PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Programa de Pós-Graduação em Informática

Phillip Gonçalves Santos

EXPLORANDO DIFERENTES PARADIGMAS PARA EXTRAÇÃO DE IMPLICAÇÕES PRÓPRIAS A PARTIR DE CONTEXTOS FORMAIS DE ALTA DIMENSIONALIDADE

Belo Horizonte

2018

Phillip Gonçalves Santos

EXPLORANDO DIFERENTES PARADIGMAS PARA EXTRAÇÃO DE IMPLICAÇÕES PRÓPRIAS A PARTIR DE CONTEXTOS FORMAIS DE ALTA DIMENSIONALIDADE

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Informática.

Orientador: Mark Alan Junho Song

Belo Horizonte

2018

FICHA CATALOGRÁFICA
Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais

Santos, Phillip Gonçalves
S237e Explorando diferentes paradigmas para extração de implicações próprias a partir de contextos formais de alta dimensionalidade / Phillip Gonçalves Santos. Belo Horizonte, 2018.
73 f.: il.

Orientador: Mark Alan Junho Song
Dissertação (Mestrado) – Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Informática

1. Software - Desenvolvimento. 2. Programação orientada a objetos (Computação). 3. Engenharia de software. 4. Programação paralela (Computação). 5. Conceitos - Análise. I. Song, Mark Alan Junho. II. Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Informática. III. Título.

SIB PUC MINAS

CDU: 681.3.011

Ficha catalográfica elaborada por Fernanda Paim Brito– CRB 6/2999

Phillip Gonçalves Santos

EXPLORANDO DIFERENTES PARADIGMAS PARA EXTRAÇÃO DE IMPLICAÇÕES PRÓPRIAS A PARTIR DE CONTEXTOS FORMAIS DE ALTA DIMENSIONALIDADE

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Informática.

Mark Alan Junho Song – (PUC Minas)

Luís Enrique Zárate Gálvez – (PUC Minas)

Sérgio Vale Aguiar Campos – (UFMG)

Sérgio Mariano Dias – (SERPRO e UFMG)

Belo Horizonte, 25 de Abril de 2018

Aos meus pais e minha irmã, por sempre acreditarem em mim.

AGRADECIMENTOS

Gostaria de agradecer aos meus pais por sempre terem me apoiado e minha irmã que está sempre ao meu lado. Também gostaria de agradecer o professor Dr. Mark Alan pela orientação e confiança depositada em mim. Agradeço também ao Dr. Luis Zárate e Sérgio M. Dias pela ajuda em diversos momentos de dúvidas. Também agradeço a Giovana pela disponibilidade e facilidade para lidar com os trâmites internos, e a todos colegas e amigos da PUC que me ajudaram de alguma maneira a concluir mais está etapa da minha vida.

"A lei da mente é implacável.
O que você pensa, você cria;
O que você sente, você atrai;
O que você acredita, torna-se realidade."

Buda

RESUMO

A Análise Formal de Conceitos (AFC) é uma técnica da matemática aplicada para análise de dados em que relações entre objetos e atributos são identificados. Esta é baseadas em noções de conceitos e na estruturação (hierarquia). Uma parte do conhecimento que pode ser obtido através desta representação é o conjunto de implicações entre atributos que caracterizam o problema. No entanto, o volume de informações a ser processado pode tornar inviável o uso desta técnica. Assim, há demanda por novas soluções e algoritmos para possibilitar tal processamento. Este artigo explora diferentes abordagens para extrair implicações próprias em contextos de alta dimensionalidade baseado no algoritmo $PropIm$. O algoritmo proposto neste artigo utiliza a estrutura de dados denominada Diagrama Binário de Decisão (BDD) para representar um contexto formal de forma reduzida e realizar operações sobre este contexto. Além disso, propõe uma otimização para reduzir a geração desnecessária de premissas para a obtenção das devidas implicações e também implementa um modelo de computação paralela para a geração e obtenção simultânea das diferentes regras de implicação. Para analisar cada solução proposta foi desenvolvido um algoritmo e realizados diferentes testes utilizando contextos sintéticos em que se varia o número de objetos, atributos e densidade. Os resultados demonstraram ganhos de até 22 vezes se comparado a solução original ($PropIm$).

Palavras-chave: AFC, BDD, Implicações Próprias, Diagrama Binário de Decisão, Análise Formal de Conceitos

ABSTRACT

Formal concept analysis (FCA) is a applied mathematical technique for data analysis in which relationships between objects and attributes are identified. This is based on notions of concepts and conceptual hierarchy. A part of the knowledge that can be obtained through this representation is the set of implications between attributes that characterize the problem. However, the volume of information to be processed may make the use of this technique impracticable. Thus, there is a demand for new solutions and algorithms to enable such processing. This article explores different approaches to extracting own implications in high dimensional contexts based on the $PropIm$ algorithm. The algorithm proposed in this article uses the data structure called Decision Binary Diagram (BDD) to represent a formal context in a reduced form and perform operations on this context. In addition, it proposes an optimization to reduce the unnecessary generation of premises to obtain the necessary implications and also implements a model of parallel computation for the generation and simultaneous obtaining of the different rules of implication. To analyze each proposed solution an algorithm was developed and different tests were performed using synthetic contexts in which the number of objects, attributes and density were varied. The results showed gains of up to 22 times when compared to the original solution ($PropIm$).

Keywords: FCA, BDD, Proper implications, Formal concept analysis, Binary Decision Diagram

LISTA DE FIGURAS

FIGURA 1 – Comparador de dois bits 26
FIGURA 2 – BDT (Árvore Binária de Decisão) 26
FIGURA 3 – Remoção de nós terminais duplicados 27
FIGURA 4 – Remoção de nós não terminais duplicados 27
FIGURA 5 – Remoção de redundância 27
FIGURA 6 – BDD (Diagrama Binario de Decisão) 28
FIGURA 7 – BDD gerado baseado no contexto da Tabela 1. 28
FIGURA 8 – Modelo de computação Singlecore e Multicore 30
FIGURA 9 – Conclusão - Terrestre 43
FIGURA 10 – Premissa - Penas 43
FIGURA 11 – Interseção entre Terrestre e Penas 43
FIGURA 12 – Implicação: Penas igual a Interseção entre Terrestre e Penas 43
FIGURA 13 – Modelo de computação paralela do algoritmo PImplicPBDD 48

LISTA DE TABELAS

TABELA 1 – Contexto formal: as linhas são os atributos e as colunas são os objetos 20

TABELA 2 – Conceitos formais gerados da Tabela 1 21

TABELA 3 – Regras de implicação extraídas da Tabela 1 22

TABELA 4 – Conjunto de implicações próprias extraído da Tabela 1 22

TABELA 5 – Contexto formal simples 35

TABELA 6 – Premissas geradas na primeira iteração 35

TABELA 7 – Premissas candidatas geradas na primeira iteração 35

TABELA 8 – Premissas geradas na segunda iteração 36

TABELA 9 – Premissas candidatas geradas na segunda iteração 36

TABELA 10 – Premissas geradas na terceira iteração 36

TABELA 11 – Premissas candidatas geradas na terceira iteração 36

TABELA 12 – Resultados (PropIm) X (ImplicP) utilizando contextos sintéticos 50

TABELA 13 – Resultados (PropIm) X (ImplicPBDD) utilizando contextos sintéticos 53

TABELA 14 – Resultados (ImplicP) X (ImplicPBDD) utilizando contextos sintéticos 55

TABELA 15 – Resultados (ImplicP) X (ImplicPBDD) utilizando contextos sintéticos com uma maior volume de objetos 57

TABELA 16 – Resultados (PropIm) X (PImplicPBDD) utilizando contextos sintéticos 58

TABELA 17 – Resultados (ImplicPBDD) X (PImplicPBDD) utilizando contextos sintéticos 60

TABELA 18 – Amostragem de implicações próprias extraídas do contexto formal do Linkedin 62

TABELA 19 – Habilidades utilizadas no contexto formal do Linkedin 63

TABELA 20 – Profissões utilizadas no contexto formal do Linkedin 64

TABELA 21 – Quantidade de atributos nas premissas das regras de implicações . . . 65

TABELA 22 – Comparativo do aumento de velocidade entre todos os algoritmos . . . 67

LISTA DE GRAFICOS

GRÁFICO 1 – (PropIm) X (ImplicP) 16 atributos 51
GRÁFICO 2 – (PropIm) X (ImplicP) 20 atributos 52
GRÁFICO 3 – (PropIm) X (ImplicPBDD) 16 atributos 54
GRÁFICO 4 – (PropIm) X (ImplicPBDD) 20 atributos 54
GRÁFICO 5 – (ImplicP) X (ImplicPBDD) 16 atributos 56
GRÁFICO 6 – (ImplicP) X (ImplicPBDD) 20 atributos 56
GRÁFICO 7 – (PropIm) X (PImplicPBDD) 16 atributos 59
GRÁFICO 8 – (PropIm) X (PImplicPBDD) 20 atributos 59
GRÁFICO 9 – (ImplicPBDD) X (PImplicPBDD) 16 atributos 61
GRÁFICO 10 – (ImplicPBDD) X (PImplicPBDD) 20 atributos 61
GRÁFICO 11 – Todos os algoritmos com 16 atributos 66
GRÁFICO 12 – Todos os algoritmos com 20 atributos 66

LISTA DE ABREVIATURAS E SIGLAS

AFC – *Análise Formal de Conceitos*

BDD – *Diagrama Binário de Decisão*

CPU – *Unidade Central de Processamento*

JNI – *Interface Nativa do Java*

SUMÁRIO

1 INTRODUÇÃO **15**
1.1 Justificativa 17
1.2 Objetivos 17
1.3 Organização do Trabalho 18

2 REFERENCIAL TEÓRICO **19**
2.1 Análise Formal de Conceito 19
2.1.1 Contexto Formal 19
2.1.2 Conceitos Formais 20
2.1.3 Reticulado Conceitual 21
2.1.4 Implicações 21
2.1.5 Implicações Próprias 22
2.1.5.1 Impec 23
2.1.5.2 PropIm 24
2.2 Diagrama binário de decisão 25
2.3 Paralelismo 29

3 TRABALHOS RELACIONADOS **32**

4 METODOLOGIA **34**
4.1 Algoritmo ImplicP 34
4.1.1 Ordem de complexidade na geração de premissas 39
4.2 Algoritmo ImplicPBDD 40
4.2.1 Processo interno do BDD para obter uma implicação 43
4.3 Algoritmo PImplicPBDD 44

5 EXPERIMENTOS E ANÁLISE DOS RESULTADOS **49**
5.0.1 Resultados ImplicP 50
5.0.1.1 PropIm *vs.* ImplicP 50

5.0.2 Resultados ImplicPBDD 52
5.0.2.1 PropIm *vs.* ImplicPBDD 52
5.0.2.2 ImplicP *vs.* ImplicPBDD 55
5.0.3 Resultados PImplicPBDD 57
5.0.3.1 PropIm *vs.* PImplicPBDD 57
5.0.3.2 ImplicPBDD *vs.* PImplicPBDD 59
5.0.3.3 PropIm *vs.* PImplicPBDD em um contexto formal real 62
5.0.3.4 Utilização do Algoritmo PImplicPBDD em um contexto formal real utilizando ordenação de variáveis com o BDD 62
5.0.4 Impacto da quantidade de premissas no tempo de execução 64
5.0.5 Visão geral da execução de tempo de todos os algoritmos 65

6 CONCLUSÕES E TRABALHOS FUTUROS **68**
6.1 Publicações 69

REFERÊNCIAS **70**

1 INTRODUÇÃO

Com o avanço da tecnologia, o volume de informação coletado e armazenado pelas aplicações cresceu exponencial tornando impraticável a análise dos dados de forma manual (DAVENPORT; PRUSAK, 1997). Com isto, tornam-se necessárias técnicas para automatizar ou auxiliar o processo de obtenção de informações a partir de dados.

Diferentes técnicas têm sido propostas nesta área, nas quais baseiam-se em sua maioria, em conceitos estatísticos, computação natural e bio-inspiradas. Em geral, essas técnicas, apesar de distintas, possuem pontos em comum, como o fato de utilizarem a distribuição de probabilidades para analisarem os dados (VIMIEIRO, 2007). Alternativamente, uma das técnicas que vem se consolidando para a extração e análise dos dados é a *Análise Formal de Conceitos* (AFC) (WILLE, 2009).

A AFC é um campo da matemática criado para análise de dados em que associações e dependências entre objetos e atributos são identificados. Dessa forma, acredita-se que os métodos baseados na AFC podem propiciar melhores resultados na análise dos dados por realçar as relações estruturais (STUMME; WILLE; WILLE, 1998; WILLE, 2001).

Na Análise Formal de Conceito (AFC), uma maneira de representar um conjunto de dados composto de objetos e suas características é através de um contexto formal (G, M, I) representado por uma tabela de incidência, sendo G os objetos, M os atributos e I a relação de incidência entre objetos e atributos (GANTER; WILLE, 1999).

Um conhecimento que pode ser extraído do contexto formal é o conjunto de implicações $\mathcal{I}$ entre os atributos que definem o contexto formal (TAOUIL; BASTIDE, 2001). A implicação $P \rightarrow Q$ $(P, Q \subseteq M)$ revela que cada objeto contendo os atributos pertencentes ao conjunto P possui também os atributos do conjunto Q. Os conjuntos P e Q são considerados, respectivamente, premissa e conclusão.

Aplicando diferentes propriedades a um conjunto $\mathcal{I}$, podem ser gerados conjuntos de implicações com restrições para domínios específicos. Um desses conjuntos específicos é o conjunto de IMPLICAÇÕES PRÓPRIAS sendo que, para cada implicação a premissa é mínima e a conclusão é unitária (TAOUIL; BASTIDE, 2001).

Esse conjunto de implicações é útil porque fornece um tipo de representação mínima sobre os dados, quando o objetivo é encontrar o conjunto mínimo de atributos para alcançar propósitos específicos. Esse tipo de implicação pode ser utilizado em várias aplicações, como detecção de cópia (BRODER, 1997; SHIVAKUMAR; GARCIA-MOLINA,

1996), clustering (GUHA; RASTOGI; SHIM, 1998), filtragem colaborativa (GOLDBERG et al., 1992; AGRAWAL; IMIELINSKI; SWAMI, 1993), rede social (SILVA et al., 2017) e outros.

O algoritmo proposto originalmente por (TAOUIL; BASTIDE, 2001), intitulado *Impec*, é um dos algoritmos mais utilizados para obtenção e extração de implicações próprias a partir do contexto formal. O *Impec* fornece as implicações do tipo $A \rightarrow b$ onde $A, b \subseteq M$, cujo lado esquerdo (premissa) é mínimo e o lado direito (conclusão) possui apenas um atributo. A partir de um conjunto de atributos M o algoritmo encontra a base de implicações próprias. Entretanto, em alguns casos, pode-se desejar que apenas alguns atributos sejam considerados para a conclusão de uma implicação, o que não é possível no algoritmo *Impec*, sem antes gerar todas as implicações e posteriormente a seleção daquelas implicações cuja conclusão em interesse se encontra do lado direito da implicação.

Diferentemente do algoritmo *Impec*, que encontra as implicações próprias considerando todo o conjunto de atributos M com suporte maior ou igual a 0, o algoritmo *PropIm* gera as implicações próprias (também considerando todo o conjunto M de atributos) porém somente com suporte maior que 0. Implicações com suporte igual a 0, são implicações válidas porém, no contexto onde estão sendo extraídas as implicações não existe nenhuma ocorrência desta implicação.

No entanto, os algoritmos *Impec* e *PropIm* tem sua aplicabilidade limitada pela capacidade da AFC de lidar com grande conjuntos de dados (BAIXERIES et al., 2009; MORAES; ZÁRATE; FREITAS, 2010). A AFC induz uma complexidade combinatória potencialmente alta, e mesmo a partir de um pequeno conjunto de dados as estruturas obtidas pode se tornar proibitivamente grande (MORAES et al., 2016). Apesar do fato de que o pior caso do tamanho do reticulado ($2^{min(|G|,|M|)}$) é raramente encontrado na prática, o custo computacional ainda é muito proibitivo para muitas aplicações (MORAES et al., 2016).

Nesse sentido, considerando um contexto de alta dimensionalidade com um grande número de objetos em que deseja-se encontrar implicações próprias especificando um subconjunto C de conclusões, onde $C \subseteq M$ e todas as regras com suporte maior que 0, esse trabalho apresenta três algoritmos baseados no algoritmo *PropIm* de (SILVA et al., 2017) sendo que, o algoritmo *ImplicP* contém uma otimização para evitar a geração desnecessária de combinações de premissas para serem avaliadas. O algoritmo *ImplicPBDD* contém a solução proposta no *ImplicP* incluindo a estrutura de dados BDD para ser possível representar e manipular o contexto formal de forma mais eficiente. Por fim, o algoritmo *PImplicPBDD* contém as soluções dos algoritmos anteriores incluindo um modelo de computação paralela para ser possível processar múltiplas conclusões simultaneamente.

Para analisar o desempenho dos algoritmos *ImplicP*, *ImplicPBDD* e *PImplicPBDD*, foram realizados experimentos utilizando contextos sintéticos e também um contexto de um problema real. Decidimos utilizar contextos sintéticos para ser possível controlar os experimentos e produzir uma análise de dados satisfatória. Para avaliar esta abordagem, os contextos foram gerados variando o número de atributos, objetos e densidade. Para demonstrar o desempenho do algoritmo em um problema real, foi realizado teste em um contexto formal criado baseado em uma amostragem do Linkedin.

1.1 Justificativa

Há um enorme interesse na comunidade de apresentar soluções eficientes para possibilitar resolver um problema aberto na AFC: processar contextos densos e de alta dimensionalidade (GÉLY et al., 2005; GÉLY; MEDINA; NOURINE, 2010). Este problema também ocorre quando é necessário gerar regras de implicação utilizando contextos de alta dimensionalidade. Na literatura existem diferentes soluções e algoritmos para extrair regras de implicações próprias (SILVA et al., 2017) e (TAOUIL; BASTIDE, 2001), porém, estes algoritmos não possuem um desempenho satisfatório para contextos de alta dimensionalidade.

Até o momento não há algoritmos ligados a AFC que utilizam BDD para obtenção de regras de implicação. Há a utilização do BDD na geração de contextos formais conforme apresentado em (RIMSA; ZÁRATE; SONG, 2009) e (NETO, 2016). Em (YEVTUSHENKO, 2002) foram propostos dois algoritmos utilizando BDD, porém somente para a manipulação do reticulado.

1.2 Objetivos

O objetivo desta dissertação foi propor diferentes soluções para a redução do custo computacional na extração de regras de implicação em contextos de alta dimensionalidade com um grande número de objetos. Esse tipo de implicação pode ser utilizado em várias aplicações, como detecção de cópia (BRODER, 1997; SHIVAKUMAR; GARCIA-MOLINA, 1996), clustering cite c50, filtragem colaborativa (GOLDBERG et al., 1992; AGRAWAL; IMIELINSKI; SWAMI, 1993), rede social (SILVA et al., 2017) e outros.

Como objetivos específicos pode-se citar:

- Propor uma solução possibilitando selecionar o conjunto atributos que são desejados como conclusão, não sendo necessário gerar todas as implicações para todos os atributos e somente depois extrair as implicações das conclusões desejadas;

- Inclusão de uma otimização para eliminar a geração desnecessária de combinações de premissas para serem avaliadas;
- Utilização da estrutura de dados Diagrama Binário de Decisão (BDD) para representar e manipular os dados do contexto formal de maneira mais eficiente. Condição útil quando lidamos com conjuntos de dados de alta dimensionalidade;
- Elaboração de um modelo de computação paralela para ser possível processar múltiplas conclusões simultaneamente.

1.3 Organização do Trabalho

Este trabalho esta organizado da seguinte forma. A Seção II apresenta os conceitos básicos da AFC, BDD e Paralelismo. Na Seção III descreve a metodologia do trabalho. Na Seção IV descreve os trabalhos relacionados. Na Seção V descreve os experimentos e análises dos resultados. Na Seção VI são apresentadas as conclusões e possíveis trabalhos futuros.

2 REFERENCIAL TEÓRICO

A AFC tem aplicação em diferentes campos da computação como na mineração de dados (SNELTING; TIP, 2000), recuperação de informação (GODIN; CHAU, 2000), engenharia de software (ALLERI; HUCHARD, 2008), redes sociais (ARÉVALO; FALLERI, 2006), redes neurais (JOSHI; JOSHI, 2009), ontologias (YEVTUSHENKO, 2000), entre outras.

As próximas sessões irão apresentar o referencial teórico para o entendimento do uso desta teoria.

2.1 Análise Formal de Conceito

A Análise Formal de Conceito (AFC) é uma teoria para análise de dados na qual são identificadas estruturas conceituais em um conjunto de dados (GANTER; STUMME; WILLE, 2005). A AFC aplica a teoria dos reticulados conceituais para organizar hierarquicamente os dados a partir de um contexto formal composto de objetos, atributos e de suas incidências. Entretanto, os benefícios do uso desta estrutura são acompanhados de altos custos da sua construção e manipulação.

A AFC é baseada em quatro elementos fundamentais: contexto formal, conceito formal, reticulado e regras.

2.1.1 Contexto Formal

Um contexto formal é definido por uma tripla (G, M, I) em que G é um conjunto de objetos (linhas), M um conjunto de atributos (colunas) e I é definido como as incidências sendo que, $I \subseteq G \times M$ é uma relação binária entre objetos e atributos dito que $I \subseteq G \times M$ ou gIm o qual se lê como "o objeto g possui o atributo m"(GANTER; STUMME; WILLE, 2005).

Utilizando a Tabela 1 como exemplo, os objetos são as linhas da tabela: Baleia, Coruja, Humano, Tubarão e Pinguim. Os atributos são as colunas: Aquático, Terrestre, Irracional e Penas. As incidências (”X”) são as relações entre os objetos e atributos:

Baleia tem os atributos Aquático e Irracional, Coruja tem o atributo Terrestre, Irracional e Penas, assim por diante.

Tabela 1 – Contexto formal: as linhas são os atributos e as colunas são os objetos

	Aquático	**Terrestre**	**Irracional**	**Penas**
Baleia	X	.	X	.
Coruja	.	X	X	X
Humano	.	X	.	.
Tubarão	X	.	X	.
Pinguim	.	X	X	X

2.1.2 Conceitos Formais

Um conceito formal é definido como um par (A, B) em que $A \subseteq G$ é denominado de extensão e $B \subseteq M$ denominado intensão. Este par deve seguir as condições $\mathrm{A} = \mathrm{B}'$ e $\mathrm{B} = \mathrm{A}'$ (GANTER; STUMME; WILLE, 2005). Esta relação é definida pelo operador de derivação $'$:

$\mathrm{A}' = \{\mathrm{m} \in \mathrm{M} \mid \mathrm{gIm}\ \forall\ \mathrm{g} \in \mathrm{A}\}$
$\mathrm{B}' = \{\mathrm{g} \in \mathrm{G} \mid \mathrm{gIm}\ \forall\ \mathrm{m} \in \mathrm{B}\}$

Note que, se $A \subseteq G$, então A' é um conjunto de atributos comuns aos objetos de A. Aplicando novamente o operador de derivação a A' teremos A'', resultado em um conjunto de objetos que possuem, em comum, os atributos dos objetos de 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.

Normalmente, um conceito formal busca identificar uma lista de atributos (intensão) que delimita e caracteriza um determinado objeto (extensão). Utilizando a Tabela 1 como exemplo, os conceitos formais gerados são apresentados na Tabela 2:

Tabela 2 – Conceitos formais gerados da Tabela 1

Intensão	***Extensão***
{Coruja, Humano, Pinguim}	{Terrestre}
{Coruja, Pinguim}	{Terrestre, Irracional, Penas}
{}	{Aquático, Terrestre, Irracional, Penas}
{Baleia, Coruja, Tubarão, Pinguim}	{Irracional}
{Baleia, Tubarão}	{Aquático, Irracional}

2.1.3 Reticulado Conceitual

Um reticulado conceitual é descrito como um conjunto de todos os conceitos formais ordenados de forma hierárquica. Esta forma hierárquica é baseada em como os conceitos se relacionam. Formalmente, $(E_1, I_1) \leq (E_2, I_2)$, em que $E_1 \subseteq E_2$ e $I_2 \subseteq I_1$. O par (E_1, I_1) é chamado subconceito e (E_2, I_2) é chamado superconceito (WILLE, 2004), sendo que E é a extensão (atributos) e I intensão (objetos).

Um reticulado de conceitos pode ser representado por meio de um diagrama. Neste, os conceitos são os vértices e as arestas as relações em que $(E_1, I_1) \leq (E_2, I_2)$. O vértice mais baixo no diagrama representa a intensão do conceito formal contendo todos os atributos cuja a extensão contém todos os objetos (WILLE, 2004).

2.1.4 Implicações

Implicações são dependências entre elementos de um conjunto de atributos que foram obtidos de um contexto formal.

Utilizando um contexto formal (G, M, I) uma implicação seria $A \rightarrow B$ onde $A, B \subseteq M$ (A e B são chamados, respectivamente, premissa e conclusão).

Uma regra de implicação P $\rightarrow$ Q é considerada válida para o contexto (G, M, I) se, e somente se, todo objeto que possui os atributos de P também possuir os atributos de Q. Formalmente $\forall g \in G[\forall p \in P\ gIp \rightarrow \forall q \in Q\ gIq]$.

Para um exemplo do que é uma implicação, na Tabela 1 todo animal aquático é irracional. Esse tipo de relação pode ser descrito como uma implicação onde "Aquático implica irracional"($Aquatic \rightarrow Irrational$). Tabela 3 é um exemplo de regras de implicação baseadas no contexto formal apresentado na Tabela 1.

Tabela 3 – Regras de implicação extraídas da Tabela 1

$C \rightarrow D$
Irracional → Terrestre, Penas
Penas → Terrestre, Irracional
Aquático → Irracional

Uma implicação pode ser obtida através de conceitos formais (WOLFF; YAMEOGO, 2005), contextos formais (NILANDER; ZÁRATE, 2011) e reticulado conceitual (BERTET; MONJARDET, 2010). Em nosso trabalho, a extração de implicação é baseada em contextos formais.

2.1.5 Implicações Próprias

Um conjunto de implicações é definido como conjunto de implicações próprias (TAOUIL; BASTIDE, 2001) quando sua implicação no lado direito (conclusão) contém apenas um atributo e o lado esquerdo (premissa) é reduzido. Formalmente, o conjunto de IMPLICAÇÕES PRÓPRIAS $\mathcal{F}$ para (G, M, I) é definido como: $\{C \rightarrow m \in \mathcal{F} \mid C \subseteq M$ e $m \in M \setminus C$ and $\forall Z \subset C \colon Z \rightarrow m \notin \mathcal{F}\}$.

A Tabela 4 é um exemplo de implicações próprias geradas a partir do contexto formal apresentado na Tabela 1.

Tabela 4 – Conjunto de implicações próprias extraído da Tabela 1

$C \rightarrow b$
Penas → Terrestre
Aquático → Irracional
Penas → Irracional
Irracional, Terrestre → Penas

Implicações próprias são uteis em situações onde você quer encontrar o mínimo de condições que levam a um objetivo. Um exemplo desta utilização seria o conjunto mínimo de competências necessárias para alcançar posições organizacionais.

Atualmente, existem algoritmos da AFC para extrair implicações próprias como o Impec (TAOUIL; BASTIDE, 2001) e PropIm (SILVA et al., 2017). Estes dois algoritmos serão descritos abaixo.

2.1.5.1 Impec

O Impec apresentado por (TAOUIL; BASTIDE, 2001) foi utilizado como base para o PropIm (SILVA et al., 2017). Tem como objetivo encontrar a base de implicações próprias cujo lado esquerdo é mínimo e o lado direito possui apenas um atributo. O algoritmo baseia-se em conjuntos e um operador de fecho definido sobre estes conjuntos. Em outras palavras, o objetivo do algoritmo é encontrar implicações do tipo $C \rightarrow b$, em que $\subseteq M$ e $(.)''$ um operador de fecho definido sobre M.

Para obter o lado direito da implicação é utilizada a Preposição 2.1: Seja $C \subseteq M$ e (.)" um operador de fecho definido sobre M.

$$ladoD(A) = C'' - (C \cup \bigcup ladoD(B)|BsubsetA) \tag{2.1}$$

Para o lado esquerdo da implicação é utilizada a Preposição 2.2: Seja $C, B \subseteq M, C \subset B$ e (.)" um operador de fecho definido sobre M.

$$B'' = C'' \Leftrightarrow B - C \subseteq C'' - C \tag{2.2}$$

Algoritmo 1: Impec

Input : Um conjunto de atributos M, e um operador de fecho $(.)''$ definido sobre M
Output: Uma cobertura própria F

```
1  F = {∅ → ∅''};
2  foreach m ∈ M do
3  |   F' = F;
4  |   foreach A → B ∈ F' do
5  |   |   Z = (A ∪ {m})'' − (A ∪ {m});
6  |   |   if Z ≠ ∅ then
7  |   |   |   R = {X → Y ∈ F'|B ⊃ A, B − A ⊆ Z};
8  |   |   |   F' = F' − R;
9  |   |   |   foreach X → Y ∈ F', X ⊃ A ∪ {m} do
10 |   |   |   |   Z = Z − Y;
11 |   |   F = F ∪ {A ∪ {m} → Z};
12 return {A → b|b ∈ B, A → B ∈ F, B ≠ ∅, A ≠ ∅}
```

O algoritmo inicia com a implicação $F = \{\emptyset \rightarrow \emptyset''\}$. Nas linhas (2–11) é executado um ciclo para verificar as implicações de cada atributo de M. Já nas linhas (4–11) é realizada a obtenção de novas implicações baseando nas implicações já existentes. Para cada implicação a ser considerada, aplica-se as proposições nas linhas 7 e 8 e linhas 9–10 respectivamente. Por fim, adiciona-se a nova implicação encontrada (linha 11). A complexidade do $Impec$ é $O(|M||F|(|G||M| + |F||M|))$.

2.1.5.2 PropIm

O PropIm apresentado por (SILVA et al., 2017) foi baseado no Impec e utilizado como base para todos os algoritmos proposto nesta dissertação.

O algoritmo recebe como entrada um contexto formal $\beta(G, M, I)$, e retorna como resultado um lista de implicações próprias com suporte maior que 0.

Algoritmo 2: PropIm

Input : contexto formal (G,M,I)
Output: conjunto de implicações *imp* com suporte maior que 0

```
1  imp = ∅;
2  foreach m ∈ M do
3      P =m";
4      size =1;
5      Pa = ∅;
6      while size < |P| do
7          C = (P choose size);
8          Pc = getCandidateProp(C,Pa);
9          foreach P1 ⊂Pc do
10             if P1' ≠ ∅ AND P1' ⊂m' then
11                 Pa = Pa ∪ {P1};
12                 imp = imp ∪ {P1 → m};
13         size++;
14 return imp
```

A linha 1 inicializa o conjunto *imp* atribuindo vazio. O seguinte for (linhas 2-13) analisa cada atributo presente em M. Inicialmente, cada atributo m pode ser uma conclusão para um conjunto de premissas. Para cada m, é calculado as premissas $P1$.

Na linha 3, P registra todos os atributos que contém os mesmos objetos de m. A variável *size* determina o tamanho de cada premissa, pois o menor tamanho possível é 1 (uma implicação do tipo $x \rightarrow z$), é inicializado com 1 (Linha 4).

Pa armazena um conjunto de premissas auxiliares que podem gerar uma implicação usando m como conclusão (na linha 5 Pa é inicializado como vazio).

Nas linhas 6-13, o conjunto de premissas mínimas é encontrado e é limitado por |P|. Na linha 7, o conjunto C obtém todas as combinações com o tamanho *size* a partir de elementos em P. Na Linha 8, o conjunto de premissas candidatas é formado através do Algoritmo 3 que obtém todos os subconjuntos que não contêm um atributo que pertence à premissa Pa. Ele recebe, como parâmetro, os conjuntos C e Pa e retorna um conjunto D de premissas.

Algoritmo 3: getCandidateProp

```
1 Function getCandidateProp(C,Pa):
2     D = ∅;
3     foreach a ∈A|A⊂Pa do
4         foreach B ⊂C do
5             if a ∉B then
6                 D =Pc/B;
7     return D;
```

Cada premissa candidata $P1 \subset PC$ é verificada para garantir que a premissa P1 e a conclusão m resultem em uma implicação correta válida. Caso $P1' \neq \emptyset$ e $P1' \subset m'$, a premissa $P1$ é adicionada ao conjunto de premissas auxiliares Pa e também a implicação $\{\text{P1} \rightarrow \text{M}\}$ é adicionado à lista de implicações imp linha 12. A complexidade do $Impec$ é $O(|M||G|(|M|^2 + |I||M|(|M| + |M||G|)))$.

Esta sessão finaliza a descrição dos elementos necessários para a AFC. A sessão seguinte apresenta a estrutura BDD utilizada nesta dissertação.

2.2 Diagrama binário de decisão

O *Diagrama Binário de Decisão* (BDD) é uma forma para representar uma fórmula booleana canônica (BRYANT, 1986). É uma estrutura substancialmente mais compacta do que as formas tradicionais (forma conjuntiva e disjuntiva) e pode ser manipulada de forma eficiente (CLARKE JR.; GRUMBERG; PELED, 1999). Esta propriedade simplifica algumas operações, como verificar a equivalência de duas fórmulas booleanas.

O BDD formalmente pode ser descrito como um grafo acíclico dirigido contendo nós terminal e não-terminal. Cada nó não-terminal é uma variável distinta da fórmula booleana correspondente. Além disso, cada nó não-terminal contém duas arestas de saída que representam caso o valor da variável seja falso (esquerda) ou se o valor da variável for verdadeiro (direita). Existe um caminho correspondente da parte superior do BDD para um nó terminal de cada fórmula booleana existente no BDD.

Apesar do BDD ser uma estrutura substancialmente mais compacta do que as formas tradicionais, algumas limitações devem ser levadas em consideração durante a sua utilização uma vez que a complexidade espacial depende da ordem em que suas variáveis são adicionadas.

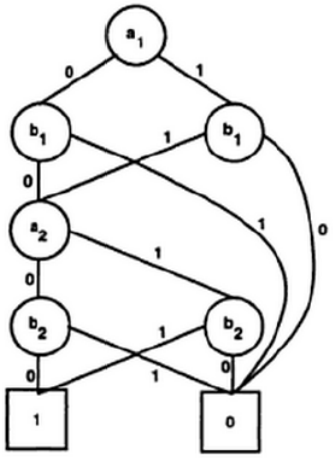


Figura 1 – Comparador de dois bits

Bryant (BRYANT, 1986) utiliza, como exemplo, um comparador de dois bits com as variáveis a1, a2, b1 e b2. Os valores são idênticos se a1 é igual a b1 e a2 é igual a b2, caso contrário, eles são diferentes. Se o BDD for construído utilizando a ordem de a1 $<$ a2 $<$ b1 $<$ b2 o resultado tem 11 nós. No entanto, se mantivéssemos as variáveis relacionadas próximas entre si, na ordem a1 $<$ b1 $<$ a2 $<$ b2 o BDD resultante teria 8 nós. No melhor caso $T(n) = 3n + 2$ a complexidade é linear $O(n)$ sendo que no pior caso, $T(n) = 3.2^n - 1$ o comportamento é exponencial $O(2^n)$.

AS Figuras 3, 4, 5 e 6 demonstram um exemplo no qual o BDD é utilizado para representar uma árvore de decisão binária descrita na Figura 2. Note que, é possível representar a mesma informação de uma forma consideravelmente mais compacta que a estrutura original.

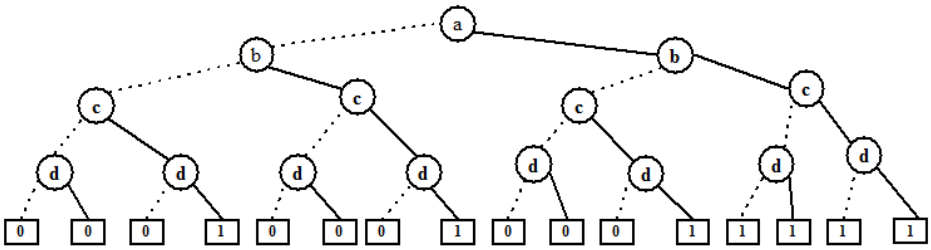


Figura 2 – BDT (Árvore Binária de Decisão)

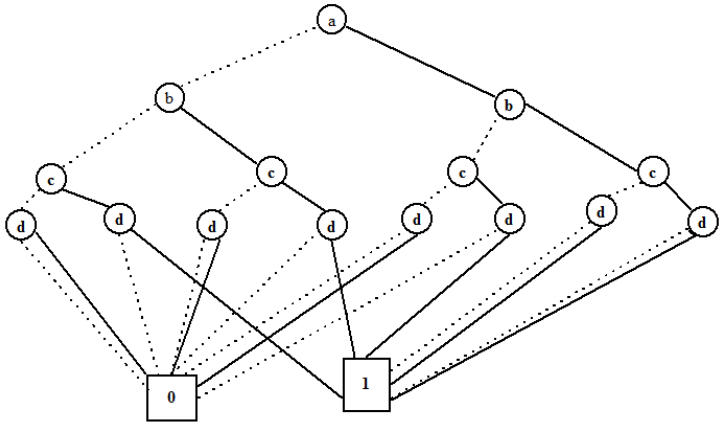


Figura 3 – Remoção de nós terminais duplicados

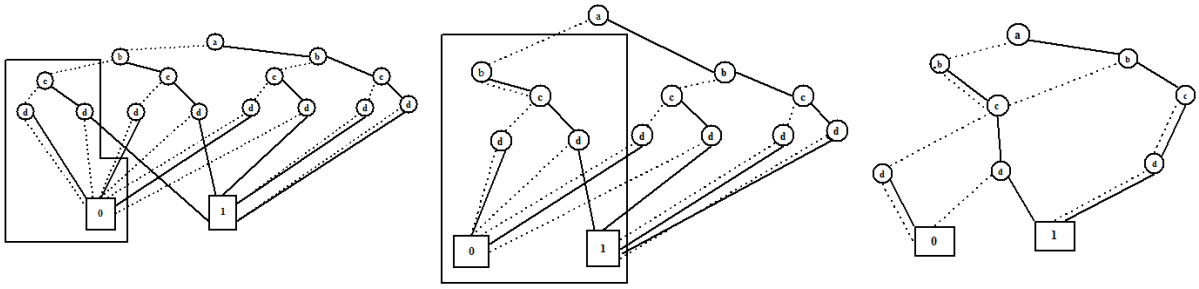


Figura 4 – Remoção de nós não terminais duplicados

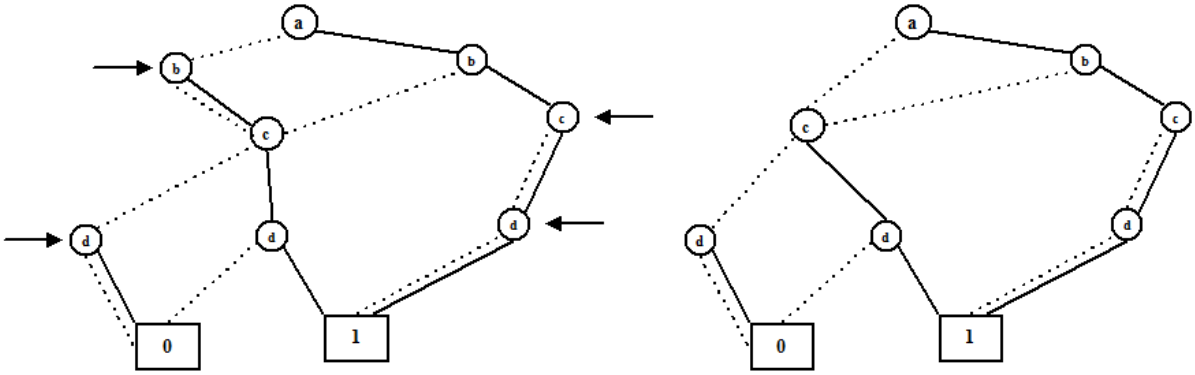


Figura 5 – Remoção de redundância

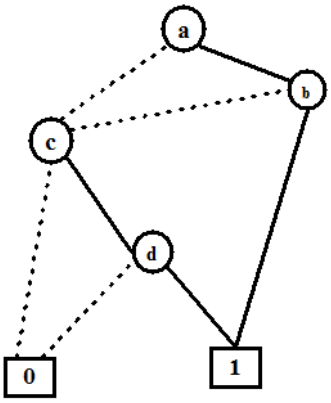


Figura 6 – BDD (Diagrama Binario de Decisão)

O BDD e a função correspondente ao contexto formal apresentado na Tabela 1 pode ser vista abaixo:

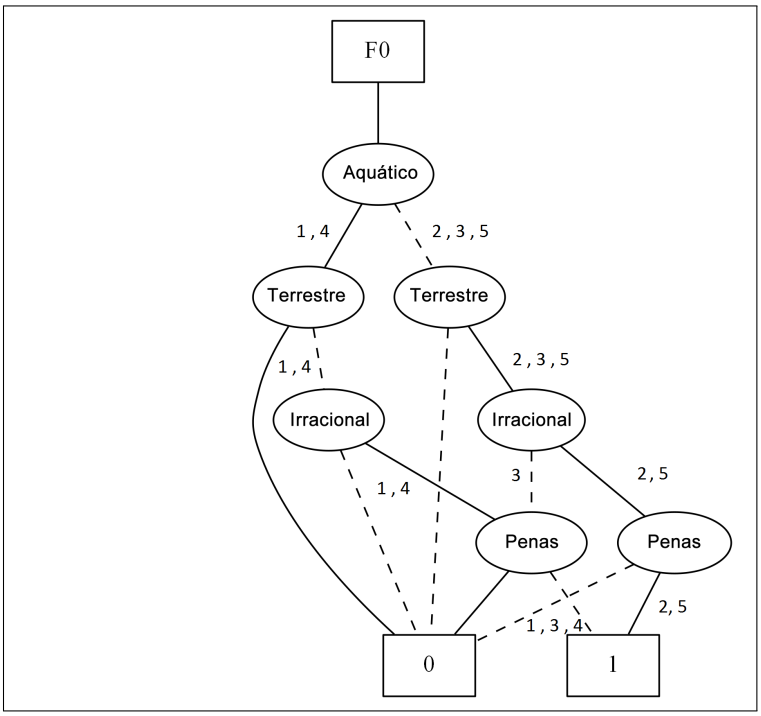


Figura 7 – BDD gerado baseado no contexto da Tabela 1.

$$f(a_1, a_2, a_3, a_4) = a_1\overline{a_2}a_3\overline{a_4} + \overline{a_1}a_2a_3a_4 + \overline{a_1}a_2\overline{a_3a_4} \tag{2.3}$$

Esta função é utilizada para a geração do BDD que representa o contexto formal da Tabela 1. Para uma melhor visualização de cada objeto no BDD, os nomes dos objetos foram alterados para números e os atributos foram alterados para letras:

- Baleia(1), Coruja(2), Humano(3), Tubarão(4), Pinguim(5);

- Aquático(a_1), Terrestre(a_2), Irracional(a_3), Penas(a_4).

Em nossa abordagem, utilizamos o BDD para criar uma representação mais compacta do contexto formal. A equação (2.3) representa a fórmula booleana correspondente a Tabela 1. O atributo $\overline{a_1}$ significa que neste termo da função o atributo é falso indicando a ausência desta característica. Repare que o termo $a_1\overline{a_2}a_3\overline{a_4}$ representa os objetos Baleia e Tubarão, $\overline{a_1}a_2a_3a_4$ representa os objetos Pinguim e Coruja, e $\overline{a_1}a_2\overline{a_3a_4}$ representa o objeto humano.

Mantendo a compatibilidade com a linguagem em que o algoritmo *PropIm* foi escrito (JAVA), utilizou-se uma biblioteca que fornece suporte para o BDD em JAVA. A biblioteca escolhida JavaBDD (JAVABDD, 2017), fornece *Interface Nativa do Java* (JNI) para bibliotecas relevantes de BDD que são a CUDD, BuDDy e CAL (RIMSA; ZÁRATE; SONG, 2009).

2.3 Paralelismo

Atualmente na computação, existem inúmeros problemas que necessitam de mais poder de processamento que uma única *Unidade Central de Processamento* (CPU) pode oferecer. Um problema de alta complexidade pode demorar um longo período de tempo ou tempo indefinido para ser resolvido utilizando um único processador porém, este problema pode ser dividido em vários subproblemas e ser processado simultaneamente utilizando computação paralela (KHAN; ALI, 2012). Uma possível solução para este tipo de problema é a utilização da computação paralela na qual são conectados múltiplos processadores através de meios de computação de alta velocidade. Para a elaboração de um algoritmo em paralelo algumas técnicas podem ser utilizadas para obter uma melhor utilização do paralelismo. Esta técnicas são: modelo de computação paralela, projeto do algoritmo em paralelo, divisão adequada do problema e programação paralela (KHAN; ALI, 2012).

No modelo de computação paralela pode ser utilizada a arquitetura multiprocessador sendo que, mais de uma CPU é incorporada em um único computador. O compilador é responsável por paralelizar o código automaticamente. Esse tipo de arquitetura não é tão eficiente, mas é melhor que um computador com um única CPU. Na arquitetura de memória compartilhada, vários processadores estão conectados a uma memória central comum. Uma vez que todos os processadores estão compartilhando um único espaço de endereço, o compartilhamento de dados é rápido, mas os processos podem concorrer ao mesmo tempo pelos mesmos dados. Semáforos e bloqueios são usados para evitar este

tipo de concorrência e alteração dos dados. Existe uma falta de escalabilidade entre processadores e memória, o que torna inviável a utilização de muitos processadores já que a memória é limitada. Este problema surge principalmente devido a velocidade do barramento (MICHAEL, 1995; JELICA; MILO; VELJKO, 1996). Na arquitetura multinúcleo é possível utilizar uma única CPU com dois ou mais núcleos e processar tarefas simultâneas.

A Figura 8 demonstra o processamento de tarefas utilizando um processador *singlecore* e um processador *multicore*. Enquanto o *singlecore* precisa aguardar uma tarefa terminar para executar a próxima, o *multicore* pode iniciar varias tarefas simultaneamente.

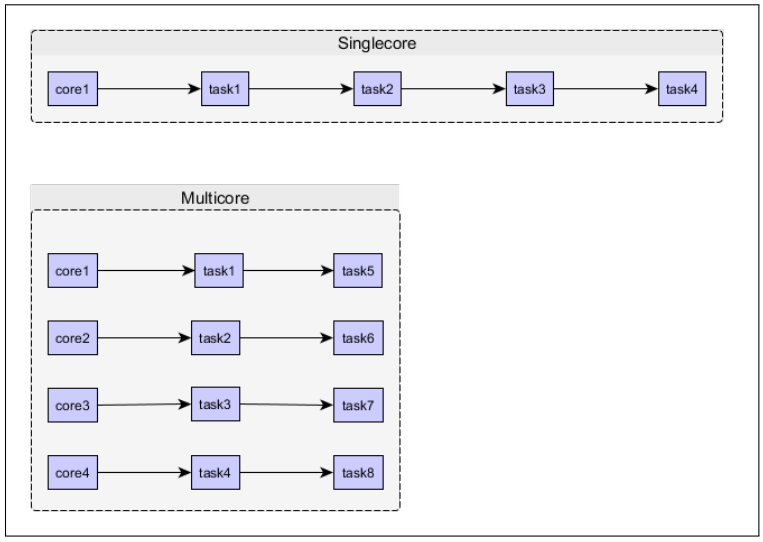


Figura 8 – Modelo de computação Singlecore e Multicore

Para projetar um algoritmo em paralelo podemos utilizar diversas técnicas. As mais comuns utilizadas são a repartição, divisão e conquista e *pipelining*. Utilizando o particionamento, um problema é divido em sub-problemas que serão resolvidos simultaneamente não ocasionando sobreposição de sub-problemas. Na divisão e conquista o problema também é divido em sub-problemas porém, os sub-problemas serão resolvidos de forma recursiva e o resultado dos sub-problemas são combinados no final. Já no *pipelining*, o problema é dividido em segmentos e a saída de um segmento é a entrada para outro segmento.

Uma divisão adequada de um problema de alta complexidade em vários sub-problemas pode facilitar a implementação efetiva de um algoritmo paralelo. Existem duas técnicas principais para a decomposição de um problema. Primeiro é a decomposição do domínio e a segunda é a decomposição funcional.

Utilizando a decomposição do domínio os dados do problema são divididos em partes de tamanho igual. O algoritmo é projetado e divido de forma a operar em cada tarefa independente. Tarefas distintas são atribuídas para operar com dados distintos. As tarefas de decomposição de domínio começam simultaneamente.

A decomposição funcional divide o algoritmo em tarefas independentes que podem ser processadas simultaneamente. Se os dados necessários para as tarefas também são independentes, a divisão é perfeita, caso contrário, a comunicação será considerada para evitar a repetição de dados. Todas as tarefas começam simultaneamente, mas algumas terão que esperar até que os dados sejam obtidos.

Na programação paralela, duas metodologias são amplamente utilizadas. Elas são compiladores de auto-paralelização e software baseado em biblioteca. Na primeiro é o compilador que encontra o paralelismo durante a compilação do código-fonte. Esta abordagem destina-se principalmente a paralelizar os *loops*. O segundo é o programador que direciona as diretrizes do compilador para tornar o código paralelo. A maioria das linguagens de programação incorpora bibliotecas para suportar paralelismo.

Neste trabalho para o modelo de computação paralela foi utilizado o *multinúcleo* com o intuito de ser possível utilizar uma única CPU com N núcleos e processar N conclusões simultaneamente. A solução foi elaborada visando processar uma conclusão independente de outra. Utilizou-se decomposição do domínio visando os dados do problema e não o algoritmo. A base da solução é o processamento de conclusões independentes e, com a decomposição do domínio conclusões distintas são atribuídas para operar com dados distintos. Para ser possível determinar quais partes do algoritmo seriam paralelizadas foi utilizado o modelo de programação de software baseado em biblioteca.

Para o modelo de computação paralela do Algoritmo $PImplicPBDD$ foi utilizado o *multinúcleo* com o intuito de ser possível utilizar uma única CPU com N núcleos e processar N conclusões simultaneamente. Este modelo de computação paralela facilita a sua utilização porque, atualmente praticamente todos os processadores contém mais do que um núcleo.

A elaboração do Algoritmo $PImplicPBDD$ foi realizada visando processar uma conclusão independente de outra. Devido a este processamento de conclusões independentes, o particionamento como desenho de computação paralela foi adotado.

A decomposição do domínio utilizada no algoritmo foi projetado visando os dados do problema e não o algoritmo. A base do algoritmo $PImplicPBDD$ é o processamento de conclusões independentes, com a decomposição do domínio conclusões distintas são atribuídas para operar com dados distintos.

Para ser possível determinar quais partes do algoritmo seriam paralelizadas foi utilizado o modelo de programação de software baseado em biblioteca. Este modelo permite definir quais pontos serão paralelizados.

3 TRABALHOS RELACIONADOS

Atualmente existem diversos algoritmos da AFC para extrair implicações próprias. No entanto, de acordo com nosso conhecimento, não existem algoritmos que extraem implicações próprias em contextos de alta dimensionalidade. Foram encontrados estudos que utilizaram o BDD na AFC, mas com foco na extração de conceitos formais e não em implicações.

Em (TAOUIL; BASTIDE, 2001) foi proposto o *Impec*, algoritmo para a extração de implicações próprias baseando-se no conjunto de atributos do contexto formal. No entanto, a estratégia utilizada pelo algoritmo não permite que sejam definidos atributos específicos para uma conclusão, sendo necessário um pós-processamento para obter somente as conclusões desejadas. A solução proposta neste artigo ($ImplicP$) permite selecionar os atributos desejados para uma conclusão, evitando esforço desnecessário de pós-processamento. Similar a (TAOUIL; BASTIDE, 2001), (SILVA et al., 2017) propôs o algoritmo *PropIm* para extrair implicações próprias com base no algoritmo Impec entretanto, o *PropIm* extrai as próprias implicações próprias com suporte maior que zero onde, todas as implicações encontradas têm pelo menos uma ocorrência no contexto. O algoritmo foi utilizado para identificar as relações entre as habilidades profissionais do perfil de usuários do LinkedIn através de implicações próprias. Os resultados apresentaram os conjuntos mínimos de habilidades que seriam necessários para atingir determinados cargos. Diferente dos trabalhos anteriores, a solução proposta neste artigo tem como objetivo extrair implicações próprias em contextos de alta dimensionalidade onde os algoritmos propostos por (TAOUIL; BASTIDE, 2001), (TAOUIL; BASTIDE, 2001) não tem um desempenho satisfatório.

Em (YEVTUSHENKO, 2002) foi proposto um algoritmo para extrair conceitos formais utilizando BDD. A utilização do BDD foi para representar a lista de conceitos encontrados. Possivelmente é o primeiro trabalho que utiliza BDD no processo de extração de conceitos formais. Entretanto foi utilizou contextos com 900 objetos e 50 atributos demonstrando ser eficiente apenas para contextos densos. Diferente de (YEVTUSHENKO, 2002), (NETO; ZÁRATE; SONG, 2018) propuseram utilizar o BDD em dois algoritmos da AFC (NextClosure e Inclose2) que extraem conceitos formais para ser possível manipular contextos de alta dimensionalidade. O BDD foi utilizado na representação do contexto

formal reduzindo o espaço e simplificando as operações de manipulação da informação. Seus resultados demonstraram que essa abordagem possibilita a manipulação de contextos de alta dimensionalidade utilizando quantidades de objetos que antes eram inviáveis para os algoritmos originais. Similar a (NETO; ZÁRATE; SONG, 2018), (SALLEB; MAAZOUZI; VRAIN, 2002) demonstrou que é possível trabalhar com maiores volumes de dados utilizando BDD fora do contexto da AFC. Em seu trabalho foi utilizado o BDD na mineração de transações em grandes bases de dados transacionais. O conjunto de dados foi representado como um BDD sendo que, se existi-se uma transação era retornado verdadeiro e falso caso contrário. Em (RIMSA; ZÁRATE; SONG, 2009) utilizou-se o BDD para a extração de conceitos formais, porém o foco do seu artigo foi de utilizar diferentes bibliotecas de BDD para verificar qual a melhor biblioteca que poderia ser utilizada em AFC. (RIMSA; ZÁRATE; SONG, 2009) utilizou métodos de força bruta para obter o conjunto de intenções, e consequentemente o BDD que representavam as extensões. Em todos os testes, os algoritmos com BDD foram mais eficientes se comparados ao algoritmos originais. Similar as trabalhos de (YEVTUSHENKO, 2002), (NETO; ZÁRATE; SONG, 2018), (SALLEB; MAAZOUZI; VRAIN, 2002) e (RIMSA; ZÁRATE; SONG, 2009) este artigo propõe a utilização do BDD para ser possível processar contextos de alta dimensionalidade porém, o BDD é utilizado para representar e manipular o contexto formal para extrair implicações.

Em (MORAES et al., 2016) foi proposta uma solução onde é possível atuar em um problema já existente na AFC que é processar um contexto de alta dimensionalidade com um grande número de objetos. Na solução proposta foi paralelizado o algoritmo NextClosure o qual, gera o conjunto mínimo de implicações de um contexto formal. O objetivo da solução foi de reduzir o tempo proibitivo em cenários onde o contexto é denso e com alta dimensionalidade. Em seus experimentos, foi constatado uma melhora de aproximadamente 75% no tempo de execução em contextos de alta dimensionalidade e densos. O trabalho de (MORAES et al., 2016) é similar a uma das soluções proposta neste artigo, sendo que foi criado um algoritmo paralelizado ($PImplicPBDD$) baseado no ($ImplicPBDD$), porém, o nosso objetivo era de trabalhar com regras de implicação próprias e não com conjunto mínimo de implicações. Com a paralelização do algoritmo obtivemos ganhos significativos se comparado ao $ImplicPBDD$.

4 METODOLOGIA

Neste capítulo, será descrita a metodologia utilizada para este trabalho. Para cada solução proposta, foi criado um algoritmo e realizados testes para analisar o desempenho deste novo algoritmo. A primeira solução proposta, o algoritmo *ImplicP*, utiliza uma otimização para evitar a geração desnecessária de combinações de premissas para serem avaliadas. Na segunda solução, O algoritmo *ImplicPBDD*, além de utilizar a otimização do *ImplicP*, inclui uma a estrutura de dados (BDD) para ser possível representar e manipular o contexto formal de forma mais eficiente. Por fim, o algoritmo *PImplicPBDD* contém as soluções dos algoritmos anteriores incluindo um modelo de computação paralela para ser possível processar múltiplas conclusões simultaneamente.

Cada solução proposta foi comparada ao algoritmo original *PropIm* e entre cada nova solução baseando-se nas bases sintéticas e em uma base real. Foram utilizados contextos sintéticos gerados aleatoriamente variando a quantidade de atributos, objetos e densidades e também um teste com uma base real a partir de uma amostragem extraída da rede social Linkedin. A ferramenta SCGAz (RIMSA; ZÁRATE; SONG, 2009) foi utilizada para gerar contextos aleatórios dentro dos limites das densidades e tamanhos controlados.

4.1 Algoritmo ImplicP

O Algoritmo *ImplicP* utiliza uma otimização (*Algoritmo getC*) para eliminar a geração desnecessária de combinações de premissas para serem avaliadas.

Para um melhor entendimento da geração de premissas e utilização da otimização do Algoritmo *ImplicP*, a Tabela 5 contém um contexto formal que será utilizado de exemplo para esta demonstração.

Tabela 5 – Contexto formal simples

	a	**b**	**c**	**d**	**c1**
obj1	X	.	.	.	X
obj2	.	.	.	X	.
obj3	.	.	X	.	.
obj4	.	X	.	.	.
obj5	.	X	X	X	X

Atributos: {a, b, c, d}
Conclusão: {c1}
Implicações:
$\{a\} \rightarrow \{c1\}$
$\{bcd\} \rightarrow \{c1\}$

Na primeira iteração do Algoritmo 4 (linhas 8-16) utilizando *c1* como conclusão, serão gerados todos os subconjuntos de premissas de tamanho 1 que não contém algum dos atributos que já existe em alguma premissa válida de uma implicação já criada. O Algoritmo 7 é responsável por realizar esta validação e geração dos subconjuntos de premissas. O funcionamento do algoritmo é descrito na Tabela 6. Após a geração das premissas pelo Algoritmo 7, estas premissas são passadas como parâmetro para o Algoritmo 6 onde este irá selecionar somente as premissas candidatas conforma a Tabela 7. Como resultado, é encontrada a implicação $\{a\} \rightarrow \{c1\}$

Iteração 1

Tabela 6 – Premissas geradas na primeira iteração

Combinação	Tamanho
{a, b, c, d}	1
Total: 4 combinações	

Tabela 7 – Premissas candidatas geradas na primeira iteração

Premissas	Tamanho
{a, b, c, d}	1
Total: 4 premissas	

Na segunda iteração do Algoritmo 4 (linhas 8-16) utilizando *c1* como conclusão, serão gerados todos os subconjuntos de premissas de tamanho 1 e 2 que não contém algum dos atributos que já existe em alguma premissa válida de uma implicação já criada. O funcionamento do algoritmo é descrito na Tabela 8. Após a geração das premissas, estas, premissas são passadas como parâmetro para o Algoritmo 6 onde este irá selecionar somente as premissas candidatas conforma a Tabela 9. Como resultado, serão retiradas

as premissas {a}, {ab, ac, ad} já que, existe uma implicação que utiliza o atributo ***a***, ***{a}*** $\rightarrow$ ***{c1}***

Iteração 2

Tabela 8 – Premissas geradas na segunda iteração

Combinação	Tamanho
{b, c, d}	1
{bc, bd, cd}	2
Total: 10 combinações	

Tabela 9 – Premissas candidatas geradas na segunda iteração

Premissas	Tamanho
{b, c, d}	1
{bc, bd, cd}	2
Total: 6 premissas	

Na terceira iteração do Algoritmo 4 (linhas 8-16) utilizando *c1* como conclusão, serão gerados todos os subconjuntos de premissas de tamanho 1,2 e 3 que não contém algum dos atributos que já existe em alguma premissa válida de uma implicação já criada. O funcionamento do algoritmo é descrito na Tabela 10. Após a geração das premissas, estas são passadas como parâmetro para o Algoritmo 6 onde este irá selecionar somente as premissas candidatas conforma a Tabela 11. Como resultado, serão retiradas as premissas {a}, {ab, ac, ad} e {abc, abd, acd} já que, existe uma implicação que utiliza o atributo ***a***, ***{a}*** $\rightarrow$ ***{c1}***. Foi encontrada uma nova implicação ***{bcd}*** $\rightarrow$ ***{c1}***. Como todos os atributos para premissas foram utilizados o algoritmo passa para a próxima conclusão.

Iteração 3

Tabela 10 – Premissas geradas na terceira iteração

Combinação	Tamanho
{b, c, d}	1
{bc, bd, cd}	2
{bcd}	3
Total: 7 combinações	

Tabela 11 – Premissas candidatas geradas na terceira iteração

Premissas	Tamanho
{b, c, d}	1
{bc, bd, cd}	2
{bcd}	3
Total: 7 premissas	

É importante ressaltar que com a inclusão do Algoritmo 7 no Algoritmo 4 evita-se a geração desnecessária de combinações de premissas obtendo somente as premissas candidatas. Na maioria dos casos, serão encontradas implicações antes da combinação de todos os tamanhos de premissas, permitindo a otimização de poda do $ImplicP$ atuar, tornando o desempenho do algoritmo melhor. No exemplo da Tabela 5, o ImplicP não gerou 43% das combinações que foram descartadas pelo $PropIm$.

O algoritmo recebe como entrada um contexto formal (G, M, I), e retorna como resultado um conjunto de implicações próprias Imp com suporte maior que 0.

Algoritmo 4: ImplicP

Input : número conclusões, contexto formal (G,M,I)
Output: conjunto de implicações Imp com suporte maior que 0

```
1  Imp = ∅;
2  Conclu = getC(número conclusões, M);
3  foreach m ∈ M\Conclu do
4      P = m"\Conclu;
5      Size =1;
6      Pa = ∅;
7      ResultImpAtr = ∅;
8      while Size < |P| do
9          C = getP(P, Size, 0, ResultImpAtr, Pa);
10         Pc = getCP(C,Pa);
11         foreach P1 ⊂Pc do
12             if P1' ≠ ∅ AND P1' ⊂ m' then
13                 Pa = Pa ∪ {P1};
14                 Imp = Imp ∪ {P1 → m};
15         ResultImpAtr = Imp;
16         Size++;
17 return Imp
```

A linha 1 inicializa o conjunto Imp atribuindo vazio. Na linha 2 o conjunto $Conclu$ recebe o resultado do algoritmo $getC$ onde são obtidos os atributos que serão utilizados como conclusão.

O seguinte for (linhas 3-18) analisa cada atributo m presente no conjunto M. Inicialmente, cada atributo m pode ser uma conclusão para um conjunto de premissas.

Na linha 4, P registra todos os atributos que contém os mesmos objetos de m, excluindo os atributos considerados como conclusão. O $Size$ determina o tamanho de cada premissa, pois o menor tamanho possível é 1 (uma implicação do tipo $x \rightarrow z$), é inicializado com 1 (Linha 5).

O conjunto Pa armazena um conjunto de atributos que representam as premissas auxiliares que podem gerar uma implicação usando m como conclusão (na linha 6 o conjunto de atributos Pa é inicializado como vazio).

Na linha 7 $resultImpAtr$ armazena o conjunto de todas as implicações que foram geradas para a conclusão atual.

Nas linhas 8-17, o conjunto de premissas mínimas é encontrado e é limitado por |P|. Na linha 9 o Algoritmo $getP$ obtém todas as combinações de tamanho $Size$ a partir de elementos em P. Ele recebe, como parâmetro, os elementos em P, $Size$, posição de início da combinação e a lista na qual vai ser armazenado a lista de resultado das combinações.

Na Linha 10, o conjunto de premissas candidatas é formado através do algoritmo $getCP$.

Foi criado o Algoritmo $getP$ que utiliza a otimização para reduzir o número total de combinações de premissas. Este algoritmo foi utilizado na linha 9 do Algoritmo $ImplicP$.

Cada premissa candidata $P1 \subset PC$ é verificada para garantir que a premissa P1 e a conclusão m resultem em uma implicação correta válida. Caso $P1' \neq \emptyset$ e $P1' \subset m'$, a premissa $P1$ é adicionada ao conjunto de premissas auxiliares Pa e também a implicação $\{\text{P1} \rightarrow \text{M}\}$ é adicionado à lista de implicações Imp linha 14.

Algoritmo 5: getC

```
1  Function getC(NunC, M):
2      for i = 1 to i <= NunC do
3          C = C ∪ M[i];
4      return C;
```

O Algoritmo $getC$ recebe como parâmetro o numero total de conclusões e a lista de atributos Pa do contexto e retorna a lista de atributos que serão utilizados como conclusões. Os atributos selecionados como conclusão não serão avaliados como possíveis premissas de uma conclusão.

Algoritmo 6: getCP

```
1  Function getCP(C,Pa):
2      D = ∅;
3      foreach B ⊂C do
4          if Pa = ∅ then
5              D = D ⊂ B;
6          else  foreach a ∈A|A⊂Pa do
7              if a ∉B then
8                  D = D ⊂ B;
9          ;
10     return D;
```

O Algoritmo $getCP$ obtém todos os subconjuntos que não contêm um atributo que pertence à premissa Pa. Ele recebe, como parâmetro, os conjuntos C e Pa e retorna um conjunto D de premissas.

Algoritmo 7: getP

```
1  Function getP(Atr, Len, StartPosition, Result, Pa):
2      if LEN = 0 then
3          D = D ∪ {result};
4          return;
5      for i = StartPosition to i <= |Atr| − Len do
6          Result[|Result| − Len] = Atr[i];
7          if Result ⊂ Pa then
8              continue;
9          getP(Atr, Len − 1, i + 1, Result);
10     return D;
```

O Algoritmo *getP* obtém as combinações de tamanho *Size* a partir de elementos em *P*. Nas linhas 2-4 é verificada a condição de parada para a recursividade. Se a premissa já foi criada (Len = 0), esta é incluída na lista de resultados *D* e a recursividade é finalizada. O laço de repetição nas linhas 5-9 monta a próxima premissa da combinação e verifica se esta premissa já não foi utilizada nas implicações anteriores. Se esta premissa já foi utilizada em alguma implicação anterior a sequência de geração utilizando esta premissa como base é descartada. Como saída é gerada a lista de premissas com o tamanho da variável *Size*.

Note que, utilizando a solução proposta no Algoritmo *getP* é possível gerar uma quantidade menor do total de combinações para premissas de um determinado tamanho da variável *Size*. No momento da geração de uma determinada sequência de premissas, se um atributo da premissa que será gerada já estiver sido utilizada em uma implicação para a conclusão atual, a sequência de geração é descartada melhorando o desempenho final do algoritmo. Se não existir nenhuma implicação para a conclusão atual utilizando o atributo da premissa, toda a sequencia será gerada.

4.1.1 Ordem de complexidade na geração de premissas

Na ordem da complexidade do ImplicP, N é definido como o tamanho da premissa que será gerada. M é a quantidade de atributos que serão utilizados como conclusão.

Melhor caso: $O(N * |M|)$

Já no pior caso, seria necessário gerar todas as combinações de todos os tamanhos de atributos de premissa para todos os atributos de conclusão.

Pior caso: $O(M(\sum_{i=1}^{n} C_{n,i}))$

Ao analisar a complexidade de tempo do algoritmo $ImplicP$, na linha 2 é no pior caso |M|, o custo relacionado ao primeiro laço (linhas 3 a 16) é no pior caso |M|, quando todos os atributos são considerados como possíveis conclusões. Na linha 4, P recebe um conjunto de atributos comuns ao atributo selecionado para conclusão, nesse caso o atributo m e derivado duas vezes, logo o custo no pior caso é O(|M||G|). No laço (linhas 8 a 16) são computadas as premissas da implicação cuja conclusão é m com o custo no pior caso, O(|M-1|) que seria todos os atributos do contexto formal menos o atributo selecionado para conclusão. Na linha 9 é realizada uma combinação, para gerar premissas candidatas de acordo com a quantidade de atributos permitida para cada premissa, assim quando Size = 1 são geradas premissas candidatas em que cada uma só pode conter 1 atributo , portanto o custo relacionado a esta combinação é O($\sum_{i=Size}^{n} C_{p,i}$). Na linha 10 é chamada a heurística que realiza o corte no conjunto de premissas candidatas, cujo custo no pior dos casos é O(|I||M|). O ciclo das linhas 11 a 15 é executado, no pior caso |M-1| vezes. Na linha 12 são executadas duas derivações, cujo custo é |M||G|. De maneira geral, o tempo necessário para a computação das implicações de todos os atributos é O(|M|(|M||G|(|M-1|($\sum_{i=Size}^{p} C_{p,i}$) + (|I||M|)(|M|+|M||G|)))).

4.2 Algoritmo ImplicPBDD

O Algoritmo $ImplicPBDD$ utiliza uma otimização para a redução do total de combinações de premissas necessárias para a obtenção das regras de implicação e também o diagrama binário de decisão (BDD) para simplificar a representação de um contexto formal e facilitar a manipulação dos objetos. Para a criação do BDD representando o contexto formal, foi utilizado o Algoritmo $generateBDDFormalContext$ proposto por (RIMSA; ZÁRATE; SONG, 2009), que recebe como entrada um arquivo no formato Burmeister (BURMEISTER, 2003), percorre todos os atributos do objeto e, se o objeto contém o atributo (incidência), a variável BDD indicada pelo índice "i" é incluída no BDD temporário do objeto com a indicação verdadeira. Caso contrário, a variável é incluída com a indicação falsa. Por fim, o algoritmo adiciona o BDD que representa o objeto atual (BDDTemp) ao BDD que representa o contexto formal (BDDContexto). Assim, realizando operações de AND para inserir os atributos e OR para os objetos, o contexto utilizando BDD é construído.

Algoritmo 8: Formal context using BDD

Input : arquivo no formato cxt
Output: contexto formal utilizando BDD

```
1 Arquivo =Arquivo.LerLinha(); while Arquivo.Fim() do
2     BDDTemp =BDDNovo;
3     for i ← 0 to Numerodeatributos do
4         if Arquivo[i] ==’X’ then
5             BDDTemp& =noVerdadeiro(i);
6         else BDDTemp& =noFalso(i) ;
7     BDDContexto||| =BDDTemp;
8 return BDDContexto;
```

O algoritmo recebe como entrada um contexto formal (G, M, I), e sua saída é um conjunto de implicações próprias com suporte maior que 0.

Algoritmo 9: ImplicPBDD

Input : contexto formal β(G,M,I)
Output: conjunto de implicações Imp

```
1  Imp = ∅;
2  foreach m ∈ M do
3      BddC = primeAtrSetBDD(m);
4      P =m";
5      Size =1;
6      Pa = ∅;
7      ResultImpAtr = ∅;
8      while Size < |P| do
9          C = getP(P, Size, 0, ResultImpAtr, Pa);
10         Pc = getCPBDD(C,Pa);
11         foreach P1 ⊂ Pc do
12             BddP = primeAtrSetBDD(P1);
13             if BddC ≠ 0 AND BddP ≠ 0 AND BddC = BddP then
14                 Pa = Pa ∪ {P1};
15                 Imp = Imp ∪ {P1 → m};
16         Size++;
17 return Imp
```

A linha 1 inicializa o conjunto Imp vazio. O seguinte for (linhas 2-16) analisa cada atributo do conjunto M. Inicialmente, cada atributo m pode ser uma conclusão para um conjunto de premissas. Para cada m, ele calcula as premissas $P1$.

O $BddC$ através do algoritmo $primeAtrSetPBDD$ recebe e armazena o BDD contendo todos os objetos do atributo m (linha 3). Este BDD será usado na verificação da igualdade entre a premissa BDD ($P1$) e a conclusão BDD (m).

P registra todos os atributos que contêm os mesmos objetos de m (linha 4).

O $Size$ determina o tamanho de cada premissa, pois o menor tamanho possível é 1 (uma implicação do tipo $b \rightarrow a$), o valor é inicializado com 1 (Linha 5).

Pa armazena um conjunto de premissas auxiliares que podem gerar uma implicação usando m como conclusão (na linha 6 Pa é inicializado como vazio). Nas linhas 8-16, o conjunto de premissas mínimas é encontrado e é limitado por |P|. Na linha 9, o conjunto C obtém todas as combinações de tamanho $size$ de elementos em P. Na Linha 10, o conjunto de premissas candidatas é criado através do algoritmo getCPBDD.

Para cada premissa candidata $P1 \subset PC$ o $BddP$ armazena o BDD que contém todos os objetos da premissa. Na linha 12, é feita uma verificação entre $BddC$ e $BddP$ para garantir que premissa P1 e conclusão m resultem em uma implicação válida. Se $BddC$ $= BddP$ a implicação é válida. Considerando que a implicação é válida, a premissa $p1$ é adicionada à hipótese de premissas Pa e também a implicação $\{\text{P1} \rightarrow \text{M}\}$ é adicionada à lista de implicações Imp linha 15.

Algoritmo 10: primeAtrSetPBDD

```
1  Function primeAtrSetPBDD(ListAttributes):
2      BddNewExt = BddCxt;
3      foreach it ∈ ListAttributes do
4          BddNewExt& =BddNewExt.And(it);
5      return BddNewExt;
```

O Algoritmo $primeAtrSetPBDD$ é responsável por obter o BDD que contém todos os objetos da lista dos atributos passados como parâmetros.

A função calcula a conjunção entre o BDD que representa todo o contexto formal e o BDD de cada atributo da lista de atributos. Como um resultado é retornado um BDD contém apenas os objetos que contém todos os atributos.

Algoritmo 11: getPPBDD

```
1  Function getPPBDD(Atr, Len, StartPosition, Result, Pa):
2      if Len = 0 then
3          D = D ∪ {result};
4          return;
5      for i = startPosition to i <= |Atr| − Len do
6          Result[|Result| − Len] = Atr[i];
7          if Result ⊂ Pa then
8              continue;
9          getP(Atr, Len − 1, i + 1, Result);
10     return D;
```

O Algoritmo *getPPBDD* obtém as combinações de tamanho *Size* a partir de elementos em P. Nas linhas 2-4 é verificada a condição de parada. Se a premissa já foi criada (Len = 0), esta é incluída na lista de resultados D. O laço de repetição nas linhas 5-9 monta a próxima premissa da combinação e verifica se esta premissa já foi utilizada nas implicações anteriores. Se esta premissa foi utilizada em uma implicação anterior, a sequencia de geração utilizando esta premissa é descartada. Como saída é gerada a lista de premissas com o tamanho de *Size*.

Algoritmo 12: getCPBDD

```
1 Function getCPBDD(C,PA):
2     D = ∅;
3     foreach a ∈A|A⊂PA do
4         foreach B ⊂C do
5             if a ∉B then
6                 D =Pc/B;
7     return D;
```

O Algoritmo *getCPBDD* obtém todos os subconjuntos que não contém um atributo que pertence à premissa Pa. Ele recebe como parâmetro os conjuntos C e Pa e retorna um conjunto D de premissas.

4.2.1 Processo interno do BDD para obter uma implicação

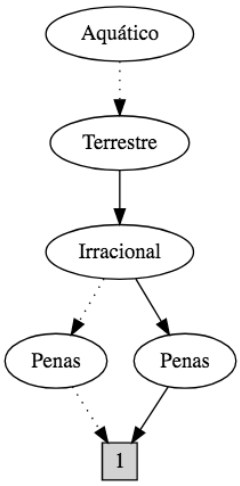


Figura 9 – Conclusão - Terrestre

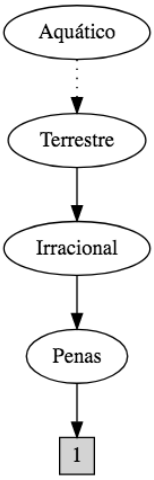


Figura 10 – Premissa - Penas

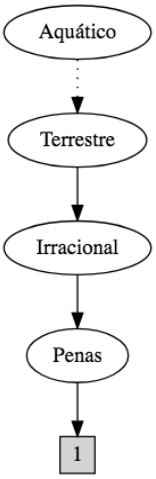


Figura 11 – Interseção entre Terrestre e Penas

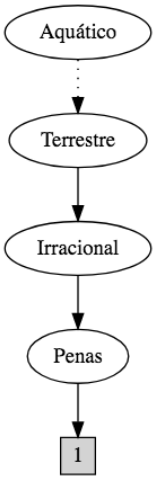


Figura 12 – Implicação: Penas igual a Interseção entre Terrestre e Penas

A Figura 9 exibe o BDD que foi gerado pela conclusão Terrestre. A Figura 10

exibi o BDD que foi gerado pela premissa Penas. A Figura 12 exibe a interseção entre o BDD conclusão Terrestre e premissa Penas. Caso a interseção entre o BDD Terrestre e premissa Penas é igual a premissa Penas uma implicação validá é criada. Observe que, a figura 12 é idêntica a Figura 10. Isto significa que todos os objetos que compartilham a premissa Penas também compartilham a conclusão Terrestre.

Ao analisar a complexidade de tempo do algoritmo $ImplicPBDD$, o custo relacionado ao primeiro laço (linhas 2 a 16) é no pior caso |M|, quando todos os atributos são considerados como possíveis conclusões. Na linha 4, P recebe um conjunto de atributos comuns ao atributo selecionado para conclusão, nesse caso o atributo m e derivado duas vezes, logo o custo no pior caso é O(|M||G|). No laço (linhas 8 a 16) são computadas as premissas da implicação cuja conclusão é m com o custo no pior caso, O(|M-1|) que seria todos os atributos do contexto formal menos o atributo selecionado para conclusão. Na linha 9 é realizada uma combinação, para gerar premissas candidatas de acordo com a quantidade de atributos permitida para cada premissa, assim quando Size = 1 são geradas premissas candidatas em que cada uma só pode conter 1 atributo , portanto o custo relacionado a esta combinação é O($\sum_{i=Size}^{p} C_{p,i}$). Na linha 10 é chamada a heurística que realiza o corte no conjunto de premissas candidatas, cujo custo no pior dos casos é O(|I||M|). O ciclo das linhas 11 a 15 é executado, no pior caso |M-1| vezes. Na linha 12 é retornado o BDD referente a premissa cujo custo é |P1|. De maneira geral, o tempo necessário para a computação das implicações de todos os atributos é O(|M|(|M||G|(|M|($\sum_{i=Size}^{p} C_{p,i}$) + (|I||M|)(|M|+|P1|)))).

4.3 Algoritmo PImplicPBDD

O Algoritmo $PImplicPBDD$ utiliza uma otimização para a redução da quantidade de premissas necessárias para a obtenção das regras e também uma estrutura de dados denominada Diagrama Binário de Decisão (BDD) para simplificar a representação de um contexto formal. Utiliza também de um modelo de computação paralela para a geração e obtenção simultânea das diferentes regras de implicação.

O algoritmo recebe como entrada um contexto formal (G, M, I), e sua saída é um conjunto de implicações próprias.

Algoritmo 13: pImplicPBDD

Input : contexto formal (G, M, I)
Output: conjunto de implicações Imp com suporte maior que 0

```
1 Imp = ∅;
2 foreach m ∈ M do
3     Imp = Imp ∪ AtrImpPBDD(m, contextoformal(G, M, I))
4 return Imp
```

A linha 1 inicializa o conjunto Imp vazio. O seguinte laço (linhas 2 e 3) através do algoritmo $atrImpPBDD$, passando como parâmetro o atributo m e o contexto formal obtém o conjunto de de implicações próprias para o atributo m.

Algoritmo 14: atrImpPBDD

Input : atributo, contexto formal (G,M,I)
Output: conjunto de implicações imp com suporte maior que 0

```
1  Imp = ∅;
2  BddC = primeAtrSetBDD(m);
3  P =m";
4  Size =1;
5  Pa = ∅;
6  ResultImpAtr = ∅;
7  while size < |P| do
8      C = getP(P, Size, 0, ResultImpAtr, Pa);
9      Pc = getCPBDD(C,Pa);
10     foreach P1 ⊂ Pc do
11         BddP = primeAtrSetPPBDD(P1);
12         if BddC ≠ 0 AND BddP ≠ 0 AND BddC = BddP then
13             Pa = Pa ∪ {P1};
14             Imp = imp ∪ {P1 → m};
15     Size++;
16 return Imp
```

A linha 1 inicializa o conjunto Imp vazio. O $BddC$ através do Algoritmo $primeAtrSetPPBDD$ recebe e armazena o BDD contendo todos os objetos do atributo m (linha 2). Este BDD será usado na verificação da igualdade entre a premissa BDD ($P1$) e a conclusão BDD (m).

P registra todos os atributos que contêm os mesmos objetos de m (linha 3).

O $Size$ determina o tamanho de cada premissa, pois o menor tamanho possível é 1 (uma implicação do tipo $b \rightarrow a$), o valor é inicializado com 1 (Linha 4).

Pa armazena um conjunto de premissas auxiliares que podem gerar uma implicação usando m como conclusão (na linha 5 Pa é inicializado como vazio). Nas linhas 7-15, o conjunto de premissas mínimas é encontrado e é limitado por |P|. Na linha 8, o conjunto

C obtém todas as combinações de tamanho $Size$ de elementos em P. Na Linha 9, o conjunto de premissas candidatas é criado através do Algoritmo $getPCPBDD$.

Para cada premissa candidata $P1 \subset PC$ o $BddP$ armazena o BDD que contém todos os objetos da premissa. Na linha 12, é feita uma verificação entre $BddC$ e $BddP$ para garantir que premissa P1 e conclusão M resultem em uma implicação válida. Se $BddC = BddP$ a implicação é válida. Considerando a implicação é válida, a premissa $p1$ é adicionada à hipótese de premissas Pa e também a implicação $\{P1 \rightarrow M\}$ é adicionada à lista de implicações Imp linha 14.

Algoritmo 15: primeAtrSetPPBDD

```
1  Function primeAtrSetPPBDD(ListAttributes):
2      BddNewExt = BddCxt;
3      foreach it ∈ ListAttributes do
4          BddNewExt& =BddNewExt.And(it);
5      return BddNewExt;
```

O algoritmo $primeAtrSetPPBDD$ é responsável por obter o BDD que contém todos os objetos da lista dos atributos passados como parâmetros.

A função calcula a conjunção entre o BDD que representa todo o contexto formal e BDD de cada atributo da lista de atributos. Como um resultado é retornado um BDD contém apenas os objetos que contém todos os atributos.

Algoritmo 16: getPPPBDD

```
1  Function getPPPBDD(Atr, Len, StartPosition, Result, Pa):
2      if Len = 0 then
3          D = D ∪ {Result};
4          return;
5      for i = StartPosition to i <= |Atr| − Len do
6          Result[|Result| − Len] = Atr[i];
7          if Result ⊂ Pa then
8              continue;
9          getP(Atr, Len − 1, i + 1, Result);
10     return D;
```

O Algoritmo $getPPPBDD$ obtém as combinações de tamanho $Size$ a partir de elementos em P. Nas linhas 2-4 é verificada a condição de parada para a recursividade. Se a premissa já foi criada (Len = 0), esta é incluída na lista de resultados D e a recursividade é finalizada. O laço de repetição nas linhas 5-9 monta a próxima premissa da combinação e verifica se esta premissa não foi utilizada nas implicações anteriores. Se esta premissa foi utilizada em alguma implicação anterior a sequencia de geração utilizando esta premissa é descartada. Como saída é gerada a lista de premissas com o tamanho de $Size$.

Algoritmo 17: getPCPBDD

```
1 Function getPCPBDD(C,Pa):
2     D = ∅;
3     foreach a ∈A|A⊂Pa do
4         foreach B ⊂C do
5             if a ∉B then
6                 D =Pc/B;
7     return D;
```

O Algoritmo 17 obtém todos os subconjuntos que não contêm um atributo que pertence à premissa Pa. Ele recebe, como parâmetro, os conjuntos C e Pa e retorna um conjunto D de premissas.

Ao analisar a complexidade de tempo do algoritmo $PImplicPBDD$, o custo relacionado ao primeiro laço (linhas 2 a 16) é no pior caso |M|, quando todos os atributos são considerados como possíveis conclusões. Na linha 4, P recebe um conjunto de atributos comuns ao atributo selecionado para conclusão, nesse caso o atributo m e derivado duas vezes, logo o custo no pior caso é O(|M||G|). No laço (linhas 8 a 16) são computadas as premissas da implicação cuja conclusão é m com o custo no pior caso, O(|M-1|) que seria todos os atributos do contexto formal menos o atributo selecionado para conclusão. Na linha 9 é realizada uma combinação, para gerar premissas candidatas de acordo com a quantidade de atributos permitida para cada premissa, assim quando Size = 1 são geradas premissas candidatas em que cada uma só pode conter 1 atributo , portanto o custo relacionado a esta combinação é O($\sum_{i=1}^{n} C_{n,i}$). Na linha 10 é chamada a heurística que realiza o corte no conjunto de premissas candidatas, cujo custo no pior dos casos é O(|I||M|). O ciclo das linhas 11 a 15 é executado, no pior caso |M-1| vezes. Na linha 12 é retornado o BDD referente a premissa cujo custo é |P1|. De maneira geral, o tempo necessário para a computação das implicações de todos os atributos é O(|M|(|M||G|(|M-1|($\sum_{i=Size}^{p} C_{p,i}$) + (|I||M|)(|M-1||P1|))))

A Figura 13 detalha o funcionamento do paralelismo do algoritmo PImplicPBDD. Para cada núcleo do processador, uma *thread* é criada para processar uma conclusão distinta.

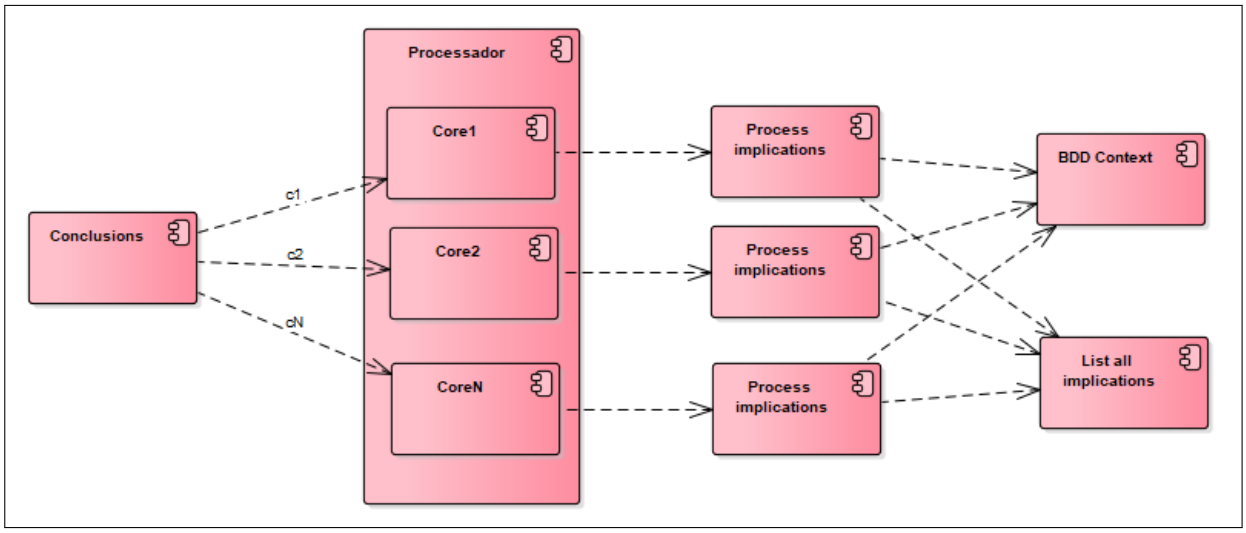


Figura 13 – Modelo de computação paralela do algoritmo PImplicPBDD

Com o modelo de computação *multinúcleo* é possível processar conclusões independentes em núcleos independentes. A elaboração do algoritmo $PImplicPBDD$ foi realizada visando processar uma conclusão independente de outra. Devido a este processamento de conclusões independentes, foi definido o particionamento como desenho de computação paralela. Como a base do algoritmo $PImplicPBDD$ é o processamento de conclusões independentes, a decomposição do domínio é ideal para este modelo sendo que, conclusões distintas são atribuídas para operar com dados distintos. Para ser possível determinar quais partes do algoritmo seriam paralelizadas, foi utilizado o modelo de programação de software baseado em biblioteca. Este modelo permite uma maior flexibilidade.

5 EXPERIMENTOS E ANÁLISE DOS RESULTADOS

O objetivo principal dos experimentos é analisar o desempenho de todos os algoritmos que utilizam as melhorias propostas neste trabalho (Otimização, BDD e Paralelismo), na geração do conjunto de implicações próprias.

Foram realizadas comparações entre o PropIm e todos os algoritmos propostos (ImplicP, ImplicPBDD e PImplicPBDD) neste trabalho. Também foram realizados comparações entre o ImplicP e ImplicPBDD, ImplicPBDD e PImplicPBDD. As comparações foram realizadas utilizando contextos sintéticos em que se varia o número de objetos, atributos e densidade.

Os algoritmos foram implementados em Java e executados em um Servidor IBM (OS Ubuntu 16.04) com processador Intel Xeon (3.1 GHz) de 4 núcleos, 32 GB de RAM e 30 GB de armazenamento em disco SSD.

Inicialmente, optou-se por utilizar contextos de 1.000, 10.000 e 100.000 objetos combinados com conjuntos de 16, 20 e 23 atributos, com densidades variando do mínimo , 30 %, 50 %, 70 % e a densidade máxima para cada contexto gerado. Para a criação de contextos foi utilizada a ferramenta SCGAz (RIMSA; ZÁRATE; SONG, 2009) sendo que é possível gerar contextos aleatórios dentro dos limites de densidades e tamanhos controlados. Foi adicionada uma restrição para o limite máximo de tempo para extrair as regras de implicação. Este limite foi estabelecido para 14 dias visto que, após testes realizados com o algoritmo *PropIm*, este não retornou resultado ou terminou após longos períodos de processamento.

Observe que a tabela de incidentes gerada aleatoriamente pode variar entre dois contextos com as mesmas dimensões e densidades, dando origem a diferentes conjuntos de implicações. Portanto, um desempenho diferenciado pode ser obtido para cada contexto. Para os testes, criamos 2 amostras para cada contexto sintético considerado. Calculamos um tempo de execução médio com base no tamanho e densidade do contexto.

Em todos os testes o desempenho dos algoritmos foram comparados *pairwise* executando o mesmo contexto com um tempo máximo de 14 dias para o processamento das implicações próprias.

Um teste separado foi realizado para demonstrar a ganho do BDD (ImplicP) se comparado a otimização para a redução do total de premissas (ImplicPBDD). Foram utilizados contextos de 400.000, 800.000 e 1200.000 objetos combinados com 21 atributos e densidade de 50%. O aumento no numero de objetos do contexto foi justamente para demonstrar o beneficio isolado do BDD. Por fim, foi realizado um teste para demonstrar o algoritmo (PImplicPBDD) processando um contexto formal real baseado em uma amostragem do Linkedin. Com o intuito de também demonstrar os benefícios da ordenação de variáveis do BDD, foi utilizado o contexto real sem nenhuma ordenação de variáveis e também um contexto sendo que foi aplicado indice de Jaccard (JACCARD, 1901) para calcular uma medida da similaridade entre dois conjuntos e ter uma ordenação por similaridade.

5.0.1 Resultados ImplicP

Nesta seção é demonstrado o resultado da comparação entre os algoritmos PropIm *vs.* ImplicP.

5.0.1.1 PropIm *vs.* ImplicP

Os testes utilizando contextos sintéticos entre o algoritmo ImplicP e PropIm demonstrou que a solução proposta nesta dissertação (otimização), reduz significativamente o tempo necessário para obtenção das regras de implicação. A Tabela 12 e os Gráficos 1 e 2 demonstram a comparação de desempenho destes algoritmos.

Tabela 12 – Resultados (PropIm) X (ImplicP) utilizando contextos sintéticos

|G|	|M|	Den(%)	PropIm (Minutos)	ImplicP (Minutos)	Speed Up
1000	16	30	6,11	2,53	2,42
1000	16	50	6,30	3,49	1,81
1000	16	70	6,38	3,59	1,78
10000	16	30	6,49	4,10	1,58
10000	16	50	6,57	4,38	1,50
10000	16	70	7,20	5,40	1,33
1000	20	30	2657	529	5,02
1000	20	50	2925	1158	2,53
1000	20	70	2081	1430	1,46
10000	20	30	2433	933	2,61
10000	20	50	3078	1738	1,77
10000	20	70	3198	1706	1,87
100000	20	30	3460	1425	2,43
100000	20	50	3068	2067	1,48
100000	20	70	2601	1869	1,39

Note que, analisando o Gráfico 1, o ganho de velocidade do ImplicP em contextos com 16 atributos chegou a ser 2 vezes maior que o PropIm. Este ganho é baseado na eficiência da otimização para redução da quantidade de combinações de premissas necessárias para uma implicação. A otimização atua de forma que, se já foi gerada uma implicação na forma de $\{A, B\} \rightarrow \{C\}$, novas combinações que iram utilizar os atributos A, B para a conclusão C não serão mais gerados. Caso fosse necessário gerar combinações com três premissas, as combinações $\{A, B, *\} \rightarrow \{C\}$ não seriam geradas, reduzindo a quantidade total de combinações para uma determinada conclusão. Pelo Gráfico 2 onde são utilizados contextos com 20 atributos, a otimização atuou de forma mais eficiente obtendo um ganho superior a cinco vezes se comparado ao original. Quanto maior a densidade do contexto, menor o desempenho. Com uma densidade maior, a tabela de incidências contém mais marcações sendo necessário na maioria das vezes, gerar mais combinações entre atributos.

Gráfico 1 – (PropIm) X (ImplicP) utilizando contextos sintéticos com 16 atributos

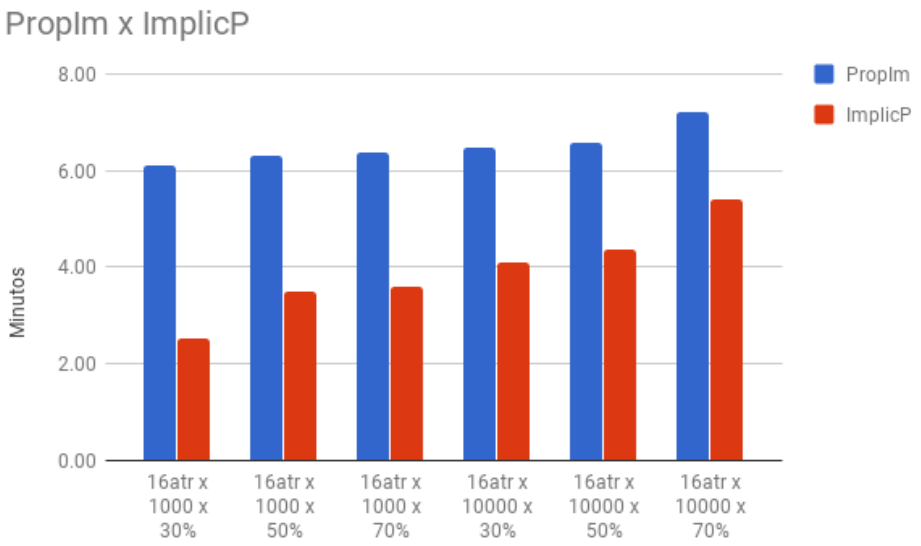
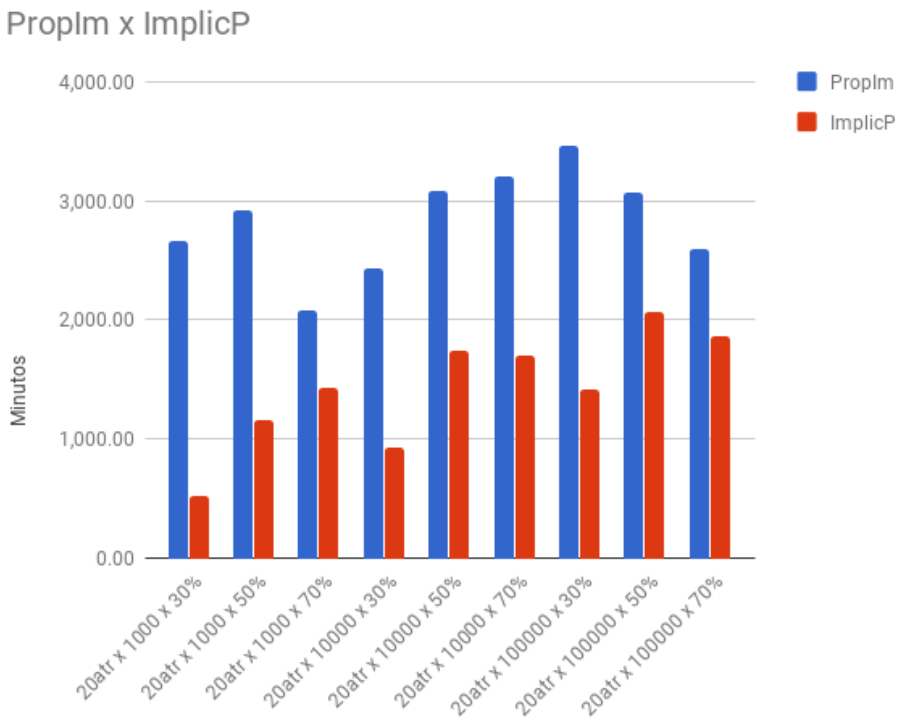


Gráfico 2 – (PropIm) X (ImplicP) utilizando contextos sintéticos com 20 atributos



5.0.2 Resultados ImplicPBDD

Nesta seção são demonstrados os resultados das comparações entre os algoritmos PropIm *vs.* ImplicPBDD e ImplicP *vs.* ImplicPBDD .

5.0.2.1 PropIm *vs.* ImplicPBDD

Os testes utilizando os contextos sintéticos entre o algoritmo ImplicPBDD e PropIm demonstrou que o algoritmo ImplicPBDD foi mais eficiente em todos os testes. A tabela 13 e as Figuras 3 e 4 demonstram a comparação de desempenho destes algoritmos.

Tabela 13 – Resultados (PropIm) X (ImplicPBDD) utilizando contextos sintéticos

|G|	|M|	Den(%)	PropIm (Minutos)	ImplicPBDD (Minutos)	Speed Up
1000	16	30	6,11	2,38	2,57
1000	16	50	6,30	3,07	2,05
1000	16	70	6,38	3,49	1,83
10000	16	30	6,49	3,13	2,07
10000	16	50	6,57	4,12	1,59
10000	16	70	7,20	4,58	1,57
1000	20	30	2657	520	5,10
1000	20	50	2925	1185	2,47
1000	20	70	2081	14947	1,39
10000	20	30	2433	581	4,19
10000	20	50	3078	1532	2,01
10000	20	70	3198	1954	1,64
100000	20	30	3460	1406	2,46
100000	20	50	3068	1595	1,92
100000	20	70	2601	1664	1,56

Analisando as Figuras 3 e 4 o ganho de velocidade do ImplicPBDD foi superior aos algoritmos PropIm e ImplicP. Em todos os contextos o ImplicPBDD obteve um desempenho melhor. Note que, além de utilizar a otimização do ImplicP também foi utilizando o BDD para reduzir o contexto formal e facilitar a manipulação dos objetos e atributos melhorando o desempenho do algoritmo. Utilizando o BDD, objetos que contém os mesmos atributos não são duplicados no contexto porque, estes, são representados pela mesma formula booleana. Esta abordagem é diferente do PropIm que representa o contexto formal como matriz e, utiliza listas para controlar objetos e atributos. A forma de representação do PropIm pode gerar duplicidade de objetos e atributos reduzindo o desempenho do algoritmo. Mesmo utilizando o BDD percebeu-se que na maioria das situações, quanto maior a densidade do contexto, menor o desempenho.

Gráfico 3 – (PropIm) X (ImplicPBDD) utilizando contextos sintéticos com 16 atributos

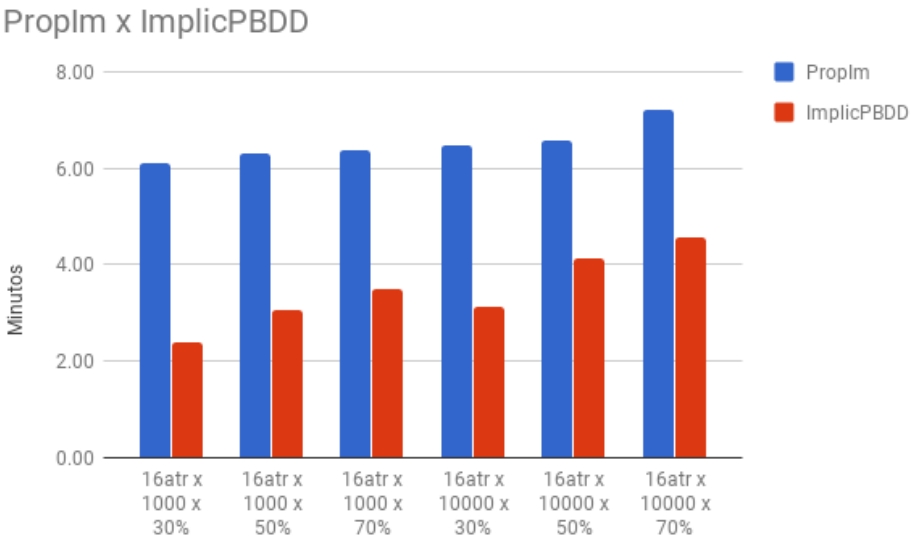
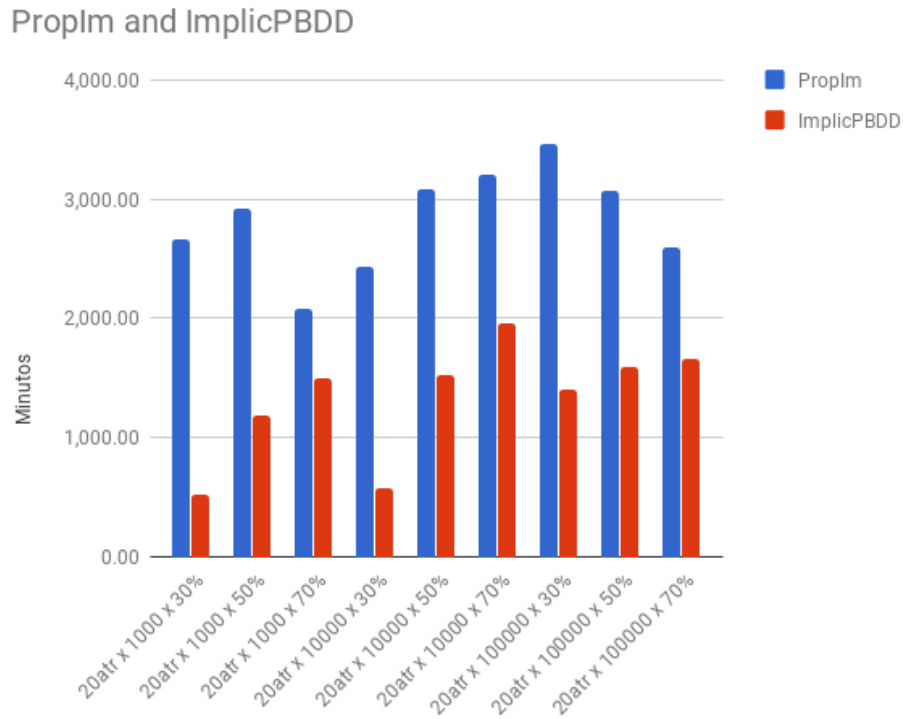


Gráfico 4 – (PropIm) X (ImplicPBDD) utilizando contextos sintéticos com 20 atributos



5.0.2.2 ImplicP *vs.* ImplicPBDD

Os testes utilizando os contextos sintéticos entre o algoritmo ImplicPBDD e ImplicP demonstrou que o algoritmo ImplicPBDD foi mais eficiente em todos os testes. A tabela 14 e as Figuras 5 e 6 demonstram a comparação de desempenho destes algoritmos.

Tabela 14 – Resultados (ImplicP) X (ImplicPBDD) utilizando contextos sintéticos

|G|	|M|	Den(%)	ImplicP (Minutos)	ImplicPBDD (Minutos)	Speed Up
1000	16	30	2,53	2,38	1,06
1000	16	50	3,49	3,07	1,14
1000	16	70	3,59	3,49	1,03
10000	16	30	4,1	3,13	1,31
10000	16	50	4,38	4,12	1,06
10000	16	70	5,4	4,58	1,18
1000	20	30	529	520	1,02
1000	20	50	1158	1145	1,01
1000	20	70	1430	1414	1,01
10000	20	30	933	581	1,61
10000	20	50	1738	1532	1,13
10000	20	70	1706	1684	1,01
100000	20	30	1425	1406	1,01
100000	20	50	2067	1595	1,30
100000	20	70	1869	1664	1,12

Analisando as Figuras 5 e 6 em todos os contextos o ImplicPBDD obteve um melhor desempenho que o ImplicP. Note que, além de utilizar a otimização do ImplicP também foi utilizando o BDD para reduzir o tamanho da representação do contexto formal e facilitar a manipulação dos objetos e atributos melhorando o desempenho do algoritmo. Um exemplo de utilização do BDD no algoritmo ImplicPBDD pode ser visto na Seção 4.2. Utilizando o BDD, objetos que contém os mesmos atributos não são duplicados no contexto porque, estes, são representados pela mesma formula booleana no BDD.

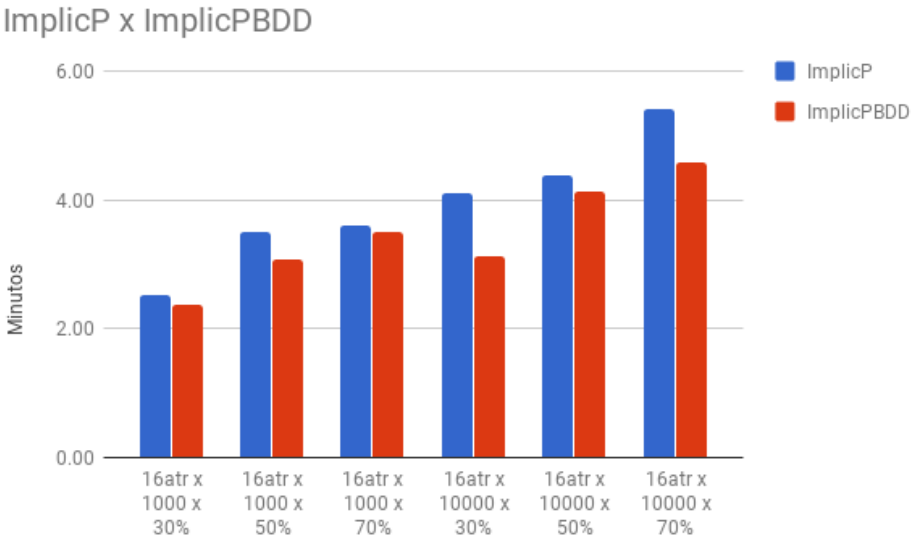


Gráfico 5 – (ImplicP) X (ImplicPBDD) utilizando contextos sintéticos com 16 atributos

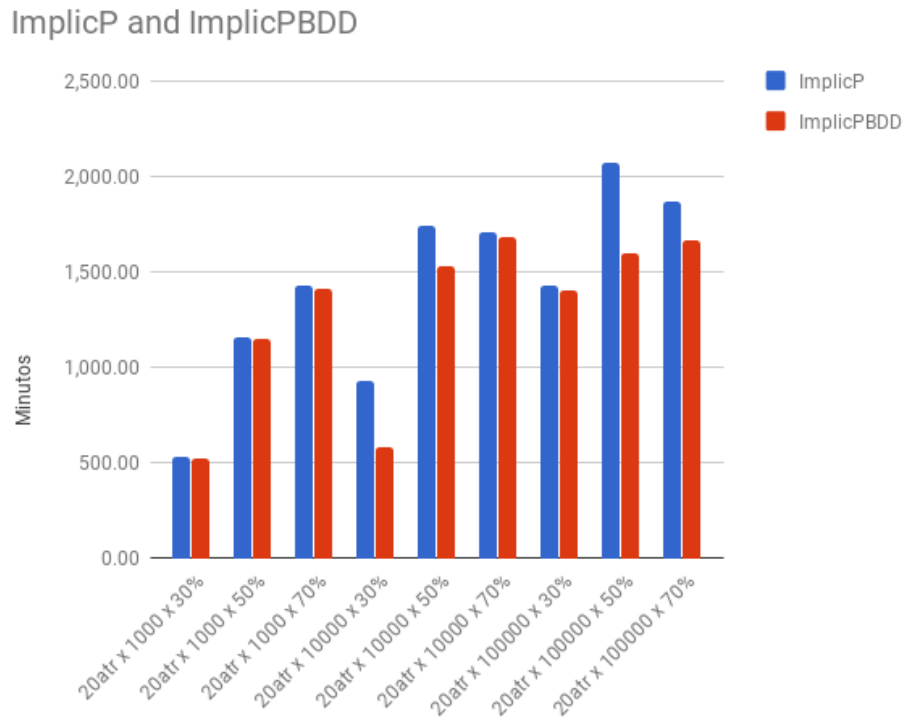


Gráfico 6 – (ImplicP) X (ImplicPBDD) utilizando contextos sintéticos com 20 atributos

Tabela 15 – Resultados (ImplicP) X (ImplicPBDD) utilizando contextos sintéticos com uma maior volume de objetos

|G|	|M|	Den(%)	ImplicP (Min)	ImplicPBDD (Min)	Speed Up
400000	21	50%	5513	5305	4%
800000	21	50%	6742	6070	11%
1200000	21	50%	8298	7111	16%

Pode-se perceber pelas Tabelas 14 e 15 que o BDD foi superior em todos os contexto desde 1.000 até 1.200.00 objetos. Quanto maior o volume de objetos, o BDD se torna mais eficiente por sua representação mais compacta do contexto formal e sua eficiente manipulação. Quanto maior o contexto, maiores são as chances de existirem objetos duplicados no contexto, e o BDD se tornar mais compacto. Este comportamento pode ser visto pela Tabela 15.

5.0.3 Resultados PImplicPBDD

Nesta seção são demonstrados os resultados das comparações entre os algoritmos PropIm *vs.* PImplicPBDD e ImplicPBDD *vs.* PImplicPBDD.

5.0.3.1 PropIm *vs.* PImplicPBDD

Os testes utilizando os contextos sintéticos entre o algoritmo PImplicPBDD e PropIm demonstrou que o algoritmo PImplicPBDD foi mais rápido em todos os testes inclusive, foi o único algoritmo que possibilitou o processamento de contextos com 23 atributos. A tabela 22 e as Figuras 9, 10 demonstram a comparação de desempenho dos algoritmos.

Tabela 16 – Resultados (PropIm) X (PImplicPBDD) utilizando contextos sintéticos

|G|	|M|	Den(%)	PropIm (Min)	PImplicPBDD (Min)	Speed Up
1000	16	30	6,11	1,08	5,66
1000	16	50	6,30	2,10	3,00
1000	16	70	6,38	2,73	2,34
10000	16	30	6,49	1,85	3,51
10000	16	50	6,57	3,76	1,75
10000	16	70	7,20	4,30	1,67
1000	20	30	2657	118	22,39
1000	20	50	2925	264	11,07
1000	20	70	2081	328	6,34
10000	20	30	2433	144	16,89
10000	20	50	3078	343	8,96
10000	20	70	3198	355	8,99
100000	20	30	3460	683	5,06
100000	20	50	3068	371	8,25
100000	20	70	2601	517	5,02
1000	23	30	.	2344	.
1000	23	50	.	4724	.
1000	23	70	.	4871	.
10000	23	30	.	2432	.
10000	23	50	.	16219	.
10000	23	70	.	16422	.
100000	23	30	.	3891	.
100000	23	50	.	18580	.
100000	23	70	.	18958	.

Por utilizar as abordagens do ImplicPBDD junto com computação paralela, o PImplicPBDD obteve ganhos em todos os contextos com 16 e 20 atributos. O processamento de contextos com 20 atributos chegou a ter um ganho de 22 vezes se comparado ao PropIm. Este ganho deve-se ao paralelismo em que, múltiplas conclusões puderam ser processadas simultaneamente reduzindo o tempo total de processamento.

Gráfico 7 – (PropIm) X (PImplicPBDD) utilizando contextos sintéticos com 16 atributos

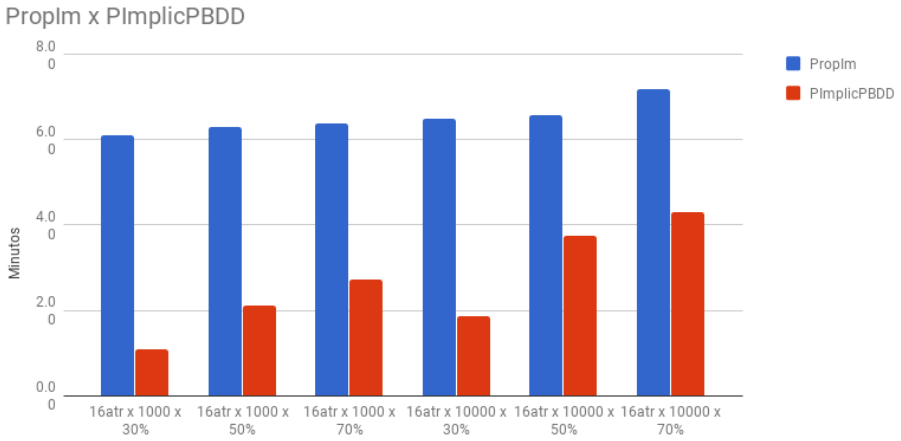
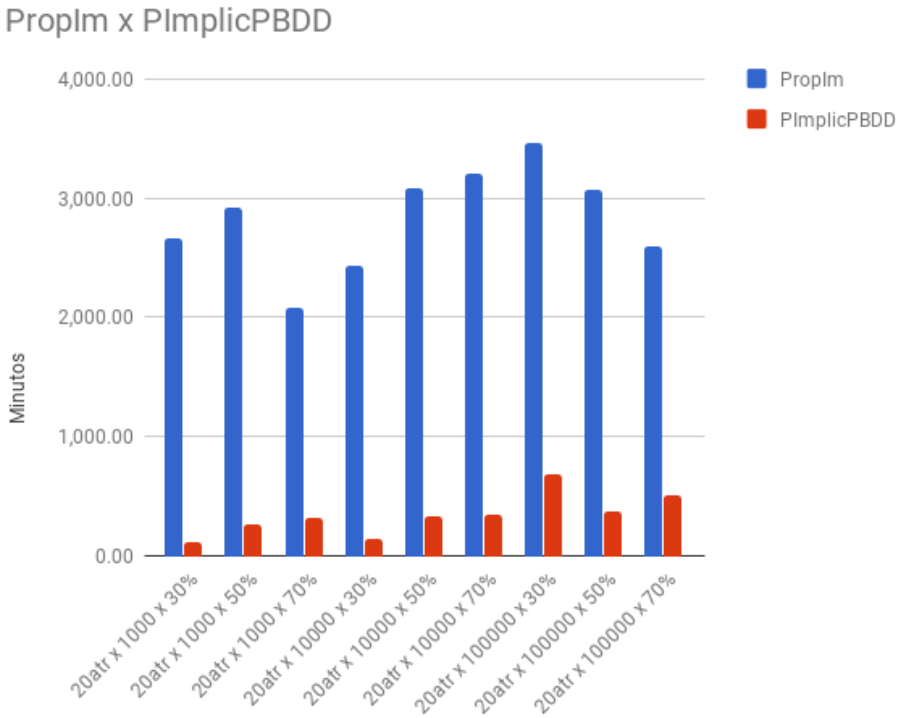


Gráfico 8 – (PropIm) X (PImplicPBDD) utilizando contextos sintéticos com 20 atributos



5.0.3.2 ImplicPBDD *vs.* PImplicPBDD

Os testes utilizando os contextos sintéticos entre o algoritmo PImplicPBDD e ImplicPBDD demonstrou que o algoritmo PImplicPBDD foi mais rápido em todos os testes.

A Tabela 17 e as Figuras 9, 10 demonstram a comparação de desempenho dos algoritmos.

Tabela 17 – Resultados (ImplicPBDD) X (PImplicPBDD) utilizando contextos sintéticos

|G|	|M|	Den(%)	ImplicPBDD (Minutos)	PImplicPBDD (Minutos)	Speed Up
1000	16	30	2,38	1,08	2,20
1000	16	50	3,07	2,1	1,46
1000	16	70	3,49	2,73	1,28
10000	16	30	3,13	1,85	1,69
10000	16	50	4,12	3,76	1,10
10000	16	70	4,58	4,3	1,07
1000	20	30	520	118	4,41
1000	20	50	1145	264	4,34
1000	20	70	1414	328	4,31
10000	20	30	581	144	4,03
10000	20	50	1532	343	4,47
10000	20	70	1684	355	4,74
100000	20	30	1406	269	5,23
100000	20	50	1595	371	4,30
100000	20	70	1664	517	3,22
1000	23	30	.	2344	.
1000	23	50	.	4724	.
1000	23	70	.	4871	.
10000	23	30	.	2432	.
10000	23	50	.	16219	.
10000	23	70	.	16422	.
100000	23	30	.	3891	.
100000	23	50	.	18580	.
100000	23	70	.	18958	.

Por utilizar as abordagens do ImplicPBDD junto com computação paralela, o PImplicPBDD obteve ganhos em todos os contextos com 16 e 20 atributos. O processamento de contextos com 20 atributos chegou a ter um ganho de 22 vezes se comparado ao PropIm. Este ganho deve-se ao paralelismo em que, múltiplas conclusões puderam ser processadas simultaneamente reduzindo o tempo total de processamento.

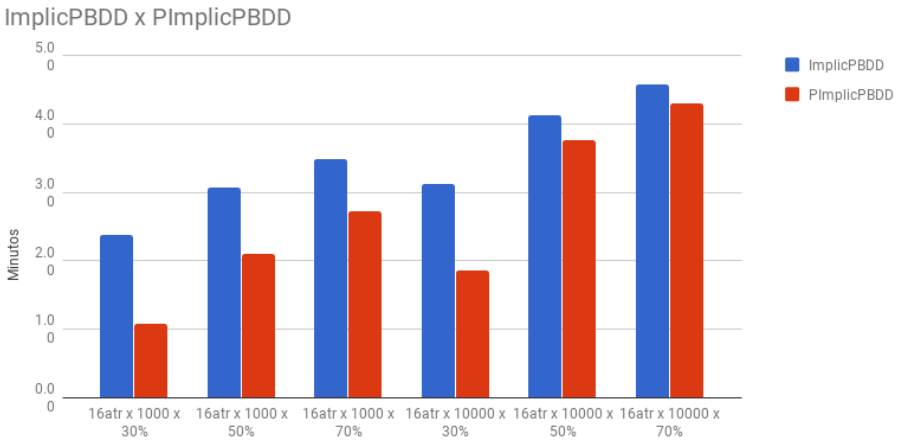


Gráfico 9 – (ImplicPBDD) X (PImplicPBDD) utilizando contextos sintéticos com 16 atributos

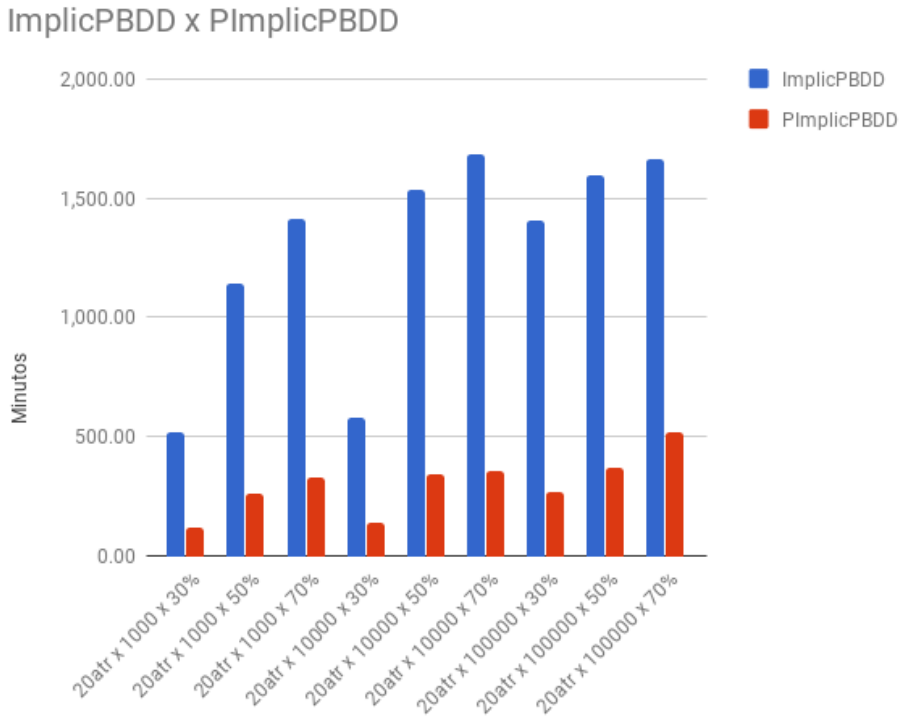


Gráfico 10 – (ImplicPBDD) X (PImplicPBDD) utilizando contextos sintéticos com 20 atributos

Os testes utilizando os contextos sintéticos entre o algoritmo PImplicPBDD e ImplicPBDD demonstrou que o algoritmo PImplicPBDD foi mais rápido em todos os testes. A Tabela 17 e as Figuras 9, 10 demonstram a comparação de desempenho dos algoritmos.

Por utilizar as abordagens do ImplicPBDD junto com computação paralela, o PIm-

plicPBDD obteve ganhos em todos os contextos com 16 e 20 atributos. O processamento de contextos com 20 atributos chegou a ter um ganho de 22 vezes se comparado ao PropIm. Este ganho deve-se ao paralelismo em que, múltiplas conclusões puderam ser processadas simultaneamente reduzindo o tempo total de processamento.

5.0.3.3 PropIm *vs.* PImplicPBDD em um contexto formal real

O contexto formal real foi criado baseado na mostragem extraída da rede social Linkedin que contém como conclusão 10 profissões e 24 habilidades como premissas. Foram utilizados todos os registros disponibilizados na amostragem como objetos do contexto, gerando um total de 1.269 objetos. O teste utilizando o contexto do Linkedin demonstrou que o algoritmo PImplicPBDD processou todo o contexto em 4 dias enquanto o PropIm após 14 dias não retornou nenhum resultado. O PImplicPBDD por utilizar todas as soluções propostas neste artigo conseguiu processar todo o contexto em menos da metade do tempo máximo definido. Repare que, o contexto do Linkedin por conter 24 atributos, necessita de uma maior geração de combinações e premissas candidatas para cada conclusão tornando inviável para o PropIm que funciona com somente um núcleo e sem otimização na geração combinações processar o contexto Este ganho permite obter novas informações utilizando contextos maiores.

A Tabela 18 descreve três implicações que foram extraídas do contexto formal do Linkedin com a conclusão de Data Scientist.

Tabela 18 – Amostragem de implicações próprias extraídas do contexto formal do Linkedin

$A \rightarrow b$
{algorithms, digital marketing] → [data scientist}
{computer architecture, office applications, user interface] → [data scientist}
{computer architecture, user experience, user interface] → [data scientist}

5.0.3.4 Utilização do Algoritmo PImplicPBDD em um contexto formal real utilizando ordenação de variáveis com o BDD

Para uma melhor demonstração de desempenho do algoritmo PImplicPBDD, além de ser utilizado o contexto formal do Linkedin sem nenhuma ordenação de variáveis, foi utilizado o índice de Jaccard para gerar o mesmo contexto do Linkedin porem como uma ordenação diferente de variáveis utilizando uma medida da similaridade. O intuito deste teste foi de mostrar a diferença de desempenho quando ocorre uma ordenação de variáveis

do BDD. O Índice de Jaccard é uma medida da similaridade entre dois conjuntos. Por exemplo, se tivermos dois conjuntos A e B, com os seguintes elementos A = BB; BC; DD; DI; EF e B = BB; BD; DD; DF; EF, podemos calcular o Índice de Jaccard pela intersecção entre esses dois conjuntos divido pela união deles (JACCARD, 1901).

A Tabela 20 e a Tabela 19 descrevem todas as habilidades e profissões que foram utilizadas para a geração dos contextos.

Tabela 19 – Habilidades utilizadas no contexto formal do Linkedin

Habilidades sem ordenação	Habilidades com índice de Jaccard
agile methodologies	software engineering
algorithms	computer languages
artificial intelligence	databases
basic software	basic software
budgets	information systems
databases	telecom
computer languages	agile methodologies
digital marketing	budgets
embedded systems	tests
enterprise architecture	graphic design
entrepreneurship	artificial intelligence
formal language and automata	enterprise architecture
graphic design	office applications
information systems	virtualization
office applications	solutions
social media	user experience
software engineering	user interface
solutions	algorithms
telecom	social media
user experience	entrepreneurship
user interface	formal language and automata
tests	embedded systems
virtualization	digital marketing

O tempo necessário para o processamento do contexto sem ordenação de variáveis foi de 3 dias e 20 horas enquanto o tempo para o processamento do contexto com ordenação foi de 3 dias e 12 horas. Conforme descrito pela Seção 2.2 a ordenação de variáveis no BDD é um fator muito importante para o desempenho do próprio. Com a utilização do índice de Jaccard foi possivel obter um desempenho de 12% se comparado ao contexto sem ordenação.

Tabela 20 – Profissões utilizadas no contexto formal do Linkedin

Profissões sem ordenação	Profissões com índice de Jaccard
software developer	software developer
professor	professor
data scientist	analista de teste
project manager	engenheiro de software
analista de teste	data scientist
founder	project manager
IT manager	founder
ux/ui-designer	ux/ui-designer
arquiteto de software	arquiteto de software
engenheiro de software	IT manager

5.0.4 Impacto da quantidade de premissas no tempo de execução

Analisando os resultados obtidos com todos os testes realizados, é possível perceber que o tempo de execução dos algoritmos avaliados tende a crescer quando aumentamos a quantidade de atributos e, principalmente, a densidade de incidência no contexto formal. Com o objetivo de verificar a razão desse aumento do tempo computacional, foi avaliado a quantidade de atributos nas premissas das regras de implicações encontradas pelos algoritmos. Para isso, foi determinado um intervalo de confiança sobre a média ($\alpha = 0.05$) para determinar quantos atributos são encontrados nas premissas das regras de implicação.

A Tabela 21 apresenta o intervalo de confiança (intervalo mínimo e máximo, lidos na Tabela como Int.Min Premissas e Int. Max. Premissas, respectivamente) para a média de atributos nas premissas encontrados para cada cenário avaliado nesse trabalho. Além disso, foi verificado a quantidade de premissas mais frequente para cada cenário (moda), qual a porcentagem que a quantidade de premissas mais frequente (moda) representa considerando o total de regras, e por fim, o total de implicações. É importante ressaltar que, independentemente dos algoritmos (*PropIm*, *ImplicP*, *ImplicPBDD* e *PImplicPBDD*), o conjunto das regras de implicações é o mesmo.

Tabela 21 – Quantidade de atributos nas premissas das regras de implicações

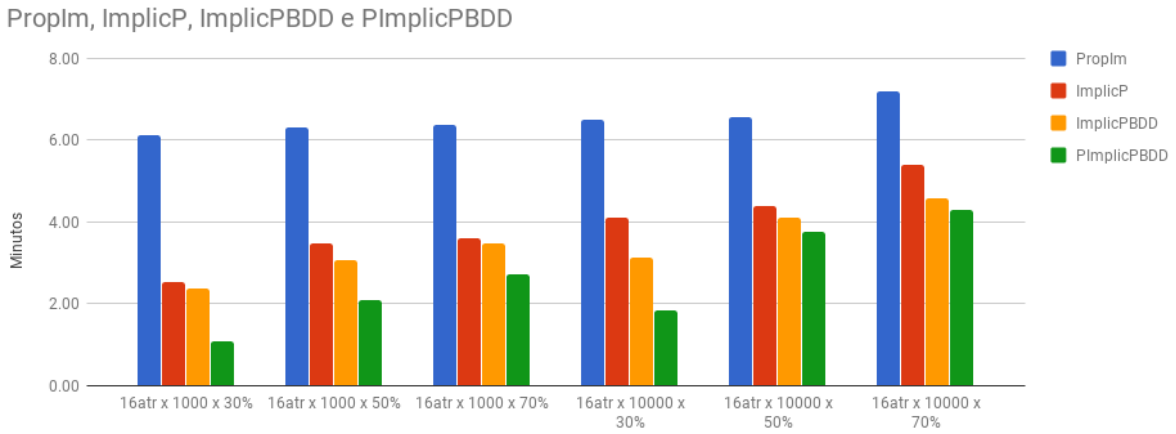
$\|G\|$	$\|M\|$	Den(%)	Int. Min. Premissas	Int. Max. Premissas	Moda	% da Moda na base	Total de Implicações
1000	16	30	4,99	5,01	5	91,28%	4.357
1000	16	50	7,66	7,68	8	61,81%	6.300
1000	16	70	11,80	12,11	12	86,36%	22
10000	16	30	6,01	6,01	6	99,08%	2.489
10000	16	50	10,52	10,58	11	54,91%	1.273
10000	16	70	-	-	-	-	0
1000	20	30	5,11	5,12	5	85,00%	24.576
1000	20	50	7,71	7,72	8	65,25%	80.695
1000	20	70	11,61	11,67	12	43,30%	3.270
10000	20	30	6,60	6,60	7	59,93%	50.390
10000	20	50	10,51	10,52	11	51,25%	78.021
10000	20	70	16,74	17,08	17	90,91%	11
100000	20	30	8,86	8,86	9	85,97%	135.921
100000	20	50	13,37	13,39	13	59,97%	13.409
100000	20	70	-	-	-	-	0
1000	23	30	5,09	5,09	5	87,66%	58.582
1000	23	50	7,83	7,84	8	71,37%	389.572
1000	23	70	11,69	11,70	12	48,52%	69.091
10000	23	30	6,82	6,83	7	82,11%	244.704
10000	23	50	10,69	10,69	11	65,38%	717.603
10000	23	70	17,05	17,15	17	57,99%	695
100000	23	30	7,87	7,87	8	87,02%	30.256
100000	23	50	12,98	12,98	13	97,87%	55.078
100000	23	70	-	-	-	-	0

Pode-se perceber que os cenários que possuem 30% de densidade (incidência de atributos em relação aos objetos) foram os que apresentaram a menor quantidade de atributos nas premissas, o que, consequentemente, obtiveram o menor tempo para encontrar o conjunto de implicações próprias. Isso acontece pois esse é o melhor cenário para a otimização utilizada nos algoritmos apresentados nesse trabalho, uma vez que ao encontrar um regra mínima, por exemplo, $\{a \rightarrow b\}$, não será mais gerada novas regra utilizando o atributo $\{a\}$ do lado esquerdo da implicação. Em contrapartida, quando analisamos os cenários com 70% de densidade, foi possível verificar que, de maneira geral, temos uma maior quantidade de atributos nas premissas das implicações, o que demanda um maior esforço computacional para encontrar o conjunto de regras de implicação.

5.0.5 Visão geral da execução de tempo de todos os algoritmos

A Figura 11 demonstra o resultado dos testes de desempenho de todos os algoritmos utilizando contextos sintéticos com 16 atributos.

Gráfico 11 – Gráfico 16 atributos



A Figura 12 demonstra o resultado dos testes de desempenho de todos os algoritmos utilizando contextos sintéticos com 20 atributos.

Gráfico 12 – Gráfico 20 atributos

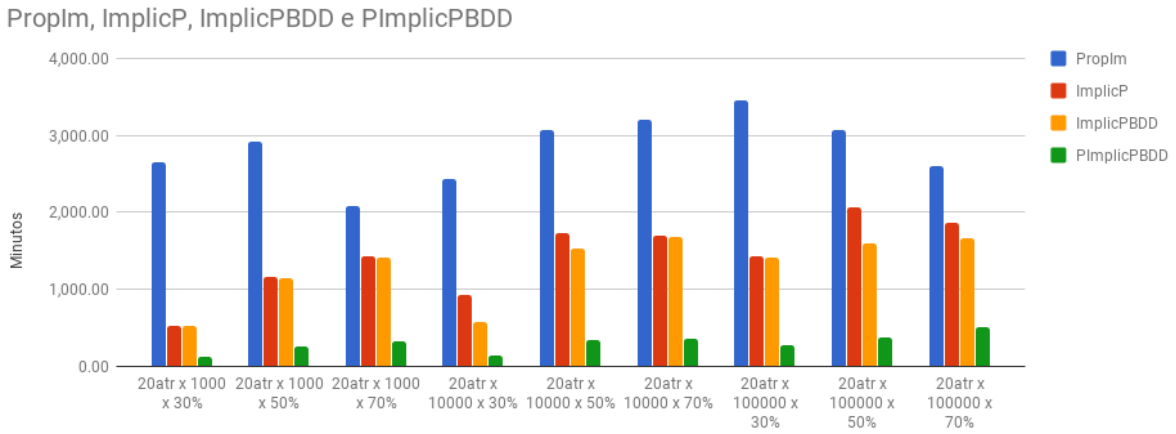


Tabela 22 – Comparativo do aumento de velocidade entre todos os algoritmos

|G|	|M|	Den(%)	PropIm	ImplicP	ImplicPBDD	PImplicPBDD
1000	16	30	6,11	2,53	2,38	1,08
1000	16	50	6,3	3,49	3,07	2,1
1000	16	70	6,38	3,59	3,49	2,73
10000	16	30	6,49	4,1	3,13	1,85
10000	16	50	6,57	4,38	4,12	3,76
10000	16	70	7,2	5,4	4,58	4,3
1000	20	30	2657	529	520	118
1000	20	50	2925	1158	1145	264
1000	20	70	2081	1430	1414	328
10000	20	30	2433	933	581	144
10000	20	50	3078	1738	1532	343
10000	20	70	3198	1706	1684	355
100000	20	30	3460	1425	1406	269
100000	20	50	3068	2067	1595	371
100000	20	70	2601	1869	1664	517

6 CONCLUSÕES E TRABALHOS FUTUROS

Determinar implicações próprias em contextos de alta dimensionalidade pode-se tornar uma tarefa inviável devido ao seu alto custo computacional em realizar a combinação de todos os atributos e as relações com os objetos.

Considerando o cenário em que é desejado encontrar apenas as implicações próprias que contém o subconjunto C, onde $C \subseteq M$, como conclusão, computar todas as regras se torna desnecessário, pois após o processamento do conjunto completo de implicações próprias da base, será necessário filtrar as regras e selecionar apenas aquelas que satisfaçam as restrições de determinados atributos na conclusão.

O objetivo deste trabalho foi propor uma abordagem em que é possível processar implicações próprias em contextos de alta dimensionalidade quando utilizado contextos sintéticos, e no caso do contexto de uma base real (Linkedin), manipular quantidades maiores de atributos. Neste trabalho, conseguimos manipular contextos de 24 atributos.

Para ser possível processar os contextos sintéticos de alta dimensionalidade, foi necessário representar o contexto utilizando BDD e adaptar os operadores de fecho do PImplicPBDD para manipular os objetos e atributos. Além disso, utilizou-se de uma otimização para eliminar a geração desnecessária de combinação de premissas para a obtenção das devidas regras e por fim, foi utilizado de um modelo de computação paralela para a geração e obtenção simultânea das diferentes regras de implicação.

Constatou-se que o PImplicPBDD foi mais rápido quando comparado a todos os outros algoritmos ImplicP, ImplicPBDD e PropIm em todos os contextos criados para os testes. Conforme pode-se observar no Capítulo 5, o ganho de cada abordagem adotada é demonstrada de forma detalhada.

Utilizando contextos que contém 23 atributos o PImplicPBDD permitiu extrair todas as implicações próprias, enquanto o algoritmo PropIm não retornou nenhum resultado com o tempo máximo estabelecido de 14 dias. Observa-se também que com o PImplicPBDD foi possível obter desempenho de até 22 vezes mais rápido na extração das implicações próprias se comparado ao algoritmo original.

A utilização do PImplicPBDD no contexto real baseado na amostragem do Linkedin que contém 24 atributos, 10 conclusões e 1.269 objetos, permitiu extrair todas as implicações próprias, enquanto o algoritmo PropIm não retornou nenhum resultado no tempo máximo estabelecido.

Embora o BDD inicialmente tenha sido a abordagem principal para permitir o processamento de um contexto de alta dimensionalidade, este acabou não tendo um ganho tão significativo como inicialmente esperado. Sugere-se como trabalho futuro a paralelização do BDD de forma a permitir processar um volume ainda maior de dados, pois consegue-se ter uma maior disponibilidade de recursos computacionais para a geração das regras de implicações próprias em menor tempo.

No algoritmo paralelizado apresentado, PImplicPBDD, o mesmo contexto formal é compartilhado entre todas as threads podendo gerar concorrência na utilização do contexto. Neste contexto, como trabalho futuro, sugere-se a criação de um contexto formal para cada thread que poderia processar uma conclusão isoladamente, reduzindo a concorrência no acesso do contexto formal já que, atualmente todas as threads fazem acesso ao mesmo contexto formal.

6.1 Publicações

Por fim, é importante destacar as contribuições científicas produzidas ao longo do trabalho:

- Artigos publicados em Conferência
  - An approach to extract proper implications set from high-dimension formal contexts using Binary Decision Diagram - (ICEIS 2018)
  - Selecionado para concorrer ao prêmio de melhor trabalho do ICEIS 2018

- Convite de extensão
  - An Approach to Extract Proper Implications Set from High-dimension Formal Contexts using Binary Decision Diagram em Information Journal (ISSN: 2078-2489)

REFERÊNCIAS

AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. Mining association rules between sets of items in large databases. In: SIGMOD Rec. [S.l.: s.n.], 1993. v. 22, p. 207–216.

ALLERI, J.; HUCHARD, M. A generic approach for class model normalization. In: PROCEEDINGS OF 2ND. LISBON IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING (ASE). [S.l.], 2008. p. 225–274.

ARÉVALO, G.; FALLERI, J. Building abstractions in class models. In: AFC IN MDA MODELS. [S.l.], 2006. p. 503–527.

BAIXERIES, J. et al. Yet a faster algorithm for building the hasse diagram of a concept lattice. In: Proceedings of the 7th International Conference on Formal Concept Analysis. Berlin, Heidelberg: Springer-Verlag, 2009. (ICFCA '09), p. 162–177. ISBN 978-3-642-01814-5. Disponível em: <http://dx.doi.org/10.1007/978-3-642-01815-2_13>.

BERTET, K.; MONJARDET, B. Fundamental study: The multiple facets of the canonical direct unit implicational basis. Theor. Comput. Sci., Elsevier Science Publishers Ltd., Essex, UK, v. 411, n. 22-24, p. 2155–2166, maio 2010. ISSN 0304-3975. Disponível em: <http://dx.doi.org/10.1016/j.tcs.2009.12.021>.

BRODER, A. On the resemblance and containment of documents. In: Proceedings of the Compression and Complexity of Sequences 1997. [S.l.: s.n.], 1997.

BRYANT, R. E. Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput., IEEE Computer Society, Washington, DC, USA, v. 35, n. 8, p. 677–691, ago. 1986. ISSN 0018-9340. Disponível em: <http://dx.doi.org/10.1109/TC.1986.1676819>.

BURMEISTER, P. Formal concept analysis with conimp: Introduction to the basic features. 05 2003.

CLARKE JR., E. M.; GRUMBERG, O.; PELED, D. A. Model Checking. Cambridge, MA, USA: MIT Press, 1999. ISBN 0-262-03270-8.

DAVENPORT, T. H.; PRUSAK, L. Working knowledge: How organizations manage what they know. In: HARVARD BUSINESS SCHOOL PRESS. [S.l.], 1997.

GANTER, B.; STUMME, G.; WILLE, R. Formal concept analysis: foundations and applications. In: VOL. 3626, SPRINGER SCIENCE & BUSINESS MEDIA. [S.l.], 2005.

GANTER, B.; WILLE, R. Formal concept analysis: Mathematical foundations. In: SPRINGER-VERLAG, GERMANY. [S.l.], 1999.

GÉLY, A.; MEDINA, R.; NOURINE, L. About the enumeration algorithms of closed sets. In: KWUIDA, L.; SERTKAYA, B. (Ed.). Formal Concept Analysis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 1–16. ISBN 978-3-642-11928-6.

GÉLY, A. et al. Uncovering and reducing hidden combinatorics in guigues-duquenne bases. In: GANTER, B.; GODIN, R. (Ed.). Formal Concept Analysis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 235–248. ISBN 978-3-540-32262-7.

GODIN, R.; CHAU, T. Comparaison d'algorithmes de construction de hierarchies de classes. In: L'OBJET, V.18. [S.l.], 2000. p. 311–338.

GOLDBERG, D. et al. Using collaborative filtering to weave an information tapestry. In: . New York, NY, USA: ACM, 1992. v. 35, n. 12, p. 61–70. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/138859.138867>.

GUHA, S.; RASTOGI, R.; SHIM, K. Cure: An efficient clustering algorithm for large databases. In: . New York, NY, USA: ACM, 1998. v. 27, n. 2, p. 73–84. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/276305.276312>.

JACCARD, P. Etude comparative de la distribution florale dans une portion des alpes et des jura. Bull Soc Vaudoise Sci Nat, v. 37, p. 547–579, 1901.

JAVABDD. Javabdd. 2017. Disponível em: <http://javabdd.sourceforge.net>.

JELICA, P.; MILO, T.; VELJKO, M. Distributed shared memory: Concept and systems. In: IEEE PARALLEL AND DISTRIBUTED TECHNOLOGY. [S.l.], 1996.

JOSHI, P.; JOSHI, R. Concept analysis for class cohesion. In: PROCEEDINGS OF EUROPEAN CONFERENCE ON SOFTWARE MAINTENANCE AND REENGINEERING (CSMR). [S.l.], 2009. p. 207–240.

KHAN, R. Z.; ALI, M. F. Current trends in parallel computing. In: INTERNATIONAL JOURNAL OF COMPUTER APPLICATIONS (0975 – 8887) VOLUME 59– NO.2. [S.l.], 2012.

MICHAEL, J. F. Computer architecture: Pipelined and parallel processor design. In: JONES AND BARLETT. [S.l.], 1995.

MORAES, N. R. M. et al. Parallelization of the next closure algorithm for generating the minimum set of implication rules. In: Artificial Intelligence Research. [S.l.: s.n.], 2016.

MORAES, N. R. M. de; ZÁRATE, L. E.; FREITAS, H. C. A distributed algorithm for formal concepts processing based on search subspaces. In: ICEIS 2010 - Proceedings of the 12th International Conference on Enterprise Information Systems, Volume 1, DISI, Funchal, Madeira, Portugal, June 8 - 12, 2010. [S.l.: s.n.], 2010. p. 105–111.

NETO, S. M. Formal Concept Analysis Applied In Social Networks With Use Of BDD For High Dimensional Formal Contexts. Dissertação (Mestrado) — Pontifícia Universidade Católica de Minas Gerais (PUC-MG), Belo Horizonte, Minas Gerais, Brazil, 2016.

NETO, S. M.; ZÁRATE, L. E.; SONG, M. A. J. Handling high dimensionality contexts in formal concept analysis via binary decision diagrams. In: INFORMATION SCIENCES. [S.l.: s.n.], 2018. v. 429, p. 361–376.

NILANDER, R.; ZÁRATE, L. Handling large formal contexts with support of distributed systems. In: IN PROCEEDINGS OF IADIS INTERNATIONAL CONFERENCE, V. 22, P. 1-15, 2011. [S.l.], 2011.

RIMSA, A.; ZÁRATE, L. E.; SONG, M. A. Evaluation of different bdd libraries to extract concepts in fca — perspectives and limitations. In: Proceedings of the 9th International Conference on Computational Science: Part I. Berlin, Heidelberg: Springer-Verlag, 2009. (ICCS '09), p. 367–376. ISBN 978-3-642-01969-2. Disponível em: <https://doi.org/10.1007/978-3-642-01970-8_36>.

SALLEB, A.; MAAZOUZI, Z.; VRAIN, C. Mining maximal frequent itemsets by a boolean based approach. In: Proceedings of the 15th European Conference on Artificial Intelligence. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2002. (ECAI'02), p. 385–389. ISBN 978-1-58603-257-9. Disponível em: <http://dl.acm.org/citation.cfm?id=3000905.3000987>.

SHIVAKUMAR, N.; GARCIA-MOLINA, H. Building a scalable and accurate copy detection mechanism. In: Proceedings of the First ACM International Conference on Digital Libraries. New York, NY, USA: ACM, 1996. (DL '96), p. 160–168. ISBN 0-89791-830-4. Disponível em: <http://doi.acm.org/10.1145/226931.226961>.

SILVA, P. et al. Formal concept analysis applied to professional social networks analysis. In: ICEIS. [S.l.], 2017. p. 123–134.

SNELTING, G.; TIP, F. Understanding class hierarchies using concept analysis. ACM Trans. Program. Lang. Syst., ACM, New York, NY, USA, v. 22, n. 3, p. 540–582, maio 2000. ISSN 0164-0925. Disponível em: <http://doi.acm.org/10.1145/353926.353940>.

STUMME, G.; WILLE, R.; WILLE, U. 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. [S.l.], 1998.

TAOUIL, R.; BASTIDE, Y. Computing proper implications. In: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON CONCEPTUAL STRUCTURES-ICCS, PAGES 46–61. [S.l.], 2001.

VIMIEIRO, R. Um estudo de algoritmos para a extração de regras baseados em Análise Formal de Conceitos. Dissertação (Mestrado) — UNIVERSIDADE FEDERAL DE MINAS GERAIS, Belo Horizonte, Minas Gerais, Brazil, 2007.

WILLE, R. Why can concept lattices support knowledge discovery in databases? In: PROCEEDINGS OF THE CONCEPT LATTICES BASED KNOWLEDGE DISCOVERY IN DATABASES WORKSHOP. [S.l.], 2001.

WILLE, R. Dyadic mathematics — abstractions from logical thought. In: K. DENECKE, M. ERNE. [S.l.], 2004. v. 7, p. 167–221.

WILLE, R. Restructuring lattice theory: An approach based on hierarchies of concepts. In: Proceedings of the 7th International Conference on Formal Concept Analysis. Berlin, Heidelberg: Springer-Verlag, 2009. (ICFCA '09), p. 314–339. ISBN 978-3-642-01814-5. Disponível em: <http://dx.doi.org/10.1007/978-3-642-01815-2_23>.

WOLFF, K. E.; YAMEOGO, W. Turing machine representation in temporal concept analysis. In: GANTER, B.; GODIN, R. (Ed.). Formal Concept Analysis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 360–374. ISBN 978-3-540-32262-7.

YEVTUSHENKO, S. System of data analysis concept explorer. In: PROCEEDINGS OF THE 7TH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE KII BOSTON. [S.l.], 2000. p. 74–103.

YEVTUSHENKO, S. Bdd-based algorithms for the construction of the set of all concepts. foundations and applications of conceptual structures. In: Foundations and Applications of Conceptual Structures. Contributions to ICCS. [S.l.: s.n.], 2002. v. 2002, p. 61–73.