüentes, pode-se encontrar o conjunto de regras de associação válidas para o contexto formal. O algoritmo para geração das regras de associação usado por Pasquier e seus co-autores [45] é uma adaptação do algoritmo *Apriori* apresentado por Agrawal [3]. Seu princípio é bastante simples: para cada conjunto $I \in CIF$, derivam-se todos os subconjuntos $I_2 \subseteq I$ e verifica-se a confiança da regra $I_2 \rightarrow I - I_2$ é maior que a confiança mínima, ou seja, verifica-se se $sup(I_1)/sup(I_2) \geq min\_conf$. Em caso positivo, a regra é armazenada. Observe que se a regra $I_2 \rightarrow I - I_2$ não possui confiança acima do mínimo, então todo subconjunto $I_3 \subseteq I_2$ também produzirá regra com confiança abaixo do mínimo já que $sup(I_3) \geq sup(I_2)$. Utilizando essa propriedade, o algoritmo primeiro encontra as regras cujo conseqüente tem tamanho um. Em seguida, para os antecedentes que originaram regras com confiança acima do mínima, explora as regras com conseqüente de tamanho 2 e assim por diante.

O Algoritmo 14 apresenta o pseudo-código do algoritmo para encontrar as regras válidas. O algoritmo verifica todos os conjuntos de itens freqüentes para a geração de regras válidas (linhas 1 à 12). Para cada conjunto de itens freqüente, gera-se, inicialmente, o conjunto dos conseqüentes contendo apenas um atributo (linha 2). Avalia-se, então, a confiança da regra a ser obtida com cada um desses conseqüentes gerados (linha 4 e 5). Se a confiança está acima do mínimo requerido, então uma nova regra é gerada (linha 6). Caso contrário, o conseqüente é excluído do conjunto de conseqüentes uma vez que de todos os superconjuntos deste conseqüente serão obtidas regras com confiança abaixo do mínimo (linha 8). O procedimento *gerarRegras* é, então, chamado para gerar as regras com conseqüentes de tamanho i+1. O funcionamento

do procedimento *gerarRegras* é semelhante ao funcionamento do algoritmo principal. No entanto, os conseqüentes de tamanho i+1 são obtidos através da função *Apriori_Gen* (linha 2 - procedimento *gerarRegras*). O pseudo-código dessa função é apresentado em Algoritmo 15.

A função recebe como parâmetro um conjunto de conseqüentes de tamanho m e retorna o conjunto de conseqüentes de tamanho m+1. Inicialmente, o conjunto de conseqüentes de tamanho m+1, H_{m+1}, é gerado combinando-se dois conseqüentes com os mesmos m-1 elementos (linha 1). Em seguida, removem-se os conseqüentes de tamanho m+1 superconjuntos de conseqüentes de tamanho m que produziram regras com confiança abaixo do mínimo (linhas 2 à 6). Após essa poda, os novos conseqüentes são retornados.

Algoritmo Gerar Regras Válidas
Entrada: Uma família CFI de conjuntos de itens freqüentes
Saída: O conjunto das regras de associação válidas RA
início
1. **para cada** $I \in CFI$ tal que $|I| \geq 2$ **faça**
2. $H_1 := \{\{m\} | m \in I\}$
3. **para cada** $h_1 \in H_1$ **faça**
4. $conf := sup(I)/sup(I - h_1)$
5. **se** $conf \geq min\_conf$ **então**
6. $RA := RA \cup \{I - h_1 \rightarrow h_1(sup(I), conf)\}$
7. **senão**
8. $H_1 := H_1 - \{h_1\}$
9. **fim se**
10. **fim para**
11. $gerarRegras(I, H_1)$
12. **fim para**
fim

procedimento gerarRegras
Entrada: Um conjunto I de itens freqüentes de tamanho k e um conjunto H_m de conseqüentes de tamanho m
Saída: O conjunto das regras de associação válidas RA
início
1. **se** $k > m$ **então**
2. $H_{m+1} := Apriori\_Gen(H_m)$
3. **para cada** $h_{m+1} \in H_{m+1}$ **faça**
4. $conf := sup(I)/sup(I - h_{m+1})$
5. **se** $conf \geq min\_conf$ **então**
6. $RA := RA \cup \{I - h_{m+1} \rightarrow h_{m+1}(sup(I), conf)\}$
7. **senão**
8. $H_{m+1} := H_{m+1} - \{h_{m+1}\}$
9. **fim se**
10. **fim para**
11. $gerarRegras(I, H_{m+1})$
12. **fim se**
fim

Algoritmo 14: Gera regras de associação válidas.

Algoritmo Apriori_Gen
Entrada: Um H_m de conseqüentes
Saída: O conjunto H_{m+1} de conseqüentes de tamanho m+1
início
1. $H_{m+1} := \{p \cup q | p,q \in H_m \wedge \forall j < m[p_j = q_j] \wedge p_m < q_m\}$
2. **para cada** $h_{m+1} \in H_{m+1}$ **faça**
3. **se** $\{s \subseteq h_{m+1} || s| = m\} \not\subseteq H_m$ **então**
4. $H_{m+1} := H_{m+1} - \{h_{m+1}\}$
5. **fim se**
6. **fim para**
fim

Algoritmo 15: Gera novos conseqüentes.

4.1.2 *Titanic*

O algoritmo *Titanic* foi proposto por Gerd Stumme e seus co-autores [50]. O algoritmo baseia-se no algoritmo *Apriori* [3] para a construção do reticulado dos conceitos freqüentes chamado de reticulado *iceberg*.

O reticulado *iceberg* é um supremo-semi-reticulado dos conceitos freqüentes de um contexto formal. Este reticulado foi inicialmente proposto como um método para agrupamento de dados. No entanto, como do conjunto das intensões freqüentes pode-se derivar o conjunto de itens freqüentes, ele foi utilizado para a extração de regras de associação.

Para obter o reticulado *iceberg*, o algoritmo *Titanic* evita ao máximo o uso do operador de fecho $(\cdot)''$, utilizando uma função de peso compatível com ele. Antes de apresentar o algoritmo, algumas definições são necessárias.

Uma função de peso p é uma função definida sobre o conjunto potência do conjunto de atributos para uma ordem total $\langle P, \leq \rangle$, ou seja, $p : \wp(M) \rightarrow P$. O resultado da aplicação de p a um conjunto $X \subseteq M$ ($p(X)$) é chamado de peso de X. A função p é compatível com o operador de fecho $(\cdot)''$ se ela respeitar as propriedades discutidas na proposição 11.

Proposição 11. *Seja $p : \wp(M) \rightarrow P$ uma função de peso e seja $(\cdot)'' : \wp(M) \rightarrow \wp(M)$ um operador de fecho. A função p é compatível com o operador $(\cdot)''$ se ela respeitar as seguintes propriedades:*

$$X \subseteq Y \rightarrow p(X) \geq p(Y) \tag{11.a}$$

$$X'' = Y'' \rightarrow p(X) = p(Y) \tag{11.b}$$

$$X \subseteq Y \wedge p(X) = p(Y) \rightarrow X'' = Y'' \tag{11.c}$$

Pode-se determinar o fecho de um conjunto $X \subseteq M$ comparando-se seu peso com o peso dos seus superconjuntos próprios imediatos conforme apresentado na proposição 12.

Proposição 12. *Seja $X \subseteq M$. Então, $X'' = X \cup \{m \in M - X | p(X) = p(X \cup \{m\})\}$.*

Demonstração.

($\subseteq$) A prova será por contradição. Suponha que exista $m \in X'' - X$ tal que $p(X) \neq p(X \cup \{m\})$. Então, $X'' \neq (X \cup \{m\})''$ pela propriedade (11.b). Logo, $m \notin X''$. Contradição!

($\supseteq$) Seja $m \in X \cup \{m \in M - X | p(X) = p(X \cup \{m\})\}$. Se $m \in X$, então não resta nada a demonstrar. Assim, suponha $m \in \{m \in M - X | p(X) = p(X \cup \{m\})\}$. Como $X \subseteq X \cup \{m\}$ e $p(X) = p(X \cup \{m\})$, conclui-se, pela propriedade (11.c), que $X'' = (X \cup \{m\})''$.

□

O conjunto dos fechos de M pode ser obtido através da proposição 12 aplicando-se a proposição a cada conjunto $X \in \wp(M)$. Entretanto, tal abordagem mostra-se bastante ineficiente uma vez que dois conjuntos $X_1, X_2 \subseteq M$ podem gerar o mesmo fecho e, assim, um fecho pode ser obtido diversas vezes. Alternativamente, pode-se particionar $\wp(M)$ em conjuntos cujos elementos possuam o mesmo fecho e, assim, computar o fecho de apenas um conjunto de cada partição. Essas partições são as classes de equivalência da relação de equivalência $E = \{(X,Y) \in \wp(M) \times \wp(M) | X'' = Y''\}$.

Como essas classes de equivalência só são conhecidas durante a computação dos fechos, eventualmente, pode-se computar o fecho de mais de um conjunto de cada classe. Assim, é necessária uma estratégia para que o fecho de cada classe seja computado o menor número de vezes possível. A estratégia adotada pelo algoritmo *Titanic* é a gradual. Os conjuntos de $\wp(M)$ são avaliados gradualmente. Os menores (ou mínimos caso não exista o menor em uma classe de equivalência) conjuntos (com relação à ordem $\subseteq$) de cada classe de equivalência são avaliados primeiro.

Os menores (mínimos) conjuntos de cada classe são chamados de conjuntos chave. Como cada classe de equivalência possui pelo menos um mínimo, para computar o conjunto dos fechos, é suficiente computar apenas o fecho do conjunto dos mínimos. Formalmente, sendo K o conjunto de todos conjuntos chave, a afirmativa é expressa pela equação (4.1).

$$\{X \subseteq M | X = X''\} = \{C'' | C \in K\} \tag{4.1}$$

A estratégia gradual adotada pelo algoritmo consiste em descobrir os conjuntos chave durante cada iteração. Durante a i-ésima iteração, os candidatos à chave de tamanho i são ava-

liados e somente aqueles cuja classe de equivalência não tenha sido verificada são considerados conjuntos chave e são mantidos para a geração de candidatos para a etapa i+1. Um conjunto $X \subseteq M$ com tamanho $|X| = i$ é um conjunto chave para a classe de equivalência $[X] = \{Y \subseteq M | (X,Y) \in E\}$, se o peso de X é diferente de todos os seus subconjuntos de tamanho i-1, ou seja, se $p(X) \neq p(Y)$ para todo $Y \subset X$ tal que $|Y| = i - 1$. Esta afirmativa é verificada pela propriedade (11.c).

Já se sabe que a computação dos fechos dos conjuntos chave é suficiente para encontrar o conjunto de todos os fechos. No entanto, até agora, não foi mostrado como são gerados os candidatos de tamanho i+1 a partir dos de tamanho i. Antes de elucidar como os candidatos de uma etapa seguinte são gerados, é necessária a apresentação da proposição 13 que será útil para compreender como os candidatos são gerados.

Proposição 13. *O conjunto K de conjuntos chave é uma seção inferior para* $\langle \wp(M), \subseteq \rangle$. *Se* $Y \in K$ *e* $X \subseteq Y$, *então* $X \in K$ *para todo* $X, Y \subseteq M$.

Conclui-se da proposição 13 que um conjunto $X \subseteq M$ é um candidato a conjunto chave se todo $Y \subseteq X$ for um conjunto chave ($Y \in K$). Como o conjunto K é uma seção inferior, por definição, se $X \in K$, então todo subconjunto $Y \subseteq X$ também pertence a K. Em outras palavras, todo superconjunto imediato de um conjunto chave é um candidato a conjunto chave. Formalmente, seja $Y \in K$ e $X \supset Y$ tal que $\forall Z \subseteq M[X \supset Z \supseteq Y \rightarrow Z = Y]$, então X é um candidato a conjunto chave. Logo, o conjunto de candidatos a conjunto chave de tamanho i+1 é

$$C = \{X \cup Y | X, Y \in K_i \wedge \forall j < i[X_j = Y_j]^3 \wedge X_i < Y_i \wedge \forall s \subseteq (X \cup Y)[s \in K_i]\} \tag{4.2}$$

em que K_i é o conjunto dos conjuntos chave de tamanho i.

Enfim pode-se apresentar o algoritmo *Titanic* que aplica as definições apresentadas anteriormente para obter as intensões dos conceitos frqüentes. O pseudo-código do algoritmo é apresentado em Algoritmo 16. A função de peso utilizada é o suporte de um conjunto de atributos. A proposição 14 demonstra que o suporte é compatível com o operador de fecho $(\cdot)''$.

Proposição 14. *Seja* $sup : \wp(M) \rightarrow [0,1]$ *a função de suporte tal que, para* $X \subseteq M$, $sup(X) = |X'|/|G|$. *A função sup é compatível com o operador de fecho* $(\cdot)''$.

Demonstração. Sejam $X, Y \subseteq M$. Para demonstrar que *sup* é compatível com o operador de fecho $(\cdot)''$, deve-se demonstrar que *sup* possui as propriedades apresentadas em 11.

1 $(X \subseteq Y \rightarrow sup(X) \geq sup(Y))$: Suponha que $X \subseteq Y$. Como $X \subseteq Y$, pela propriedade 1

[3] Supõe-se uma ordem total entre os elementos de X e Y. Logo, X_j refere-se ao j-ésimo elemento do conjunto X; o mesmo aplica-se ao conjunto Y.

dos operadores de derivação, $X' \supseteq Y'$ e, assim, $|X'| \geq |Y'|$. Portanto, $sup(X) = |X'|/|G| \geq |Y'|/|G| = sup(Y)$.

2 $(X'' = Y'' \rightarrow sup(X) = sup(Y))$: Suponha que $X'' = Y''$. Como $X'' = Y''$, pela propriedade 3 dos operadores de derivação, $X''' = X' = Y' = Y'''$. Portanto, $sup(X) = |X'|/|G| = |Y'|/|G| = sup(Y)$

3 $(X \subseteq Y \wedge sup(X) = sup(Y) \rightarrow X'' = Y'')$: Suponha que $X \subseteq Y$ e $sup(X) = sup(Y)$. Como $X \subseteq Y$, pela propriedade 1 dos operadores de derivação, $X' \supseteq Y'$. De $sup(X) = sup(Y)$, conclui-se que $|X'| = |Y'|$. Como $X' \supseteq Y'$, segue-se que $X'' = Y''$.

□

O algoritmo inicia com o conjunto de candidatos de tamanho 0 (conjunto vazio). Como já se sabe que o peso do conjunto vazio é sempre 1, evita-se a chamada da função *pesar* na linha 1. Em seguida, o conjunto vazio é incluído como primeiro conjunto chave (linha 2) pois, pela propriedade 13 , o conjunto vazio sempre pertence ao conjunto dos conjuntos chave. Seguindo, ao indicador de iteração, é atribuído o valor zero marcando o início da computação e os candidatos a conjunto chave de tamanho 1 são gerados (linha 4). Na linha 5, para cada candidato a conjunto chave $c \in C$, armazena-se o menor peso dos subconjuntos de tamanho $|c| - 1$ em $c.s\_p$. Este parâmetro é utilizado para eliminar candidatos que não sejam conjuntos chave (linha 9). O algoritmo entra em um ciclo (linhas 6 à 12) avaliando todos os candidatos a conjuntos chave. O algoritmo desconsidera conjuntos chave com fecho não-freqüente para a geração de novos candidatos já que, deles, originar-se-ão conjuntos não-freqüentes.

Durante a i-ésima iteração, o algoritmo calcula os fechos dos conjuntos chave de tamanho i (linha 7), calcula o peso(suporte) dos candidatos a conjunto chave para (i+1)-ésima iteração (linha 8), obtem os conjuntos chave de tamanho i+1 (linha 9) e calcula os novos candidatos a conjunto chave (linha 11).

Os candidatos a conjunto chave de tamanho i+1 são gerados através da função *Titanic_Gen*. Ela utiliza a equação (4.2) para gerar os novos candidatos. No entanto, aplica-se um filtro ao conjunto dos novos candidatos, excluindo-se aqueles que são superconjuntos de conjuntos chave cujos fechos são não-freqüentes. O algoritmo é apresentado em pseudo-código pelo Algoritmo 17.

O algoritmo começa obtendo o conjunto dos possíveis candidatos a conjunto chave (linha 1). Os possíveis candidatos são encontrados combinando-se conjuntos chave de tamanho i

Algoritmo Titanic
Entrada: Um contexto formal (G,M,I)
Saída: O conjunto das intensões dos conceitos freqüentes
início
1. $\{\emptyset\}.p := 1$
2. $K_0 := \{\emptyset\}$
3. $i := 0$
4. $C := \{\{m\} | m \in M\}$
5. $\forall c \in C[c.s\_p := \emptyset.p]$
6. **equanto** $\{X \in K_i | X.p \geq min\_sup\} \neq \emptyset$ **faça**
7. $calcularFechos(K_i)$
8. $pesar(C)$
9. $K_{i+1} := \{c \in C | c.p \neq c.s\_p\}$
10. $i := i+1$
11. $C := Titanic\_Gen(K_i)$
12. **fim enquanto**
13. **retorne** $\bigcup_{j=1}^{i-1}\{X.fecho | X \in K_j \wedge X.p \geq min\_sup\}$

fim

Algoritmo 16: *Titanic*.

de forma a gerar candidatos de tamanho i+1. Em seguida, verifica-se cada candidato (linhas 3 à 11). Para cada candidato k_{i+1}, constata-se se todos os seus subconjuntos de tamanho i pertecem ao conjunto dos conjuntos chave K_i e se nenhum de seus subconjuntos possui fecho não-freqüente (linhas 4 à 10). Caso um dos subconjuntos do candidato não seja um conjunto chave, ou possui fecho não-freqüente, o candidato é excluído já que ele não será um conjunto chave (pela proposição 13) ou possui fecho não-freqüente (pela propriedade 7). Após todos candidatos terem sido avaliados, aqueles que passaram pelo filtro são retornados.

Algoritmo Titanic_Gen
Entrada: Um conjunto K_i de conjuntos chave
Saída: O conjunto C de candidatos a conjunto chave de tamanho i+1
início
1. $C := \{p \cup q | p,q \in \{X \in K_i | X.p \geq min\_sup\} \wedge \forall j < i[p_j = q_j] \wedge p_i < q_i\}$
2. $\forall k_{i+1} \in C[k_{i+1}.s\_p := 1]$
3. **para cada** $k_{i+1} \in C$ **faça**
4. **para cada** $s \in \{s \subseteq k_{i+1} | |s| = i\}$ **faça**
5. **se** $s \notin K_i \vee s.p = -1$ **então**
6. $C := C - \{k_{i+1}\}$
7. **interromper para cada** s
8. **fim se**
9. $k_{i+1} := min(k_{i+1}.s\_p, s.p)$
10. **fim para**
11. **fim para**
12. **retorne** C

fim

Algoritmo 17: Gera candidatos a conjunto chave.

O cálculo dos fechos dos conjuntos chave é feito através da proposição 12. No entanto, utilizam-se as propriedades de monotonicidade crescente e extensividade (respectivamente i e ii da definição 4) dos operadores de fecho para melhorar a eficiência do algoritmo. Antes de aplicar a proposição 12, o algoritmo faz a união do próprio conjunto chave com o fecho dos seus subconjuntos imediatos. O pseudo-código do algoritmo é apresentado em Algoritmo 18.

O Algoritmo 18 inicia com um ciclo para calcular o fecho dos conjuntos chave (linhas 1 à 19). No entanto, como o interesse é apenas nos candidatos com fechos freqüentes, os candidatos com peso igual a -1 (candidatos com suporte abaixo do mínimo) não têm seu fecho calculado. O cálculo do fecho de um candidato inicia-se com o conjunto de atributos do próprio candidato aplicando-se a propriedade ii dos operadores de fecho (linha 3). Em seguida, pela propriedade i dos operadores de fecho, faz-se a união dos fechos dos subconjuntos com o fecho atual do candidato (linhas 4 à 6). Por fim, aplica-se a proposição 12 para os atributos que ainda não foram incluídos no fecho do candidato (linhas 7 à 16).

Algoritmo calcularFechos
Entrada: Um conjunto K_i de conjuntos chave
Saída: O K_i de conjuntos chave com seus fechos calculados
início

```
 1.  para cada X ∈ K_i faça
 2.    se X.p ≠ −1 então
 3.      Y := X
 4.      para cada m ∈ X faça
 5.        Y := Y ∪ (Y − {m}).fecho
 6.      fim para
 7.      para cada m ∈ M − Y faça
 8.        se X ∪ {m} ∈ C então
 9.          p := (X ∪ {m}).p
10.        senão
11.          p := min{k.p | k ∈ ⋃_{j=0}^{i} K_j ∧ k ⊆ (X ∪ {m})}
12.        fim se
13.        se p = X.p então
14.          Y := Y ∪ {m}
15.        fim se
16.      fim para
17.      X.fecho := Y
18.    fim se
19.  fim para
```

fim

Algoritmo 18: Calcula os fechos de um conjunto de conjuntos chave.

O peso de um conjunto de candidatos a conjunto chave C é calculado verificando-se o número de objetos do contexto formal que possuem os atributos do candidato. Para cada objeto $g \in G$ do contexto formal, verificam-se os candidatos cujos atributos são atributos de g e

incrementa-se o peso do candidato indicando que mais um objeto possui aqueles atributos. Se o número de objetos que possuem os atributos de um candidato é maior ou igual ao mínimo suporte, então o peso do candidato é igual ao suporte do candidato, caso contrário o peso é igual a -1. O algoritmo é apresentado em pseudo-código pelo Algoritmo 19.

Algoritmo pesar
Entrada: Um conjunto C de candidatos a conjunto chave
Saída: O C de candidatos a conjunto chave com seus pesos calculados
início
1. $\forall X \in C[X.p := 0]$
2. **para cada** $g \in G$ **faça**
3. $\forall X \in \{Y \in C | Y \subseteq g'\}[X.p := X.p + 1]$
4. **fim para**
5. **para cada** $X \in C$ **faça**
6. **se** $X.p/|G| \geq min\_sup$ **então**
7. $X.p := X.p/|G|$
8. **senão**
9. $X.p := -1$
10. **fim se**
11. **fim para**

fim

Algoritmo 19: Calcula os pesos de um conjunto de candidatos a conjunto chave.

Finalmente, encontrados os conceitos freqüentes, eles são utilizados para a geração dos itens freqüentes e geração das regras da mesma forma que o *AClose*.

4.1.3 *Frequent Next Neighbours*

O algoritmo *Frequent Next Neighbours* (FNN) foi proposto por Carpineto e Romano [14]. O algoritmo baseia-se nos trabalhos de Zaki [61, 62] e de Luxenburger [36]. O algoritmo constrói o reticulado conceitual com os conceitos freqüentes e usa-o como um guia para extração de regras de associação.

O FNN adapta o algoritmo *Next Neighbours* proposto por Bordat [11]. O algoritmo proposto por Bordat constrói o reticulado conceitual utilizando a relação de cobertura apresentada no capítulo 2. O algoritmo descobre, a cada iteração, os próximos conceitos (de acordo com a relação de cobertura) de cada um dos conceitos descobertos anteriormente; os conceitos descobertos são usados na próxima iteração para descobrir novos conceitos. O algoritmo inicia com o conceito (G, G'), descobre seus sucessores no reticulado e, em seguida, repete-se o processo para os sucessores e, assim, sucessivamente até que todos os conceitos tenham sido encontrados.

A adaptação do FNN em relação ao *Next Neighbours* é que o FNN, a cada iteração, considera apenas os conceitos freqüentes para encontrar novos conceitos. Essa adaptação leva em conta a proposição 7. Os sucessores de conceitos não-freqüentes também são não-freqüentes, portanto, os conceitos não-freqüentes são desconsiderados.

O problema da descoberta de conjuntos de itens freqüentes é solucionado com o algoritmo FNN apresentado em Algoritmo 20. Utilizou-se o mesmo nome para designar a metodologia para encontrar regras de associação proposta por Carpineto e Romano e para o algoritmo que constrói o reticulado conceitual dos conceitos freqüentes. No entanto, pelo contexto em que o termo FNN é usado, ficará claro se se refere à metodologia ou ao algoritmo para encontrar o reticulado.

O algoritmo inicia adicionando o conceito (G,G') ao reticulado (linha 1). Na linha 2, nível atual (conceitos que serão usados para descobrir novos conceitos) também é iniciado com o conceito (G,G'). Em seguida, o algoritmo entra em um ciclo (linhas 3 à 12) para que todos os conceitos freqüentes sejam encontrados. Para cada conceito no nível atual, o algoritmo encontra seus sucessores freqüentes com a função *encontrarProximosConceitos* (linha 6), atualiza o conjunto dos próximos conceitos a serem avaliados (linha 7) e adiciona os novos conceitos ao reticulado atualizando a relação entre os conceitos (linhas 8 e 9). Em seguida, atualiza o nível atual com os novos conceitos encontrados (linha 11).

Algoritmo Frequent Next Neighbours
Entrada: Um contexto formal (G,M,I) e o mínimo suporte *min_sup*
Saída: Um supremo-semi-reticulado L dos conceitos freqüentes
início
1. $L := \{(G,G')\}$
2. $nivelAtual := \{(G,G')\}$
3. **enquanto** $nivelAtual \neq \emptyset$ **faça**
4. $proximoNivel := \emptyset$
5. **para cada** $(X,Y) \in nivelAtual$ **faça**
6. $proximoConceitosFrequentes := encontrarProximosConceitos((X,Y))$
7. $proximoNivel := proximoNivel \cup (proximoConceitosFrequentes - L)$
8. $L := L \cup proximosConceitosFrequentes$
9. $\forall (X_1,Y_1) \in proximoConceitosFrequentes[(X,Y).sucessores \cup \{(X_1,Y_1)\}]$
10. **fim para**
11. $nivelAtual := proximoNivel$
12. **fim enquanto**

fim

Algoritmo 20: Encontra conceitos freqüentes.

A função para encontrar os próximos conceitos é apresentada no Algoritmo 21. O algoritmo recebe como parâmetro um conceito (X,Y) e combina a intensão do conceito com cada atributo

almejando encontrar novas intensões. No entanto, leva-se em conta o suporte. São considerados candidatos a sucessores do conceito (X,Y) somente os conceitos cujo número de objetos na extensão é pelo menos o mínimo necessário (linhas 4 e 5). Depois, o novo conceito é gerado (linha 6). Se o conceito não foi inserido no conjunto dos candidatos, então ele é inserido e atribui-se o contador 1 a ele indicando que foi gerado pela primeira vez. Caso contrário o contador é incrementado. Esse contador é utilizado para certificar que um conceito é realmente o sucessor de (X,Y) na relação $\prec$. Um conceito (X_1,Y_1) é considerado sucessor de (X,Y), se, para todo conjunto $Y \cup \{m\}$ tal que $m \in Y_1 - Y$, o fecho $Y \cup \{m\}$ é Y_1, ou seja, $\forall m \in Y_1 - Y\ (Y \cup \{m\})'' = Y_1$.

Algoritmo encontrarProximosConceitos
Entrada: Um conceito (X,Y)
Saída: O conjunto *sucessores* dos sucessores freqüentes de (X,Y) com relação a $\prec$
início

```
 1.  candidatos := ∅
 2.  sucessores := ∅
 3.  para cada m ∈ M − Y faça
 4.      sup := |{m'} ∩ X|
 5.      se sup ≥ min_sup então
 6.          (X1,Y1) := ((Y ∪ {m})', (Y ∪ {m})'')
 7.          se (X1,Y1) ∉ candidatos então
 8.              candidatos := candidatos ∪ {(X1,Y1)}
 9.              (X1,Y1).contador := 1
10.          senão
11.              (X1,Y1).contador := (X1,Y1).contador + 1
12.          fim se
13.          se |Y1| − |Y| = (X1,Y1).contador então
14.              sucessores := sucessores ∪ {(X1,Y1)}
15.          fim se
16.      fim se
17.  fim para
18.  retorne sucessores
```

fim

Algoritmo 21: Encontra próximos conceitos freqüentes.

Encontrado o reticulado dos conceitos freqüentes, pode-se encontrar as regras de associação. O problema da descoberta de regras de associação é dividido pelo método FNN, como dividido por Zaki [61, 62] e Luxenburger [36], em dois subproblemas: o problema de encontrar regras de associação com confiança de 100% e o problema de encontrar regras com confiança abaixo de 100%.

O problema de encontrar regras de associação com confiança igual a 100% é tratado da mesma forma que o problema da descoberta de implicações. Logo, Carpineto e Romano solucionaram o problema aplicando o algoritmo *Find Implications* apresentado na seção 3.5.2.

Para solucionar o problema da descoberta de regras de associação com confiança abaixo de 100%, o algoritmo investiga cada conceito atributo $\mu(m)$ e gera regras cujos antecedentes são os subconjuntos contendo m na intensão de $\mu(m)$ e cujos conseqüentes são todos os subconjuntos dos sucessores de $\mu(m)$ que contêm atributos não pertencentes à intensão de $\mu(m)$. O algoritmo é apresentado em Algoritmo 22.

Algoritmo gerarRegrasFNN
Entrada: Um reticulado $\mathfrak{B}(G,M,I)$ de conceitos freqüentes
Saída: O conjunto RA de regras de associação com confiança abaixo de 100%
início

```
1.  para cada (X,Y) ∈ 𝔅(G,M,I) faça
2.      Y_s := ⋃{A|(O,A) ∈ 𝔅(G,M,I) ∧ (O,A) ≻ (X,Y)}
    /* (X,Y) é um conceito atributo, se ele possui atributos não contidos nas intensões de conceitos maiores */
3.      para cada m ∈ Y − Y_s faça
4.          A := {lhs ∈ ℘(Y)|m ∈ lhs}
5.          sup := |X|/|G|
6.          para cada (X_1,Y_1) ∈ (X,Y).sucessores faça
7.              conf := |X_1|/|X|
8.              se conf ≥ min_conf então
9.                  I := Y_1 − Y
10.                 se I ≠ ∅ então
11.                     C := {rhs ∈ ℘(Y_1)|rhs ∩ I ≠ ∅}
12.                     RA := RA ∪ {P → Q_(sup,conf)|(P,Q) ∈ A × C}
13.                 fim se
14.             fim se
15.         fim para
16.     fim para
17. fim para
```

fim

Algoritmo 22: Encontra regras de associação com confiança abaixo de 100%.

O algoritmo verifica cada conceito do reticulado de conceitos freqüentes para constatar se ele é um conceito atributo (linhas 1 à 17). Um conceito (X,Y) é um conceito atributo de todos os atributos não pertencentes à intensão dos conceitos maiores que ele. O algoritmo verifica para quais atributos o conceito é um conceito atributo (linhas 2 e 3). Geram-se regras para todo atributo m tal que (X,Y) é um conceito atributo (linhas 3 à 16). As regras são geradas observando-se os sucessores de (X,Y). A confiança de cada regra é dada pela razão entre o número de objetos na extensão de um sucessor de (X,Y) e o tamanho de X. O suporte das regras é igual ao suporte de Y. Os antecedentes são os subconjuntos de Y que contêm m. Os conseqüentes são todos os subconjuntos das intensões de sucessores de (X,Y) que contenham algum atributo não pertencente a Y. O algoritmo primeiro gera o conjunto de antecedentes (linha 4) e calcula o suporte das regras. Depois, avalia os sucessores de (X,Y) e produz regras apenas com aqueles que a confiança está acima do mínimo (linha 6 à 15). O conjunto de conseqüentes

é gerado na linha 11. As novas regras são produzidas na linha 12. Elas representam todas as combinações entre os antecedentes encontrados na linha 4 e os conseqüentes encontrados na linha 11; o suporte é inerente ao conceito (X,Y) e a confiança inerente à relação (X,Y) e cada um de seus sucessores. Os sucessores de (X,Y) são todos os conceitos (X_1,Y_1) cobertos por (X,Y). As regras entre os conceitos (X,Y) e (X_2,Y_2) tal que $(X_2,Y_2) \leq (X,Y)$ mas $(X,Y) \not\prec (X_2,Y_2)$ não são geradas diretamente. Elas podem ser geradas por transitividade. As regras de associação com confiança abaixo de 100% não respeitam o axioma da transitividade para todas as ocasiões, mas segundo Zaki [62, teorema 6], sendo $P,Q,R \subseteq M$ conjuntos de atributos tais que $P \subseteq Q \subseteq R$, se $P \rightarrow Q_{(s1,c1)}$ e $Q \rightarrow R_{(s2,c2)}$, então $P \rightarrow R_{(s2,c1\times c2)}$.

4.1.4 *Galicia*

O nome Galicia é uma abreviação para *GAlois Lattice-based Incremental Closed Itemset Approach* termo em inglês que designa a idéia de método incremental baseado em reticulados de Galois (reticulados conceituais) para a descoberta de conjunto de itens fechados. O algoritmo Galicia constrói o reticulado de conceitos freqüentes de maneira incremental. Ele foi porposto por Valtchev e seus co-autores [55]. O algoritmo é uma adaptação do algoritmo de Godin [33].

O algoritmo atualiza o reticulado com a adição de um novo objeto sem a necessidade de reconstruir todo o reticulado. Este método torna-se interessante pelo fato das constantes atualizações nos bancos de dados. Assim, ao incluir um novo registro em um banco de dados, não existe a necessidade de se reconstruir todo o reticulado.

A intensão de um novo objeto é por si só um conjunto fechado. Como o reticulado conceitual é fechado sob interseções, ou seja, como o ínfimo e o supremo de qualquer conjunto de conceitos sempre existem no reticulado conceitual (de acordo com o teorema fundamental da AFC 1), a inclusão do novo objeto envolve a computação das interseções de sua intensão com as intensões dos conceitos do reticulado.

Ao atualizar o reticulado com a inclusão de um objeto g, divide-se os conceitos em três grupos distintos: conceitos geradores ($CG(g)$) que dão origem a novos conceitos; conceitos modificados ($CM(g)$) cujas intensões são subconjuntos da intensão de g; e conceitos imutáveis ($IM(g)$) que permanecem inalterados com a inclusão de g. Os conceitos geradores são aqueles que as interseções de suas intensões com a intensão do novo objeto não pertencem ao reticulado. Eles são utilizados para gerar os novos conceitos auxiliando na computação da intensão e extensão dos novos conceitos. Os conceitos modificados são aqueles que apenas suas extensões são modificadas incluindo-se o novo objeto. A intensão desses conceitos é sempre subconjunto dos atributos do novo objeto. Já os conceitos imutáveis são aqueles que não sofrem modificações com a inclusão do novo objeto. O algoritmo para atualizar o reticulado consiste em encontrar

esses três grupos e executar as modificações necessárias.

Para construir o reticulado de um contexto formal (G,M,I) incrementalmente, o algoritmo inicia com o reticulado $\langle\{(M',M)\},\emptyset\rangle$ e considera a inclusão de cada objeto $g \in G$. O Algoritmo 23 apresenta o pseudo-código do algoritmo para a construção incremental do reticulado. Como o objetivo do algoritmo é encontrar o conjunto das intensões freqüentes, ao invés de armazenar a extensão e a intensão de cada conceito, armazena-se a intensão do conceito e o número de objetos na extensão como forma de melhorar o desempenho do algoritmo.

Algoritmo Galicia
Entrada: Um contexto formal (G,M,I), e o mínimo suporte *min_sup*
Saída: O conjunto das intensões dos conceitos freqüentes
início
1. $L := \{(|M'|/|G|, M)\}$
2. **para cada** $g \in G$ **faça**
3. $\quad atualizarReticulado(L,g)$
4. **fim para**
5. **retorne** $\{(sup,Y) \in L | sup \geq min\_sup\}$

fim

Algoritmo 23: Encontra o conjunto das intensões freqüentes.

A função *atualizarReticulado*, apresentada em Algoritmo 24, é responsável por incluir um novo objeto ao reticulado. O algoritmo encontra as três classes de conceitos mencionadas anteriormente, verifica se as interseções dos atributos do novo objeto estão presentes no reticulado e cria novos conceitos. Não é necessário atualizar as listas de sucessores dos conceitos pois o interesse é nas intensões freqüentes. Porém, caso algoritmo fosse utilizado com o objetivo de construir um reticulado conceitual, então deve-se considerar a atualização das listas de sucessores entre os conceitos.

O algoritmo verifica cada conceito (sup,Y)[4] presente no reticulado comparando sua intensão com a intensão do novo objeto (linhas 2 à 15). Se a intensão do conceito Y é subconjunto da intensão do novo objeto (linha 3), então o conceito (sup,Y) é um conceito modificado e basta adicionar o objeto à extensão do conceito, logo, o suporte do conceito é incrementado (linha 4). Se o conceito não é um conceito modificado, então ele é um conceito imutável ou um conceito gerador. Para descobrir a qual dos dois conjuntos, $CG(g)$ ou $IM(g)$, pertence o conceito (sup,Y), verifica-se a interseção $Y \cap g'$ (linha 6). Se existe um conceito em L cuja intensão é $Y \cap g'$, então (sup,Y) é um conceito imutável. Caso contrário o conceito é um potencial gerador. O conceito (sup,Y) é um conceito gerador se o conceito cuja intensão é $Y \cap g'$ não fora

[4]Como foi mencionado anteriormente, um conceito formal (X,Y) é substituído pelo par $(|X'|,Y)$ por motivos de desempenho.

gerado em outra iteração (linha 8)[5]. Se o conceito (sup, Y) é um conceito gerador, então um novo conceito é gerado. A intensão do novo conceito é a interseção entre a intensão do objeto e Y e o suporte é o suporte de (sup, Y) mais um (linha 9). Se o conceito (sup, Y) não é um conceito gerador, ou seja, o conceito $(x, Y_1 = Y \cap g')$ já foi gerado, então o suporte de (x, Y_1) é atualizado (linha 11). Após todos os conceitos terem sido verificados, os novos conceitos são incluídos no reticulado (linha 16).

Algoritmo atualizarReticulado
Entrada: Um reticulado conceitual L e um objeto $g \in G$
Saída: O reticulado L atualizado com a inclusão do novo objeto
início

```
 1.  novosConceitos := ∅
 2.  para cada (sup,Y) ∈ L faça
 3.    se Y ⊆ g' então
                /* (sup,Y) é um conceito modificado */
 4.      sup := sup + 1
 5.    senão
 6.      Y1 := Y ∩ g'
                /* se (x,Y1) ∈ L, então (x,Y1) é um conceito imutável */
 7.      se (x,Y1) ∉ L então
                /* senão é um potencial conceito gerador */
 8.        se (x,Y1) ∉ novosConceitos então
 9.          novosConceitos := novosConceitos ∪ {(sup + 1, Y1)}
10.        senão
11.          x := max(sup + 1, x)
12.        fim se
13.      fim se
14.    fim se
15.  fim para
16.  L := L ∪ novosConceitos
```

fim

Algoritmo 24: Atualiza o reticulado com um novo objeto.

O algoritmo Galicia é adequado para sitações em que se deseja atualizar um reticulado. Seu desempenho na construção do reticulado, segundo Valtchev [55], é inferior em relação a outros algoritmos que constróem o reticulado inteiro. O algoritmo também apresenta outra deficiência. Durante a computação dos conceitos freqüentes é necessário que todos os conceitos sejam mantidos, mesmo aqueles não-freqüentes. Isto deve-se ao fato de que após sucessivas inclusões de objetos, conceitos freqüentes podem se tornar conceitos não-freqüentes, assim como conceitos não-freqüentes podem se tornar conceitos freqüentes.

[5] A váriavel x no conceito (x, Y_1) foi usada para indicar que o interesse é na intensão do conceito, assim, o suporte é desprezado.

Finalmente, encontrado o reticulado conceitual com o algoritmo Galicia, pode-se selecionar o conjunto das intensões freqüentes, derivar o conjunto de itens freqüentes e extrair o conjunto de regras válidas utilizando os algoritmos 13 e 14 empregados no algoritmo AClose (seção 4.1.1).

4.2 Regras de Classificação

Outro processo importante para a mineração de dados é o processo de classificação. O processo de classificação consiste em descobrir características de classes de objetos e, depois, utilizá-las para classificar objetos com classes desconhecidas.

O processo de classificação é dividido em duas etapas: utilizar um conjunto de objetos com classes conhecidas para determinar um modelo descrevendo as características de cada classe e utilizar o modelo obtido na etapa anterior para classificar objetos que não se conhece a classe. A primeira etapa é conhecida como **fase de aprendizagem**, enquanto a segunda como **fase de classificação**.

Durante a fase de aprendizagem, o conjunto de objetos com a classe conhecida é dividido em dois subconjuntos. Um subconjunto de objetos é chamado de **conjunto de treinamento** e o outro é chamado de **conjunto de validação**. O conjunto de treinamento é utilizado diretamente na construção do modelo. É desse conjunto que se extraem as características de cada classe, ou seja, o conjunto de atributos que, geralmente, um conjunto de objetos de uma classe possui. Já o conjunto de validação é utilizado na avaliação do modelo. O modelo é aplicado aos objetos do conjunto de validação e verifica-se a precisão do modelo, quantos objetos do conjunto de validação foram classificados corretamente.

A fase de classificação consiste em utilizar um modelo construído e já validado durante a etapa anterior na predição de classes a objetos com a classe desconhecida. Nesta etapa, o modelo obtido do conjunto de treinamento é efetivamente usado.

Os modelos podem ser obtidos a partir do conjunto de treinamento, por exemplo, através de redes neurais artificiais, árvores de decisão e regras de classificação. Regras de classificação são uma das técnicas mais populares para expressar um modelo [60]. As regras de classificação tal como as regras de associação relatam correlações entre conjuntos de atributos. Contudo, as regras de classificação mostram correlações entre atributos e classes. No antecedente de cada regra de classificação podem aparecer quaisquer atributos, enquanto no conseqüente restringe-se que seja um atributo-classe. Seja M um conjunto de atributos e $C \subset M$ um conjunto de atributos-classe. Formalmente, as regras de classificação são pares (X,Y) tais que $X \subseteq M - C$ e $Y \in C$. Uma regra (X,Y) também recebe a notação $X \rightarrow Y$.

Os conjuntos de regras de classificação são divididos em dois grupos: listas de decisão e regras não-ordenadas. Nas listas de decisão, existe uma ordem nas regras. Ao tentar classificar um novo objeto, inicia-se o processo verificando a primeira regra, caso ela não sirva para classificar o objeto, investiga-se a próxima regra seqüencialmente. Já as regras de um conjunto de regras não-ordenadas, como o próprio nome revela, não existe ordem em relação às regras. Logo, considera-se, em geral, todas as regras que servem para classificar um objeto. Este processo é ineficaz quando existem regras conflitantes no conjunto de regras, ou seja, quando diferentes regras atribuem classes diferentes a um mesmo objeto. Esse problema é geralmente resolvido utilizando-se uma estratégia de voto majoritário, atribui-se a classe a um objeto de acordo com a maioria das regras.

O uso de reticulados conceituais para classificação ainda foi pouco explorado apesar de ser uma das primeiras aplicações da AFC. Os reticulados explicitam as dependências entre atributos e as relações entre atributos e objetos. Com isso, eles podem ser utilizados como guia no espaço de busca de regras tal como na busca por regras de associação. Além disso, a estrutura dos reticulados também pode ser usada na classificação [12, 14]. Nesse caso, não são geradas regras, mas, usa-se o reticulado para inferir a classe de um novo objeto.

Como as regras de classificação são relações entre atributos, os algoritmos estudados nas seções 3.5 e 4.1 para extração de implicações e regras de associação podem ser usados na identificação de regras de classificação. Além daqueles, os algoritmos GRAND [43, 44] e *Rulearner* [47] foram propostos especificamente para a geração de regras de classificação. O GRAND permite apenas a extração de regras não-ordenadas. Já o *Rulearner* pode também ser utilizado para a extração de listas de decisão. Esses serão os dois algoritmos avaliados neste trabalho para extração de regras de classificação. Existem outros métodos para classificação através de reticulados como os métodos discutidos por Carpineto e Romano [14], por Fu e seus co-autores [25] e por Nguifo [41]. No entanto, eles utilizam a estrutura do reticulado para classificar e, com isso, não serão avaliados neste trabalho.

4.2.1 GRAND

O algoritmo *Graph-based induction* (GRAND) foi proposto por Oosthuizen [43, 44] para a indução de regras de classificação. O algoritmo utiliza um pseudo-reticulado na geração das regras.

A estrutura do pseudo-reticulado assemelha-se ao reticulado conceitual e consideram-se três tipos de elementos no reticulado: nós-atributos, nós-objetos e nós-intermediários. Para um contexto formal (G, M, I), nós-atributos representam cada um dos atributos de M, os nós-objetos representam os objetos do contexto formal e os nós-intermediários servem para manter

a estrutura do reticulado, ou seja, para garantir que os supremos de quaisquer nós pertençam ao reticulado. O pseudo-reticulado é construído incrementalmente. Inicialmente, cada atributo é representado por um nó-atributo no pseudo-reticulado. Em seguida, inclui-se um novo objeto representado por um nó-objeto. Se já existe no reticulado um nó-objeto com os mesmos atributos que o novo objeto, então a atualização do pseudo-reticulado é feita incluindo-se o novo objeto à extensão do nó-objeto correspondente. Caso contrário, cria-se um nó-objeto para o novo objeto. Depois, associa-se o novo nó-objeto a cada um dos nós-atributos representando a intensão do objeto. Em seguida, verifica-se a estrutura do reticulado; se o supremo (interseção) do novo nó e cada um dos nós presentes no pseudo-reticulado está presente no pseudo-reticulado. Então, atualizam-se as listas de sucessores dos conceitos envolvidos na atualização do pseudo-reticulado e remove-se as associações redundantes.

O algoritmo para a construção do pseudo-reticulado é apresentado em pseudo-código na Figura 25. Inicialmente, criam-se os nós-atributos (linha 1). Depois, o algoritmo entra em um ciclo para incluir todos os objetos no reticulado (linhas 2 a 29). Se existe um nó-objeto pertencente ao pseudo-reticulado com a mesma intensão do objeto a ser incluído, então a atualização do reticulado consiste em incluir o novo objeto à extensão do nó-objeto correspondente (linha 4 e 5). Caso contrário, cria-se um nó-objeto para o novo objeto (linha 7). O novo nó-objeto é associado a todos os nós-atributos que estão na intensão do objeto (linhas 8 a 10). Depois, verifica-se se a estrutura do pseudo-reticulado foi preservada após a inclusão do nó-objeto (linhas 11 a 27), ou seja, se os supremos do nó-objeto e os demais nós do pseudo-reticulado também pertencem ao pseudo-reticulado. Para verificar a estrutura do reticulado, todos os nós já pertencentes ao reticulado, exceto os nós atributos, são avaliados (linha 3). Os nós são avaliados em ordem de tamanho da interseção entre a intensão do nó e a do novo objeto. A cada iteração, escolhe-se o nó (X,Y) que possui o maior número de atributos em comum com o novo objeto (linha 12). Então, cria-se um novo nó cuja intensão é a interseção da intensão do objeto e a do nó que está sendo avaliado e cuja extensão é a extensão do nó avaliado acrescida do novo objeto (linha 13). Este novo nó é incluído no pseudo-reticulado caso ele já não pertença ao mesmo (linha 14). Depois, o novo nó é associado aos demais nós do pseudo-reticulado (linhas 15 a 23). Para evitar que associações redundantes sejam mantidas, ou seja, para garantir que apenas a relação de cobertura seja mantida pelas listas de sucessores, utilizaram-se os conjuntos $conjA$ e $conjB$. O $conjA$ contém os antecessores do novo nó cujas listas de sucessores devem ser atualizadas removendo-se os sucessores do novo nó (linha 25). O $conjB$ contém os sucessores do novo nó e serve para atualizar a lista de sucessores desse novo nó. Os sucessores de nós que estão em $conjB$ são removidos da lista de sucessores do novo nó (linha 26). Após avaliar o nó (X,Y), ele é retirado da lista de nós para que os supremos sejam avaliados (linha 24). O processo é repetido até que não existam mais nós a serem avaliados.

Algoritmo construirPseudoReticulado
Entrada: Um contexto formal (G,M,I)
Saída: O pseudo-reticulado L associado ao contexto formal (G,M,I)
início

1. $L := \{(m', \{m\}) | m \in M\}$
2. **para cada** $g \in G$ **faça**
3. $C := \{(X,Y) \in L - ||Y| > 1\}$ /* *C contém os nós-objetos e nós-intermediários* */
4. **se** $(W,g') \in L$ **então**
5. $W := W \cup \{g\}$
6. **senão**
7. $L := L \cup \{(\{g\}, g')\}$
8. **para cada** $m \in g'$ **faça**
9. $(m', \{m\}).sucessores := (m', \{m\}).sucessores \cup \{(\{g\}, g')\}$
10. **fim para**
11. **enquanto** $C \neq \emptyset$ **faça**
12. $(X,Y) := (X_i, Y_i) \in C$ tal que $|Y_i \cap g'| = max\{|Z \cap g'| | (W,Z) \in C\}$
13. $(X_2, Y_2) := (\{g\} \cup X, g' \cap Y)$
14. $L := L \cup \{(X_2, Y_2)\}$
15. **para cada** $(X_1, Y_1) \in L$ **faça**
16. **se** $(X_1, Y_1) > (X_2, Y_2)$ **então**
17. $(X_1, Y_1).sucessores := (X_1, Y_1).sucessores \cup \{(X_2, Y_2)\}$
18. $conjA \leftarrow conjA \cup (X_1, Y_1)$
19. **senão se** $(X_1, Y_1) < (X_2, Y_2)$ **então**
20. $(X_2, Y_2).sucessores := (X_2, Y_2).sucessores \cup \{(X_1, Y_1)\}$
21. $conjB := conjB \cup (X_1, Y_1)$
22. **fim se**
23. **fim para**
24. $C := C - \{(X,Y)\}$
25. $\forall (W,Z) \in conjA \quad (W,Z).sucessores := (W,Z).sucessores - (X_2, Y_2).sucessores$
26. $\forall (W,Z) \in conjB \quad (X_2, Y_2).sucessores := (X_2, Y_2).sucessores - (W,Z).sucessores$
27. **fim enquanto**
28. **fim se**
29. **fim para**

fim

Algoritmo 25: Constrói pseudo-reticulados.

Após a construção do pseudo-reticulado, as regras de classificação podem ser extraídas. As regras são geradas para cada um dos atributos-classe. Ao gerar as regras para um atributo-classe, o algoritmo faz uma busca pelo pseudo-reticulado tentando identificar o nó mais geral (com menos atributos) a partir do qual pode-se deduzir a classe. A busca inicia-se pelo nó-atributo corresponde à classe que se está extraindo as regras. Visitam-se os sucessores desse nó até encontrar um nó (X,Y) que atenda às seguintes propriedades:

1 o atributo-classe c pertence à intensão Y do nó;

2 $c \in (Y - \{c\})''$;

3 Y é mínimo em relação às regras já geradas.

Quando as propriedades forem atendidas, gera-se a regra $Y - \{c\} \rightarrow c$. Os sucessores de (X,Y) não são avaliados pois eles não serão mínimos.

O algoritmo para encontrar as regras é apresentado na Figura 26. É executada uma busca em largura nos sucessores do nó-atributo da classe cujas regras serão extraídas (linha 1). A função *gerarRegrasNos* executa efetivamente a geração de regras a partir dos nós do pseudo-reticulado. Ela é apresentada na Figura 27. O algoritmo verifica, para cada nó (X,Y) que ainda deve ser avaliado (*nosVisitar*), seus sucessores na tentativa de extrair as regras (linhas 1 a 9). Se um nó (X_1,Y_1) sucessor de (X,Y) atende às propriedades descritas acima, então uma nova regra é criada (linhas 3 e 4). Caso contrário, os sucessores do nó (X_1,Y_1) deverão ser avaliados na próxima iteração e, com isso, o nó é inserido no conjunto dos próximos nós a serem avaliados (linha 6). Após avaliar todos os nós de um nível, a função *gerarRegrasNos* é chamada recursivamente para os nós do próximo nível (linhas 10 a 12). O algoritmo termina quando não existem mais nós a serem avaliados, ou seja, quando *proximosNos* está vazio.

Algoritmo GerarRegras
Entrada: Um pseudo-reticulado L e um conjunto de atributos-classe C
Saída: Um conjunto R de regras de classificação válidas
início
1. **para cada** $c \in C$ **faça**
2. $(X,Y) := (W,\{c\}) \in L$ /* (X,Y) *é o nó-atributo da classe* c */
3. $R := R \cup gerarRegrasNos((X,Y).sucessores, L, c)$
4. **fim para**
fim

Algoritmo 26: Gera regras de classificação.

Algoritmo gerarRegrasNos
Entrada: Uma lista de nós a visitar *nosVisitar*, um pseudo-reticulado L e um atributo-classe c
Saída: Um conjunto R de regras de classificação válidas
início
1. **para cada** $(X,Y) \in nosVisitar$ **faça**
2. **para cada** $(X_1,Y_1) \in (X,Y).sucessores$ **faça**
3. **se** $c \in (Y_1 - \{c\})''$ **então**
4. $R := R \cup \{Y_1 - \{c\} \rightarrow c\}$
5. **senão**
6. $proximosNos := proximosNos \cup \{(X_1,Y_1)\}$
7. **fim se**
8. **fim para**
9. **fim para**
10. **se** $proximosNos \neq \emptyset$ **então**
11. $R := R \cup gerarRegrasNos(proximosNos, L, c)$
12. **fim se**
13. **retorne** R

fim

Algoritmo 27: Gera regras de classificação para nós sucessores.

4.2.2 *Rulearner*

O algoritmo *Rulearner* foi proposto por Sahami [47]. Assim como o GRAND, o algoritmo *Rulearner* utiliza pseudo-reticulados na extração das regras. No entanto, o *Rulearner* apresenta algumas diferenças em relação ao GRAND.

A primeira diferença está na construção do pseudo-reticulado. Enquanto o GRAND considera os atributos-classe durante a construção do pseudo-reticulado, o *Rulearner* desconsidera esses atributos. Ao invés dos atributos-classe, o *Rulearner* considera uma função que, dado um conjunto de objetos, retorna a classe desses objetos. A função, ao ser aplicada sobre um único objeto, retorna o atributo-classe desse objeto. Quando aplicada a um conjunto com mais de um atributo, a função retorna a classe da maioria dos objetos. A maioria é determinada por um parâmetro de exceção informado pelo usuário. Esse parâmetro de exceção, como a confiança nas regras de associação, informa qual o percentual de objetos é necessário para que uma classe seja considerada para um conjunto de objetos. Se não existir essa maioria para uma determinada classe, então a função retorna o valor classe mista, indicando que não se pode determinar qual a classe daquele conjunto.

Outra diferença está no tipo de regras que o *Rulearner* pode gerar. O *Rulearner* pode ser usado tanto para gerar regras não-ordenadas (como o GRAND) quanto para gerar listas de decisão.

O algoritmo é apresentado em pseudo-código na Figura 28. O algoritmo recebe como parâmetros o contexto formal de que se deseja extrair as regras, o conjunto de atributos-classe

C e uma função de classificação $c : \wp(G) \rightarrow C$ com um parâmetro de exceção associado. Como o algoritmo constrói o pseudo-reticulado desconsiderando os atributos-classe, inicialmente, um novo contexto formal sem os atributos-classe é obtido (linha 1). Em seguida, constrói-se o pseudo-reticulado (linha 2) e geram-se as regras (linha 3). O algoritmo para construção do pseudo-reticulado foi apresentado na seção 4.2.1 na Figura 25. A função para extração das regras é apresentada na Figura 29.

Algoritmo Rulearner
Entrada: Um contexto formal (G,M,I), um conjunto de atributos-classe $C \subset M$
e uma função de classificação $c : \wp(G) \rightarrow C$ associada a um parâmetro de exceção $e \in [0,1]$
Saída: Um conjunto R de regras de classificação válidas
início
1. $\mathbb{K} := (G, M-C, J)$ tal que $J = G \times (M-C) \cap I$
2. $L := construirReticulado(\mathbb{K})$
3. $R := extrairRegras(L)$
4. **retorne** R
fim

Algoritmo 28: *Rulearner*.

A função *extrairRegras* empregada no algoritmo *Rulearner* extrai regras de classificação que atendem a todos os objetos do contexto formal utilizado como conjunto de treinamento. O algoritmo marca inicialmente todos os nós do pseudo-reticulado como ativos, indicando que seus objetos ainda não foram classificados (linha 1). Então, o algoritmo entra em um ciclo até que todos os objetos tenham sido classificados, ou seja, enquanto existirem nós ativos (linhas 2 a 16). Os nós são avaliados em ordem decrescente em relação a ordem de inclusão entre as extensões (linha 3). Com isso, as regras geradas classificam o maior número de objetos por vez. São considerados, para a extração de regras, apenas os nós cuja classe é conhecida. Escolhido o nó com a maior extensão, uma nova regra é gerada em que o antecedente é a intensão do nó e o conseqüente é a classe dos objetos da extensão (linha 4). O próximo passo é a atualização da atividade dos nós. Um nó é considerado inativo se não existem mais objetos em sua extensão para serem classificados. Para atualizar a atividade de um nó, o algoritmo decrementa a cobertura (número de objetos na extensão que ainda não foram classificados) de todos os nós que possuam algum objeto em comum com o nó utilizado para a extração da regra. O algoritmo considera cada um dos objetos na extensão do nó utilizado para gerar a regra (linhas 5 a 12). Para cada um desses objetos, o algoritmo encontra o nó-objeto correspondente e decrementa a cobertura de todos os nós maiores ou iguais a ele (linhas 6 a 11). Se a cobertura de um nó for menor ou igual a zero, ele é considerado inativo (linhas 8 e 9). Caso o algoritmo seja usado para a extração de listas de decisão, os nós ativos devem ser re-rotulados. Nesse

caso, a função de classificação não deve levar em conta apenas os objetos de um conjunto, a classe deve ser atribuída com base nos nós inferiores, assim, a classificação deixa de ser com base na maioria das classes dos objetos e passa a ser com base na maioria das classes dos nós.

Algoritmo extrairRegras
Entrada: Um pseudo-reticulado L e uma função de classificação $c : \wp(G) \rightarrow C$
Saída: Um conjunto R de regras de classificação válidas
início
1. $\forall (X,Y) \in L$[marcar (X,Y) como ativo]
2. **enquanto** existirem nós ativos **faça**
   ** maior com respeito a ordem de inclusão sobre as extensões **
3. $(X,Y) := max\{(X_1,Y_1) \in L | c(X_1) \neq$ misto$\}$
4. $R := R \cup \{Y \rightarrow c(X)\}$
5. **para cada** $g \in X$ **faça**
6. **para cada** $(X_1,Y_1) \in \uparrow\gamma(g)$ **faça**
7. $(X_1,Y_1).cobertura := (X_1,Y_1).cobertura - 1$
8. **se** $(X_1,Y_1).cobertura \leq 0$ **então**
9. marcar (X_1,Y_1) como inativo
10. **fim se**
11. **fim para**
12. **fim para**
13. **se** lista-de-decisão **então**
14. re-rotular nós ativos
15. **fim se**
16. **fim enquanto**
17. **retorne** R

fim

Algoritmo 29: Extrai regras de classificação de um pseudo-reticulado.

Geradas as regras, a classificação de novos objetos é feita de duas formas. Se uma lista de decisão for obtida, então o objeto é classificado com base na primeira regra aplicável. Se for obtido um conjunto não-ordenado de regras, então um objeto é classificado de acordo com a maioria das regras. Observam-se todas as regras aplicáveis e atribui-se a classe obtida pela maioria das regras. Mais uma vez, essa maioria é determinada pelo usuário.

4.3 Comparação dos métodos

Esta seção é dedicada à comparação dos algoritmos para extração de regras de associação e de regras de classificação. Como na seção 3.6, os métodos foram comparados sob o ponto de vista prático. Eles foram implementados em Java e foram executados em um computador com processador Pentium 3, 850 MHz e com 440MB de memória principal em ambiente operacional Windows XP.

Tabela 10: Contextos formais para avaliação dos algoritmos

ID	Contexto	# Objetos	# Atributos	$\|I\|$	média de atributos/objeto	Densidade(%)
1	hepatite	80	24	1231	15	64,11
2	contraceptivo	1473	30	12777	8	28,91
3	T5I4D10K	10000	10	52899	5	52,90
4	M15T3I4D1K	1000	15	3747	3	24,98
5	M15T5I4D1K	1000	15	4935	5	32,90
6	M15T6I4D1K	1000	15	5497	6	36,65
7	M15T8I4D1K	1000	15	6788	8	45,25
8	M7T3I2D50K	50000	7	152259	3	43,50
9	Votos	232	18	2171	9	51,99
10	Monk1	124	19	714	6	30,31
11	Monk2	169	19	970	6	30,21
12	Monk3	122	19	704	6	30,37
13	Car	1728	25	12096	7	28,00
14	lentes	24	11	108	4	40,91

Os algoritmos foram comparados utilizando-se bases de dados reais e sintéticas. As bases de dados reais, assim como na seção 3.6, foram retiradas do repositório de bases de dados da Universidade da Califórnia em Irvine. As bases de dados sintéticas foram geradas conforme descrito na seção 3.6, utilizando-se a metodologia sugerida por Agrawal [3].

A Tabela 10 apresenta as bases de dados utilizadas para a comparação dos métodos, porém, já transformadas em contextos formais. A tabela apresenta o número de objetos, o número de atributos, o tamanho da relação de incidência, a média de atributos por objetos e a densidade dos contextos. A densidade dos contextos refere-se à razão entre o número de elementos da relação de incidência e o máximo de elementos que ela poderia ter (em um contexto (G,M,I), o máximo de elementos que a relação de incidência pode ter é dado por $|G| \times |M|$).

Os contextos de 1 a 8 na Tabela 10 foram utilizados na comparação dos métodos para a extração de regras de associação. Enquanto os demais contextos, além do 1 e 2, foram utilizados para a extração de regras de classificação. O leitor pode observar que, para a comparação dos métodos de extração de regras de classificação, foram utilizadas apenas bases de dados reais. Esse fato deve-se à ausência de técnicas comprovadamente eficientes para a geração de bases de dados sintéticas para a avaliação de métodos de classificação. Logo, optou-se em utilizar, neste trabalho, bases de dados amplamente utilizadas por diversos autores para a comparação de métodos de classificação.

O contexto 1, tal como descrito na seção 3.6, refere-se a dados relativos a doenças hepáticas. O contexto, apesar de aparentemente ser um contexto pequeno, foi escolhido devido ao alto índice de correlação entre seus dados. Os objetos, em geral, apresentam muitos atributos e muitos atributos em comum com outros objetos. Logo, ele foi escolhido para verificar o comportamento dos algoritmos com bases de dados reais e densas.

O contexto 2 refere-se à base de dados sobre a escolha de métodos contraceptivos entre mulheres casadas na Indonésia de acordo com suas condições sócio-econômicas. Essa base é um exemplo de base com baixa densidade (aproximadamente 29%). Assim, ela foi escolhida para avaliar o desempenho dos algoritmos com bases de dados reais com baixa densidade.

Os contextos de 3 a 8 referem-se a bases de dados sintéticas. Os contextos 3 e 8 foram gerados para verificar o comportamento dos algoritmos com bases de dados em que o número de objetos é significativo. Já as bases de 4 a 7 foram geradas para verificar o desempenho dos algoritmos com a variação da densidade dos contextos. Em princípio, foram gerados contextos com densidades variando entre 20 e 70% para contextos com 1000 objetos e 15 atributos. No entanto, quando a média de atributos por objeto chegou a 9[6], os algoritmos, em sua maioria, não terminaram o teste devido a falta de memória. Com isso, foram utilizados apenas os contextos entre 4 e 7.

Uma ressalva em relação à comparação dos métodos de extração de regras de associação é que o algoritmo Galicia, em geral, não foi comparado para os mesmos contextos que os demais. Isso porque o método é incremental, assim, o tempo gasto para a extração de regras é muito maior que dos demais quando o número de objetos no contexto é grande. Conforme sugestão de seus próprios autores em [55], o algoritmo é adequado para ocasiões em que a base de dados é alterada com a adição de novas tuplas (objetos). Logo, o algoritmo foi analisado separadamente para avaliar seu comportamento simulando situações em que diferentes quantidades de objetos são adicionadas à base.

Os tempos de execução dos métodos para contextos com diferentes densidades são apresentados na Figura 9. Nesse teste, manteve-se suporte de 2% e confiança de 50%. Os algoritmos *Titanic* e *AClose* mostraram-se extremamente sensíveis ao aumento na densidade dos contextos, apresentando crescimento exponencial do tempo de execução. Já o FNN não se mostrou muito sensível à variação na densidade dos contextos e apresentou crescimento linear em seu tempo de execução. Isso retrata características importantes dos algoritmos. Tanto o *Titanic* quanto o *AClose* foram criados com base no algoritmo *Apriori* [3]. O algoritmo *Apriori* não é eficiente para bases de dados com alto índice de correlação entre os dados. O *Titanic* e o *AClose* seguiram o mesmo comportamento do *Apriori* para bases correlacionadas e não se mostraram eficientes nessas situações. À medida em que a densidade do contexto aumenta, os objetos tornam-se mais correlacionados, com mais atributos em comum. Foi exatamente nessas situações que o desempenho dos algoritmos caiu. Já o FNN utiliza diretamente reticulados conceituais para a extração das regras e, com isso, a influência do aumento da densidade em seu tempo de execução é retardada, pois o pior caso ocorre quando a densidade do contexto chega a 75%.

[6] A densidade do contexto foi aumentada fixando-se o número de objetos e de atributos e aumentando-se a média de atributos por objeto.

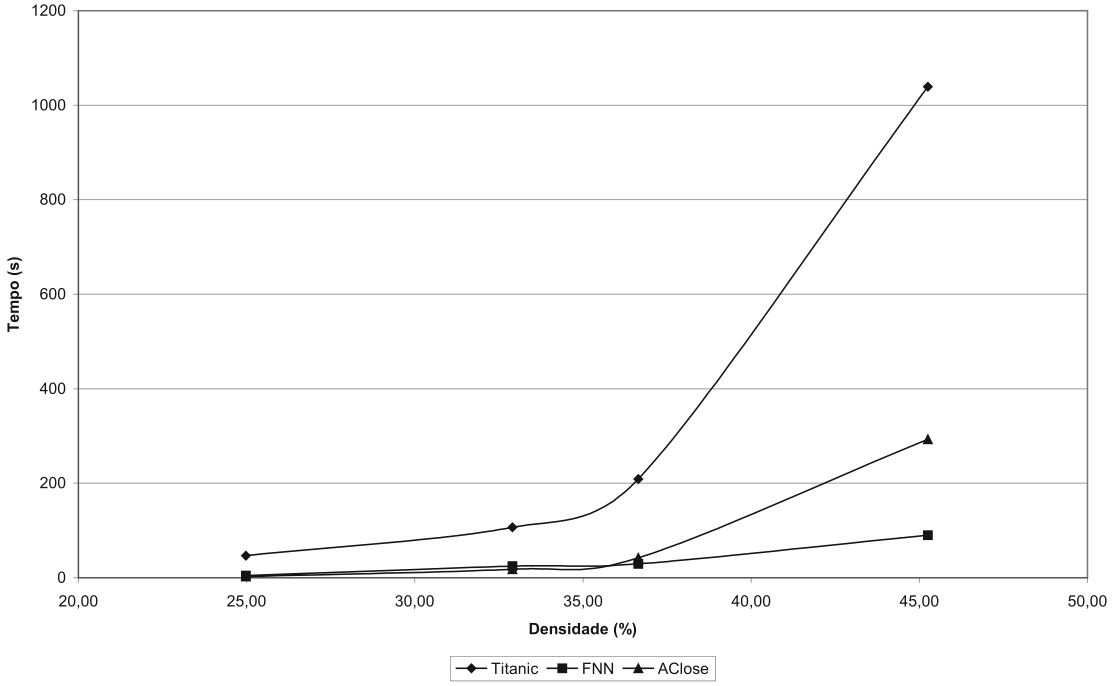

Figura 9: Desempenho dos algoritmos para extração de regras de associação variando-se a densidade e fixando-se o suporte em 2% e confiança em 50%.

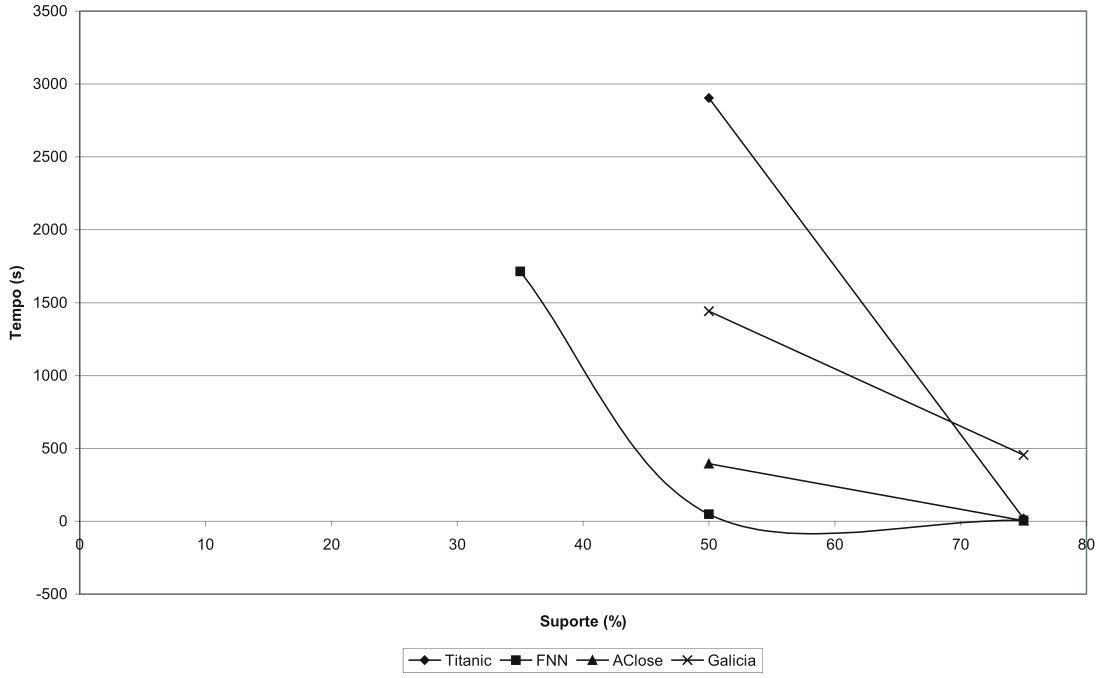

Figura 10: Desempenho dos algoritmos para extração de regras de associação variando-se o suporte e fixando-se a confiança em 50% utilizando o contexto 1.

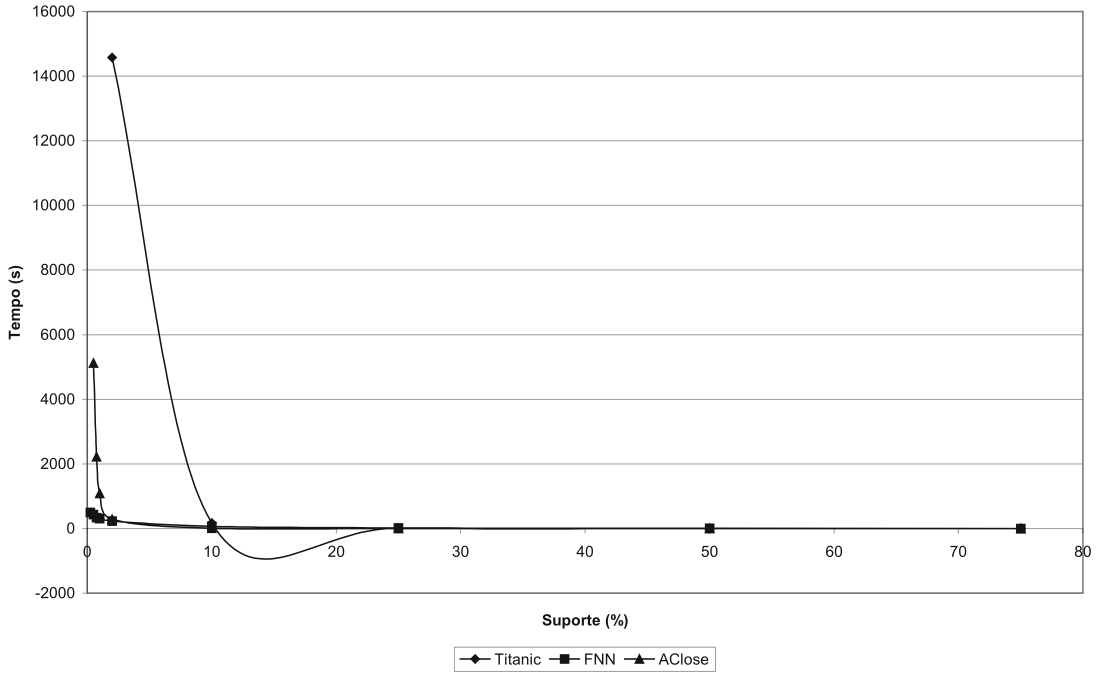

Figura 11: Desempenho dos algoritmos para extração de regras de associação variando-se o suporte e fixando-se a confiança em 50% utilizando o contexto 2.

A Figura 10 apresenta os tempos de execução dos quatro algoritmos para o contexto 1. Nesse teste, fixou-se a confiança em 50% e variou-se o suporte para verificar o desempenho dos algoritmos. O teste foi feito com variação no suporte já que o número de itens freqüentes em um contexto é determinado pelo suporte. Objetivou-se verificar o comportamento dos algoritmos com aumento do número de itens freqüentes. O suporte foi variado entre 75% e 35%. Contudo, apenas o FNN conseguiu executar o teste com suporte de 35%. Os demais executaram apenas com suporte de 75% e 50%. Ao diminuir o suporte, o *Titanic*, o *AClose* e o Galicia tiveram o teste abortado devido a falta de memória. Entre os algoritmos, o que se mostrou mais sensível à diminuição do suporte foi o *Titanic*. A redução de 25% no suporte proporcionou um aumento de quase 150 vezes no tempo de execução. A mesma redução proporcionou aumento de aproximadamente 100 vezes no tempo de execução do *AClose*, cerca de 9 vezes no tempo de execução do FNN e 3 vezes no tempo do Galicia. Isso aponta uma certa superioridade dos métodos que utilizam diretamente reticulados conceituais na extração de regras para contextos densos.

Os tempos de execução dos algoritmos para o contexto 2 são apresentados na Figura 11. Mais uma vez, fixou-se a confiança das regras em 50%, mas, variou-se o suporte entre 75 e 0,25%. Apenas o FNN conseguiu terminar o teste com 0,25%. O *AClose* teve o teste abortado com 0,25% e o *Titanic* com 1%; ambos devido a falta de memória. Os algoritmos que tiveram os testes abortados possuem crescimento exponencial no tempo de execução com a diminuição do suporte. Observe que o FNN manteve o tempo de execução praticamente constante. Isso,

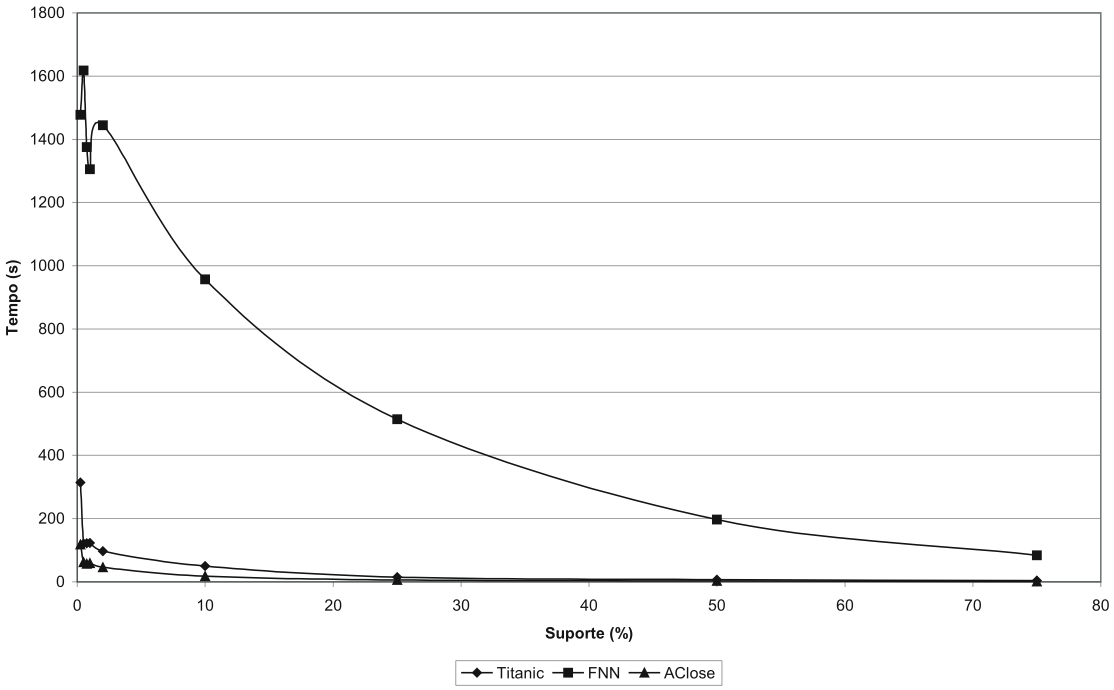

Figura 12: Desempenho dos algoritmos para extração de regras de associação variando-se o suporte e fixando-se a confiança em 50% utilizando o contexto 3.

novamente, devido à utilização do reticulado conceitual na extração de regras. Como mencionado na seção 4.1.3, o algoritmo FNN cria um supremo-semi-reticulado com os conceitos freqüentes. Esse supremo-semi-reticulado de conceitos freqüentes é parte do reticulado conceitual e, quando o suporte tende a zero, o supremo-semi-reticulado aproxima-se do reticulado conceitual. Logo, o tempo de execução do FNN limita-se ao tempo de construção do reticulado conceitual que relaciona-se diretamente com a densidade do contexto. Por isso, a variação no suporte tem menos influência para o FNN do que para os demais.

A Figura 12 apresenta o tempo de execução dos algoritmos para o contexto 3. Como nos dois últimos testes, fixou-se a confiança em 50% e variou-se o suporte entre 75% e 0,25%. Esse teste, ao contrário dos dois últimos que avaliavam o comportamento dos algoritmos para contextos com densidades diferentes, avalia os algoritmos com contextos com grande quantidade de objetos. Com o teste, constatou-se que o FNN apresenta o pior desempenho comparado com o *AClose* e *Titanic*. Apesar da densidade do contexto 3 ser de 53% (caso em que o *AClose* e *Titanic* não obtiveram resultados satisfatórios), o desempenho deles foi satisfatório mesmo com baixos suportes. Por outro lado, o desempenho do FNN degrada-se de forma quadrática com a redução do suporte.

O contexto 8 foi também utilizado para avaliar os algoritmos com contexto com muitos objetos. Os tempos de execução são apresentados na Figura 13. Nesse teste, a confiança foi

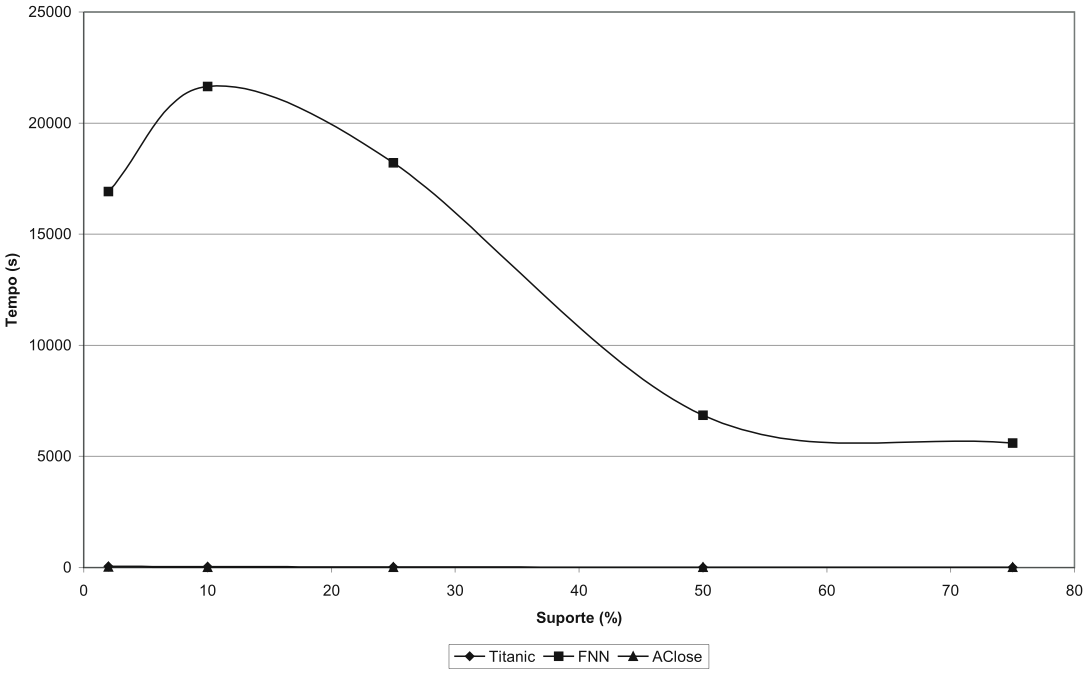

Figura 13: Desempenho dos algoritmos para extração de regras de associação variando-se o suporte e fixando-se a confiança em 50% utilizando o contexto 8.

mantida em 50% e variou-se o suporte entre 75% e 2%. Como acontecido com o contexto 3, o desempenho do FNN foi muito inferior quando comparado com o *AClose* e o *Titanic*. Tanto o *AClose* quanto o *Titanic* mantiveram o tempo de execução praticamente constante com a variação do suporte. O FNN mostrou crescimento gradual com a redução do suporte. A redução de 75% para 10% no suporte proporcionou um aumento de cerca 4 vezes no tempo de execução do FNN. Isso reflete novamente que o algoritmo não sofre grande influência com a redução do suporte, porém, sofre grande influência com o aumento do número de objetos no contexto.

Como foi dito, o algoritmo Galicia foi analisado separadamente por ser um algoritmo incremental. Ele foi analisado sob duas situações distintas. A primeira sob uma suposta utilização direta do algoritmo para a extração de regras de associação (como já dito, essa não é a situação mais adequada para a utilização do algoritmo). A segunda situação tem como objetivo analisar o algoritmo simulando a inclusão de diferentes quantidades de objetos no contexto. Para o segundo teste, foram utilizados os contextos 3, 4 e 5 da Tabela 5, simulando a inclusão de 100, 500 e 1000 objetos no contexto; sempre partindo do contexto vazio, sem objetos.

A Figura 14 apresenta o teste em que o Galicia foi utilizado para extrair regras de contextos com diferentes densidades. O algoritmo apresentou tempo de execução elevado mesmo para baixas densidades. Constatou-se também que o aumento do tempo de execução não acompanha

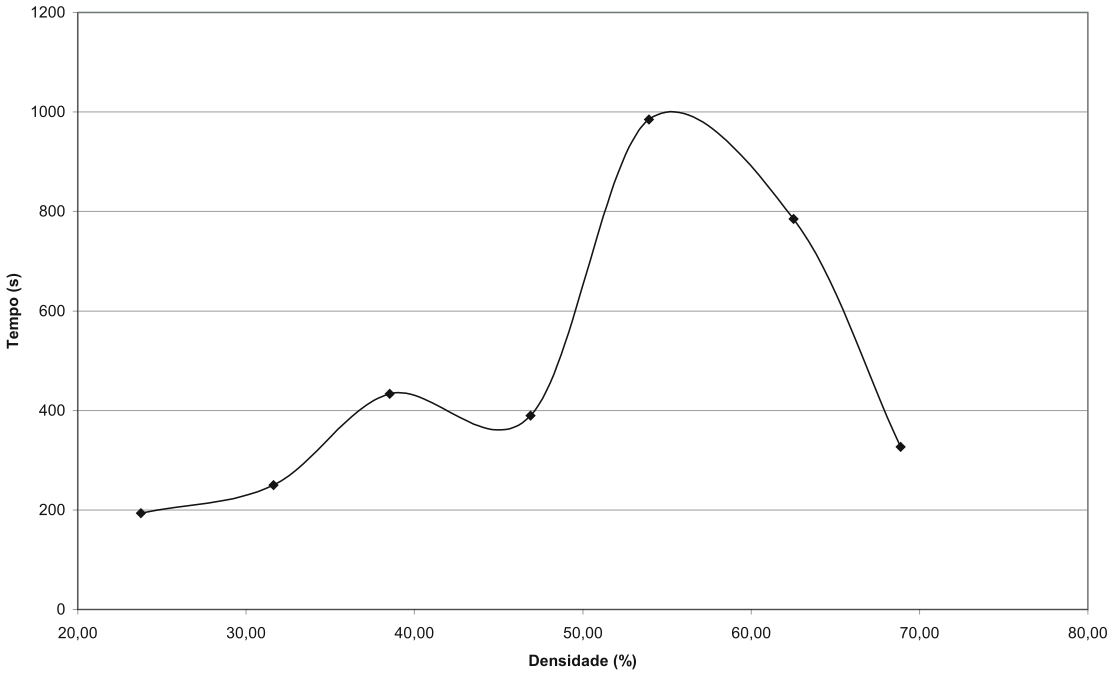

Figura 14: Desempenho do Galicia para extração de regras de associação variando-se a densidade e fixando-se o suporte em 2% e confiança em 50%.

o aumento na densidade do contexto. O tempo de execução aumentou até a densidade de cerca de 57% quando atingiu seu máximo e começou a cair logo em seguida. Esse fato, aparentemente estranho, era esperado devido ao processo de construção do reticulado conceitual do Galicia. Como explicado na seção 4.1.4, o algoritmo atualiza o reticulado conceitual adicionando um objeto por vez, criando conceitos e adaptando a estrutura do reticulado de forma a preservá-la. Mesmo sendo repetitivo, vale a pena lembrar que o aumento na densidade dos contextos torna os objetos mais similares ou, até mesmo, idênticos. Objetos similares no contexto exigem menos operações para a atualização do reticulado conceitual e, conseqüentemente, o tempo de execução é reduzido.

Os tempos de execução do Galicia sob a variação do número de objetos no contexto são apresentados na Figura 15. A figura confirma as expectativas quanto ao tempo de execução do algoritmo em relação a variação do número de objetos e mostra que o aumento do tempo de execução é linear com o aumento do número de objetos. Isso corrobora a sugestão dos autores do Galicia [55], mostrando que o algoritmo é mais adequado quando ocorrem pequenas atualizações no contexto.

Os algoritmos para extração de regras de classificação foram avaliados sob o ponto de vista de tempo de execução e de precisão de classificação. Foram utilizados contextos com densidades diferentes, com números de objetos diferentes e contextos que são amplamente utilizados

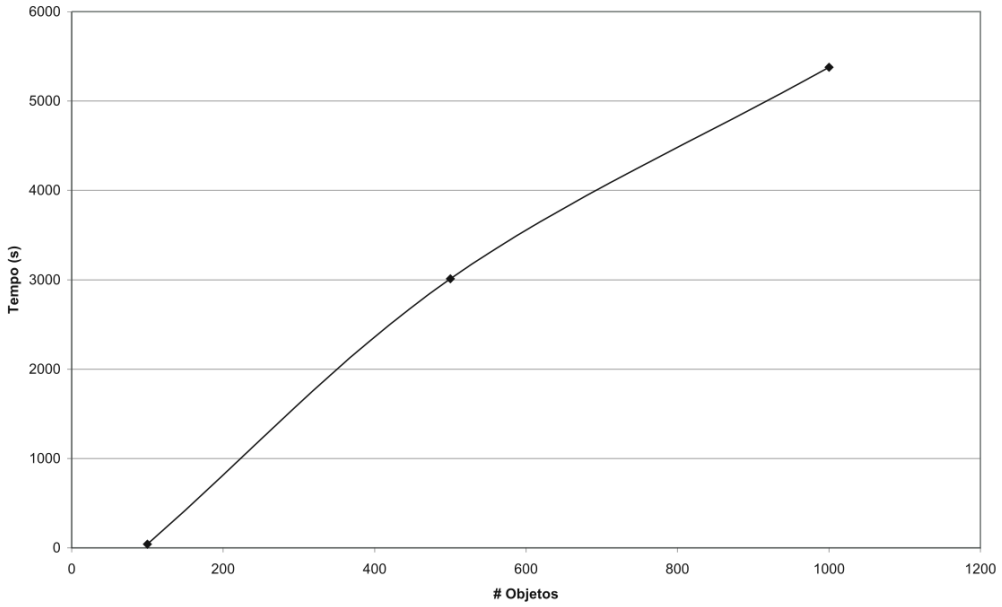

Figura 15: Desempenho do Galicia para extração de regras de associação variando-se o número de objetos e fixando-se o suporte em 2% e confiança em 50%.

Tabela 11: Desempenho dos algoritmos para a classificação

ID Contexto	Precisão (%)		# regras		Tempo de execução (s)	
	GRAND	Rulearner	GRAND	Rulearner	GRAND	Rulearner
1	83,34	83,34	7	7	13622,81	1794,45
2	44,62	47,57	255	183	40805,76	56452,25
9	70,50	76,26	13	10	5739,08	2862,24
10	76,62	87,50	62	29	994,06	1096,76
11	77,32	69,68	114	49	1354,79	441,25
12	84,95	86,34	63	28	622,72	275,75
13	80,33	83,61	183	143	29634,50	24586,40
14	61,54	53,85	3	5	1,88	4,44

na comparação de métodos de classificação, como os contextos 11 a 13 referentes ao problema dos monges [22].

Os contextos, em geral, foram divididos em conjunto de treinamento, utilizado para a obtenção das regras, e conjunto de validação, utilizado na avaliação da precisão dos métodos para classificar novos objetos. Os contextos foram divididos de forma que 40% dos objetos fazem parte do conjunto de treinamento e 60% parte do conjunto de validação. Uma única exceção para essa divisão entre conjunto de treinamento e conjunto de validção refere-se aos contextos 11 a 13, pois eles apresentam-se já divididos nos dois conjuntos no repositório.

A Tabela 11 apresenta os resultados para o GRAND e *Rulearner*. A tabela apresenta as precisões dos métodos, o número de regras extraídas para cada contexto e os tempos de execução. Em média, os algoritmos apresentaram a mesma precisão, 72,4% para o GRAND contra 73,52% para o *Rulearner* e, também, os mesmos tempos de execução, 11596,95 segundos para o GRAND contra 10939,19 segundos para o *Rulearner*.

O desempenho dos algoritmos foi satisfatório na maioria dos casos. No entanto, para o contexto 2, o resultado obtido foi ruim. Além do alto tempo para a obtenção das regras, a precisão na classificação de novos objetos ficou abaixo de 50%. Durante a transformação da base de métodos contraceptivos em contexto formal, os atributos numéricos foram desprezados. Esses atributos podem ter grande influência para a classificação dos objetos. Entretanto, como, em contextos formais, só podem haver atributos binários, eles foram descartados. Esse pode ser um dos motivos que levaram os algoritmos a obter tão baixa precisão.

Outra importante observação acerca da Tabela 11 refere-se ao tempo de execução dos algoritmos. Observe que os algoritmos são extremamente sensíveis ao número de objetos no contexto. Os piores resultados de ambos em relação ao tempo de execução foram obtidos com os contextos 2 e 13 com, respectivamente, 1473 e 1728 objetos. Esses algoritmos são incrementais assim como o Galicia e, portanto, são ineficientes para a computação de regras com bases de dados com muitos objetos. Já quanto à relação tempo de execução e densidade, o GRAND mostrou-se mais sensível ao aumento da densidade. Tomando-se como exemplo o contexto 1, o GRAND foi cerca de 8 vezes mais lento que o *Rulearner*.

5 *Conclusão*

Neste trabalho, foram comparadas técnicas baseadas em Análise Formal de Conceitos para extração de regras de bases de dados. Foram comparadas técnicas para extrair implicações, dependências funcionais, regras de associação e regras de classificação.

Os reticulados conceituais permitem a extração de regras determinísticas e probabilísticas. As regras determinísticas são implicações e dependências funcionais. Já as regras probabilísticas são regras de associação e regras de classificação.

O capítulo 3 dedicou-se à apresentação das regras determinísticas. Nesse capítulo, foram formalizadas as definições de implicações e dependências funcionais. Mostrou-se também que bases de dados relacionais podem ser transformadas em contextos formais, assim, os métodos baseados em AFC podem ser utilizados para a extração de dependências funcionais.

No capítulo 3, foi demonstrado também que o problema da descoberta de dependências funcionais em contextos multivalorados é equivalente ao problema da descoberta de implicações em contextos univalorados. Foram apresentadas duas técnicas com as quais os algoritmos baseados em AFC, inicialmente projetados para descoberta de implicações, podem ser utilizados para a extração de dependências funcionais. A primeira técnica consiste na transformação de contextos. A segunda sugere a utilização de novos operadores de fecho obtidos com a composição dos chamados operadores de particionamento.

No capítulo 3, ainda são apresentadas algumas coberturas de regras determinísticas. Coberturas de regras são conjuntos de regras com propriedades específicas. Foram apresentados diversos tipos de coberturas cujas regras possuíam características distintas.

Seguindo, foram apresentados, no capítulo 3, quatro algoritmos para a extração de regras determinísticas. Os algoritmos que foram, inicialmente, desenvolvidos para a extração de implicações, puderam, também, ser usados para a extração de dependências funcionais utilizando-se as técnicas comentadas anteriormente. Os algoritmos apresentados foram o *Next Closure*, o *Find Implications*, o Impec e o Aprem-IR.

O *Next Closure* extrai uma cobertura mínima de regras. O algoritmo foi proposto inicial-

mente para a descoberta de conceitos formais de contextos formais e foi, em seguida, adaptado para a extração de implicações.

O *Find Implications* extrai uma cobertura reduzida e redundante de regras. Ele baseia-se na construção de reticulados conceituais. Como apresentado, ele executa uma busca em largura no reticulado conceitual para a extração das implicações.

Já o Aprem-IR e o Impec extraem uma cobertura própria. Esses algoritmos foram propostos com o intuito de extrair regras que fossem mais legíveis para usuários finais leigos. A base mínima, apesar de fornecer o menor número de regras possível, faz com que a análise das regras seja difícil, necessitando a aplicação de axiomas para a descoberta das relações entre os atributos. A cobertura própria torna os relacionamento mais explicítos. Por outro lado, ela aumenta o número de regras na base. Apesar de tanto o Aprem-IR quanto o Impec fornecerem o mesmo tipo de base, eles baseiam-se em conceitos diferentes. O Aprem-IR utiliza reticulados conceituais para a extração das regras e o Impec o faz diretamente de contextos formais.

Finalizando o capítulo 3, os algoritmos foram comparados sob o ponto de vista prático. O desempenho dos algoritmos foi investigado variando-se a densidade de contextos e o número de objetos. Além disso, o desempenho dos algoritmos para a extração de dependências funcionais foi avaliado tanto com a utilização dos operadores de particionamento quanto com a transformação de contextos.

Observou-se que os algoritmos são sensíveis ao aumento da densidade dos contextos; principalmente os métodos baseados em reticulados conceituais como o Aprem-IR e o *Find Implications*. O desempenho desses algoritmos foi afetado pelo aumento da densidade pois, ao aumentar a densidade do contexto, ele se aproxima do pior caso quando, do contexto, são obtidos $2^{|G|}$ conceitos. Um contexto gera um número exponencial de conceitos se ele for equivalente ao contexto $(G, G, \neq)$. A densidade do contexto $(G, G, \neq)$ é de 75%. Logo, ao aumentar a densidade dos contextos, eles se aproximam do contexto $(G, G, \neq)$; o número de conceitos aumenta; conseqüentemente, aumenta o tempo de execução dos algoritmos baseados em reticulados conceituais.

Os algoritmos, entretanto, mostraram-se menos sensíveis ao aumento do número de objetos. Os melhores desempenhos foram alcançados com o Aprem-IR e o *Next Closure* que tiveram tempo de execução abaixo de 1 minuto mesmo para contextos com grandes quantidades de objetos. O pior desempenho obtido foi com o *Find Implications*. O algoritmo mostrou-se muito sensível ao aumento de objetos no contexto.

Analisando-se os resultados dos algoritmos tanto para a variação da densidade dos contextos quanto para a variação da quantidade de objetos, constatou-se que, em geral, os métodos

que utilizam reticulados conceituais diretamente na extração de implicações são melhores para contextos densos. Observou-se também que o *Next Closure* obteve o melhor desempenho tanto para variação da densidade quanto para a variação do número de objetos, embora não utilize reticulados conceituais diretamente. O desempenho do *Next Closure* foi surpreendente já que o algoritmo é o mais simples entre os apresentados.

Os mesmos algoritmos foram comparados para a extração de dependências funcionais utilizando-se tanto a transformação de contextos quanto os operadores de particionamento. Contudo, foi constatado que o *Find Implications* não poderia ser usado para a extração de dependências funcionais utilizando os operadores de particionamento já que o algoritmo utilizado para a construção do reticulado conceitual baseia-se nos operadores de derivação. Com isso, o algoritmo deveria ser modificado para que os novos operadores de fecho pudessem ser utilizados. Entretanto, optou-se por não modificar o algoritmo e, assim, o *Find Implications* não pôde ser testado com os operadores de particionamento.

Os algoritmos foram avaliados com contextos com diferentes quantidades de objetos e atributos. Observou-se que, em geral, o desempenho dos algoritmos com os operadores de particionamento foi inferior em relação à transformação de contextos. Esse fato deve-se ao alto custo computacional dos operadores de particionamento quando comparados ao custo dos operadores de derivação. Além do custo dos operadores, outro fator que influenciou o desempenho dos algoritmos utilizando os operadores de particionamento foi o fato deles terem sido projetados com base nos operadores de derivação. Isso pode ser observado quando compara-se os tempos do *Next Closure* utilizando os operadores de particionamento e a transformação de contextos. Ele foi o algoritmo que obteve as menores diferenças entre tempo de execução com a transformação de contextos e com os operadores de particionamento, seguido do Aprem-IR e do Impec.

A comparação entre o tempo de execução com a transformação e com os operadores de particionamento também revelou que o Aprem-IR, embora tenha sido projetado para funcionar com qualquer operador de fecho, obteve melhores resultados quando utilizado com os operadores de fechos obtidos pela composição dos operadores de derivação.

Ao investigar o impacto do aumento do número de atributos no tempo de execução dos algoritmos, constatou-se que o Impec foi o algoritmo mais sensível a tal mudança tanto com a utilização da transformação de contextos quanto com os operadores de particionamento. Em média, o aumento do número de atributos de 7 para 10 causou um aumento de 60 vezes no tempo de execução quando utilizou-se a transformação de contextos e 54 vezes quando utilizou-se os operadores de particionamento. Os piores resultados são obtidos quando o número de objetos é pequeno; o pior resultado para a transformação de contexto ocorreu com 50 objetos

e para os operadores de particionamento ocorreu com 100 objetos. O algoritmo que menos sofreu com o aumento do número de atributos utilizando a transformação de contexto foi o *Find Implications* que, em média, teve aumento no tempo de execução de 2 vezes. Já com a utilização dos operadores de particionamento ocorreu empate entre os algoritmos que obtiveram o melhor resultado; o *Next Closure* e o Aprem-IR tiveram aumento de 17 vezes em média. Isso mostra a superioridade dos métodos que utilizam reticulado conceituais diretamente para a extração de dependências funcionais.

Analisando-se o desempenho dos algoritmos tanto para a extração de implicações quanto para a extração de dependências funcionais percebe-se que, embora seja possível, o uso dos algoritmos para a extração dependências funcionais não é adequado. Mesmo em testes com poucos atributos e objetos, o desempenho dos algoritmos para a extração de dependências funcionais não é satisfatório. Isso revela a tendência constatada na literatura da falta de pesquisa de métodos baseados em AFC dedicados à extração de dependências funcionais. Em geral, os autores em suas pesquisas desenvolvem métodos para a extração de implicações e sugerem a possibilidade de sua utilização para a extração de dependências funcionais através das duas técnicas apresentadas aqui. Contudo, como verificado nesse trabalho, essas técnicas não mostraram-se eficazes. Logo, verifica-se a necessidade de pesquisas para o desenvolvimento de novas técnicas que permitam mapear o problema da extração de dependências funcionais no problema da extração de implicações ou para o desenvolvimento de algoritmos para a extração direta de dependências funcionais a partir de contextos formais.

Após apresentar tópicos relacionados às regras determinísticas, no capítulo 4, foram apresentadas as regras probabilísticas.

Quatro algoritmos para a extração de regras de associação foram apresentados; o *AClose*, o *Titanic*, o *Frequent Next Neighbours* e o Galicia. Os dois primeiros foram desenvolvidos com base no algoritmo *Apriori*. O *Frequent Next Neighbours* assemelha-se ao algoritmo *Find Implications* e utiliza reticulados conceituais para a extração das regras. O algoritmo Galicia é um algoritmo incremental e também utiliza reticulados conceituais para extração das regras. No entanto, ele difere-se do *Frequent Next Neighbours* por utilizar o reticulado conceitual apenas para encontrar o conjunto de itens freqüentes, ao contrário do FNN que utiliza a estrutura do reticulado para encontrar também as regras.

Após a apresentação dos algoritmos para a extração de regras de associação, foram apresentados os algoritmos GRAND e *Rulearner* para a extração de regras de classificação. Existem outros algoritmos baseados em AFC para classificação além dos dois apresentados. Todavia, a maioria deles utiliza a estrutura do reticulado para classificar. Como o interesse era em algoritmos que produzissem regras de classificação, optou-se por não utilizá-los. Além desses

métodos para classificação, os algoritmos para a descoberta de regras de associação também podem ser usados para a extração de regras de classificação. Mais uma vez, como eles não foram desenvolvidos com o intuito de classificar, eles não foram comparados com tal propósito.

Os algoritmos para a extração de regras de associação foram comparados, assim como os de implicações, sob a variação da densidade de contextos e, também, sob a variação no suporte mínimo das regras. Foram utilizadas bases de dados sintéticas e reais. As sintéticas foram geradas conforme metodologia proposta por Agrawal [3] e as reais foram retiradas do repositório de dados da Universidade da Califórnia em Irvine [22].

O algoritmo Galicia não foi, em geral, comparado com os demais por ser um algoritmo incremental e, com isso, demandar muito tempo de execução com contextos maiores. O Galicia, como recomendado por seus próprios autores, é mais adequado quando a base de dados é alterada com a inclusão de novas transações (objetos).

A variação da densidade dos contextos, em geral, influenciou negativamente no desempenho dos algoritmos. À medida que a densidade dos contextos aumenta, o tempo de execução também aumenta. O algoritmo que obteve o pior desempenho com o aumento da densidade foi o *Titanic*, seguido do *AClose* e FNN. Os dois primeiros, como já mencionado, foram desenvolvidos com base no *Apriori*. Agrawal [3], ao apresentar o algoritmo, afirmou que o *Apriori* era indicado para bases de dados esparsas. Com isso, naturalmente, tanto o *Titanic* quanto o *AClose* têm o tempo de execução aumentado com o aumento da densidade dos contextos. Obviamente, o FNN também sofre com o aumento da densidade dos contextos. Como já mencionado, o maior número de conceitos é obtido quando a densidade do contexto chega a 75% e o contexto assemelha-se ao $(G, G, \neq)$. Logo, o aumento no tempo de execução do FNN com o aumento da densidade do contexto é retardado para contextos com altas densidades. Esse fato foi confirmado quando se comparou os algoritmos com outro contexto denso como o das doenças hepáticas.

Entretanto, o bom desempenho do FNN não se manteve quando ele foi submetido a contextos com muitos objetos. Ao comparar os algoritmos submetendo-os a contextos com muitos objetos, o desempenho do FNN foi o pior entre ele, o *Titanic* e o *AClose*. O *AClose* obteve o melhor desempenho com os contextos com 10000 e 50000 objetos. Porém, o *Titanic* também obteve desempenho satisfatório com esses contextos. Ao se diminuir o suporte das regras, o tempo de execução do *AClose* e do *Titanic* mantiveram-se praticamente constantes, com pequeno aumento quando o suporte aproximou-se de zero. Já o FNN apresentou crescimento quadrático no tempo de execução com a variação do suporte. Constatou-se que ao diminuir o suporte o supremo-semi-reticulado construído pelo FNN aproxima-se do reticulado conceitual referente ao contexto. Assim, o tempo de execução do FNN é afetado pela construção do

reticulado.

Os testes revelaram a adequação de cada algoritmo a situações específicas. Enquanto o *Titanic* e o *AClose* adequam-se melhor a bases esparsas, os algoritmos baseados em reticulados conceituais, especificamente, o FNN é mais adequado a bases de dados densas sendo menos sensíveis à variação do suporte das regras para essas bases.

O Galicia foi analisado submetendo-o a contextos com diferentes densidades e quantidades de objetos.

Em relação à variação de densidade, o tempo de execução teve o pior caso quando a densidade aproximou-se de 55%. Isso retrata que o tempo de execução do algoritmo para a extração de regras de associação é afetado principalmente pela construção do reticulado conceitual como já era esperado. Como o algoritmo é incremental, os conceitos formais são criados à medida que novos objetos são adicionados. Em contextos densos, a similaridade dos objetos aumenta. Logo, são necessárias menos operações para atualizar o contexto.

Em relação à variação na quantidade de objetos no contexto, o tempo de execução do Galicia aumentou linearmente com o aumento do número de objetos. Mesmo para pequenas quantidades de objetos, o tempo de execução do algoritmo foi alto. Isso confirma a adequação do algoritmo para ser utilizado apenas quando ocorrem pequenas variações na quantidade de objetos do contexto.

Os algoritmos para extração de regras de classificação foram comparados utilizando-se bases de dados reais. As bases escolhidas são amplamente utilizadas na comparação de métodos para classificação. As bases, quando não estavam divididas em conjunto de treinamento e validação, foram divididas de forma que 40% dos objetos, escolhidos aleatoriamente, pertenciam ao conjunto de treinamento e o restante ao conjunto de validação.

Os algoritmos foram avaliados não só sob o ponto de vista do tempo de execução mas também sob o ponto de vista da precisão das regras produzidas.

Os algoritmos apresentaram precisão média de 73% e tempo de execução médio de cerca de 11000 segundos. O desempenho dos algoritmos foi, em geral, satisfatório tanto do ponto de vista do tempo de execução quanto da precisão. Contudo, eles não obtiveram bom desempenho para a base de dados de métodos contraceptivos quando apresentaram o maior tempo de execução e a pior precisão na classificação de novos objetos. Essa base apresenta atributos não-binários como, por exemplo, a idade das pessoas. Esses atributos foram removidos da base quando ela foi transformada em contexto formal. Essa pode ter sido a causa do desempenho ruim dos algoritmos. Esse fato revela também uma deficiência não só dos métodos de extração de regras de classificação baseados em AFC mas também de toda a técnica; o fato dela trabalhar

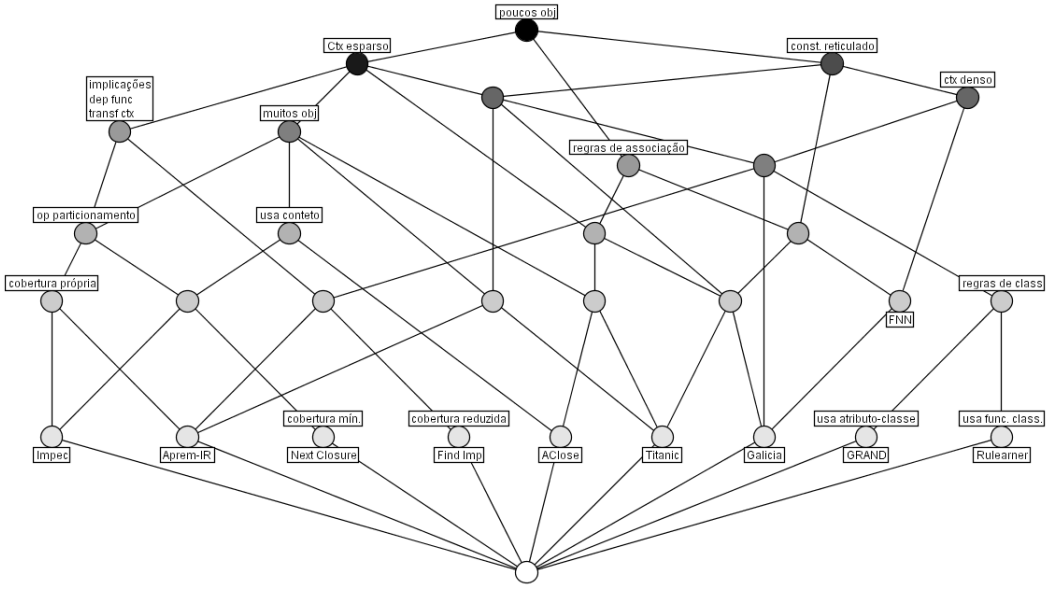

Figura 16: Resumo dos algoritmos

apenas com atributos binários. Isso restringe a utilização dos métodos a situações em que os atributos são binários ou forçam a discretização de atributos com valores contínuos o que nem sempre conduz a resultados satisfatórios.

Sob o ponto de vista do tempo de execução, o desempenho dos algoritmos não foi satisfatório. Os algoritmos são incrementais e, com isso, são extremamente sensíveis ao aumento do número de objetos no contexto. Isso demonstra certa falta de pesquisa de métodos baseados em AFC para a classificação. Em geral, os algoritmos são antigos, propostos na década de 80 e 90 e por isso obtiveram tempo de execução tão altos. Contudo, a precisão deles foi satisfatória e comprovou que reticulados conceituais podem ser utilizados para classificação.

Finalmente, a Figura 16 apresenta um resumo das características do dez algoritmos retromencionados. Construiu-se um reticulado conceitual a partir de um contexto formal em que os objetos são os algoritmos e os atributos suas características. As características consideradas são: a adequação do algoritmo a contextos formais densos e esparsos ou com muitos ou poucos objetos; se ele usa o contexto formal ou o reticulado conceitual para extrair as regras; o tipo de regra extraído (implicações, dependências funcionais, regras de associação ou classificação); o tipo de cobertura extraída no caso de implicações e dependências funcionais; se é possível usar transformação de contextos ou operadores de particionamento para extrair dependências funcionais; e se o algoritmo usa os atributos-classe ou funções de classificação para gerar regras de classificação.

Referências

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo. Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in knowledge discovery and data mining*, pages 307–328. AAAI Press, 1996.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th Very Large Data Bases Conference*, pages 487–499, 1994.

[4] W. W. Armstrong. Dependency structures of data base relationships. In *Proceedings of 1974 IFIP Congress*, pages 580–583, 1974.

[5] J. Baixeries. A formal concept analysis framework to mine functional dependencies. In *Mathematical Methods for Learning*, Villa Geno, Como, Italy, 2004.

[6] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, and L. Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. In *Computational Logic*, pages 972–986, 2000.

[7] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal. Mining frequent patterns with counting inference. *SIGKDD Exploration Newsletter*, 2(2):66–75, 2000.

[8] C. Beeri and P. A. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Transactions on Database Systems*, 4(1):30–59, 1979.

[9] C. Beeri, R. Fagin, and J. H. Howard. A complete axiomatization for functional and multivalued dependencies. In *ACM SIGMOD International Conference on Management of Data*, pages 45–62, USA, 1983. ACM Press.

[10] G. Birkhoff. *Lattice Theory*. American Mathematical Society, 1940.

[11] J. P. Bordat. Calcul pratique du treillis de Galois d'une correspondance. *Mathématiques et Sciences Humaines*, (96):31–47, 1986.

[12] C. Carpineto and G. Romano. GALOIS: An order-theoretic approach to conceptual clustering. *International Conference on Machine Learning*, pages 33–40, 1993.

[13] C. Carpineto and G. Romano. Mining short-rule covers in relational databases. *Computational Intelligence*, 19(3):215–234, 2003.

[14] C. Carpineto and G. Romano. *Concept Data Analysis: Theory and Applications*. John Wiley & Sons, England, 2004.

[15] C. Carpineto, G. Romano, and P. d'Adamo. Inferring dependencies from relations: A conceptual clustering approach. *Computational Intelligence*, 15(4):415–441, 1999.

[16] E. F. Codd. A relational model for large shared database banks. *Communications ACM*, 6(16):377–387, 1970.

[17] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. Finding interesting associations without support pruning. *Knowledge and Data Engineering*, 13(1):64–78, 2001.

[18] C. J. Date. *Introduction to Database Systems*. Addison-Wesley, USA, 2000.

[19] T. H. Davenport and L. Prusak. *Conhecimento empresarial: como as organizações gerenciam o seu capital intelectual*. Campus, 1998.

[20] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge Mathematical Textbooks, England, 2nd edition, 2001.

[21] A. B. de Holanda. *Novo dicionário da língua portuguesa*. Nova Fronteira, 2a edition, 1986.

[22] C. B. D.J. Newman, S. Hettich and C. Merz. UCI repository of machine learning databases. internet, 1998.

[23] V. Duquenne and J. L. Guigues. Familles minimales d'implications informatives resultant d'un tableau de données binaires. *Mathématiques et Sciences Humaines*, pages 5–18, 1986.

[24] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Benjamin/Cummings, Redwood City, CA, USA, second edition, 1994.

[25] H. Fu, H. Fu, P. Njiwoua, and E. M. Nguifo. A comparative study of FCA-based supervised classification algorithms. In P. Ecklund, editor, *Concept Lattices, Second International Conference on Formal Concept Analysis, ICFCA 2004*, volume 2961 of *Lecture Notes in Computer Science*, Sydney, Australia, February 2004. Springer.

[26] J.-G. Ganascia. Charade: A rule system learning system. In *International Joint Conferences on Artificial Intelligence*, pages 345–347, 1987.

[27] B. Ganter. Two basic algorithms in concept analysis. Technical report, TU Darmstadt, Germany, 1984.

[28] B. Ganter. Formal concept analysis: algorithmic aspects. lecture notes, TU Dresden, Germany, 2002.

[29] B. Ganter and S. O. Kuznetsov. Formalizing hypotheses with concepts. In B. Ganter and G. W. Mineau, editors, *ICCS*, volume 1867, Darmstadt, Germany, August 2000. Springer.

[30] B. Ganter and R. Wille. Applied lattice theory: Formal concept analysis. In G. Grätzer, editor, *General lattice theory*, pages 591–605, Germany, 1998.

[31] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, 1999.

[32] H. Garcia-Molina, J. Ullman, and J. Widom. *Database systems: the complete book*. Prentice Hall, USA, 2002.

[33] R. Godin, R. Missaoui, and H. Alaoui. Incremental concept formation algorithms based on Galois (concept) lattices. *Computational Intelligence*, 2(11):246–267, 1995.

[34] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 2(42):100–111, 1999.

[35] B. Koester. Conceptual knowledge retrieval with FooCA: Improving web search engine results with contexts and concept hierarchies. In P. Perner, editor, *Industrial Conference on Data Mining*, volume 4065 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2006.

[36] M. Luxenburger. Implications partielles dans un contexte. *Mathématiques, Informatique et Sciences Humaines*, 113(29):35–55, 1991.

[37] D. Maier. *The theory of relational databases*. Computer Science Press, 1983.

[38] H. Mannila and K.-J. Raiha. *The Design of Relational Databases*. Addison-Wesley, USA, 1992.

[39] E. M. Nguifo. Galois lattice: A framework for concept learning-design, evaluation and refinement. In *International Conference on Theretical Artificial Intelligence*, pages 461–467, 1994.

[40] E. M. Nguifo and P. Njiwoua. IGLUE: A lattice-based constructive induction system. *Intelligent Data Analysis*, 5(1):73–91, 2001.

[41] E. M. Nguifo and P. Njiwoua. Treillis de concepts et classification supervisée. *Technique et Science Informatiques*, 24(4):449–488, 2005.

[42] L. Nourine and O. Raynaud. A fast algorithm for building lattices. *Information Processing Letters*, 71(5-6):199–204, 1999.

[43] G. D. Oosthuizen. *The use of a lattice in knowledge processing*. PhD thesis, University of Strathclyde, 1988.

[44] G. D. Oosthuizen and D. R. McGregor. Induction through knowledge base normalization. In *Proceedings of the 8th European Conference on Artificial Intelligence*, pages 396–401, 1988.

[45] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT '99: Proceeding of the 7th International Conference on Database Theory*, pages 398–416, London, UK, 1999. Springer-Verlag.

[46] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25–46, 1999.

[47] M. Sahami. Learning classification rules using lattices. In N. Lavrac and S. Wrobel, editors, *Machine Learning: ECML-95, 8th European Conference on Machine Learning*, pages 343–346. Springer, 1995.

[48] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.

[49] G. Stumme. Formal concept analysis on its way from Mathematics to Computer Science. In *ICCS*, Lecture Notes in Computer Science, pages 2–19. Springer, 2002.

[50] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal. Computing iceberg concept lattices with TITANIC. *Data Knowledge Engineering*, 42(2):189–222, 2002.

[51] G. Stumme, R. Wille, and U. Wille. Conceptual knowledge discovery in databases using formal concept analysis methods. In *Proceedings of the second european symposium on principles of data mining and knowledge discovery*, pages 450–458, Nantes, France, 1998.

[52] R. Taouil and Y. Bastide. Computing proper implications. In *ICCS-2001 International Workshop on Concept Lattice-Based Theory, Methods and Tools for Knowledge Discovery in Databases*, Lecture Notes in Computer Science, pages 49–61, Palo Alto, CA, USA, 2001. Springer.

[53] R. Taouil, N. Pasquier, Y. Bastide, and L. Lakhal. Computing closed set lattices: Algorithms, applications and performances. Technical report, Université Nice – Sophia Antipolis, 2003.

[54] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 2. Computer Science Press Inc., 1988.

[55] P. Valtchev, R. Missaoui, R. Godin, and M. Meridji. Generating frequent itemsets incrementally: two novel approaches based on Galois lattice theory. *Journal of Experimental and Theoretical Artificial Intelligence*, 14(2–3):115–142, 2002.

[56] K. Wang, Y. He, D. W.-L. Cheung, and F. Chin. Mining confident rules without support requirement. In *CIKM*, pages 89–96, 2001.

[57] R. Wille. Restructuring the lattice theory. In I. Rival, editor, *Ordered Sets*, pages 445–470, Dordrecht-Boston, 1982. Reidel.

[58] R. Wille. Concept lattices and conceptual knowledge systems. *Computers and Mathematics with Applications*, (23):493–515, 1992.

[59] R. Wille. Why can concept lattices support knowledge discovery in databases? In *Proceedings of the concept lattices based knowledge discovery in databases workshop*, pages 7–20, Stanford, United States, 2001.

[60] I. H. Witten and E. Frank. *Data mining: practical machine learning tools and techniques with JAVA implementations*. Morgan Kaufmann Publishers, 2000.

[61] M. J. Zaki. Generating non-redundant association rules. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 34–43, New York, NY, USA, 2000. ACM Press.

[62] M. J. Zaki and M. Ogihara. Theoretical foundations of association rules. In *Proceedings of the 3rd SIGMOD'98 Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 71–78, 1998.